
Advancing Hyperparameter Optimization: Foundations, Multiple Objectives and Algorithmic Innovations Informed Through Benchmarking

Lennart Paul Schneider



2025

Advancing Hyperparameter Optimization: Foundations, Multiple Objectives and Algorithmic Innovations Informed Through Benchmarking

Lennart Paul Schneider

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

vorgelegt von Lennart Paul Schneider

München, den 17.03.2025

Erster Berichterstatter: Prof. Dr. Bernd Bischl
Zweiter Berichterstatter: Prof. Dr. Marius Lindauer
Dritter Berichterstatter: Prof. Dr. Matthias Feurer
Tag des Antrags auf Zulassung zum Promotionsverfahren: 01.06.2021
Tag der mündlichen Prüfung: 21.05.2025

For Louisa, the love of my life.

Acknowledgments

First and foremost, I would like to express my gratitude to Prof. Dr. Bernd Bischl for giving me the opportunity to undertake this thesis and for his support throughout the process. I am also very thankful to Prof. Dr. Marius Lindauer and Prof. Dr. Matthias Feurer for kindly agreeing to serve as the second and third reviewers, and to Prof. Dr. Thomas Nagler for participating on the examination panel at my defense.

I gratefully acknowledge financial support from the Mentoring Program of the Faculty of Mathematics, Informatics and Statistics, as well as the Munich Center for Machine Learning and the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter).

Several mentors played a key role in shaping my academic journey. I am especially indebted to Prof. Dr. Jürgen Heller and Florian Wickelmaier, PhD, whose teaching first sparked my interest in the statistical, mathematical, and computational modeling of empirical data. Their guidance laid the foundations of my scientific thinking. I am also grateful to Prof. Dr. Carolin Strobl and Dr. Rudolf Debelak, who welcomed me for a research internship during my undergraduate studies, and to Prof. Dr. Achim Zeileis for supporting my initial research collaborations.

Many thanks go to Florian and Martin, who have been wonderful colleagues since the days of my master's thesis in statistics. Their advice, encouragement, and partnership - beginning when I was a student assistant - helped shape my growth as a researcher and led to many fruitful collaborations.

I would like to extend my heartfelt thanks to my parents, Margit and Klaus, for their unwavering support and for granting me the freedom to follow whichever academic and professional paths inspired me. I also extend this gratitude to Anette, who has supported me in a similar spirit. I am equally grateful to my sisters, Sophia and Katharina, for always being there for me. Katharina, thank you for your guidance as an older sister, for taking on the more challenging hurdles first so that I could follow in your footsteps.

I am also grateful for the soothing companionship of Luna, who often reminds me that the boundary between human and feline intelligence can be surprisingly thin.

Finally, thank you, Louisa, for your unwavering support, patience, and love. Your constant encouragement has helped me stay balanced throughout my research and the writing of this thesis.

Zusammenfassung

Hyperparameteroptimierung (HPO) ist ein essenzieller Bestandteil maschinellen Lernens (ML), der die Performanz von Modellen und weitere zentrale Eigenschaften wie beispielsweise Modellkomplexität maßgeblich beeinflusst. Als rechnerisch anspruchsvolles Black-Box-Optimierungsproblem erfordert HPO effiziente Algorithmen, um optimale Hyperparameterkonfigurationen zu identifizieren. Diese Dissertation erweitert das Forschungsfeld HPO in drei zentralen Dimensionen: grundlegende Erkenntnisse, Optimierung in Szenarien von mehr als einer relevanten Zielfunktion und algorithmische Innovationen durch Benchmarking.

Erstens untersuchen wir Resampling-Strategien zur Schätzung des Generalisierungsfehlers von Modellen und zeigen sowohl theoretisch als auch empirisch, dass ein zufälliges Auswählen von Resampling-Splits über verschiedene Hyperparameterkonfigurationen hinweg die Generalisierung von HPO verbessern kann. Zudem analysieren wir detailliert die Optimierungslandschaften von HPO-Problemen hinsichtlich des Validierungsfehlers und identifizieren charakteristische Merkmale wie geringe Multimodalität und weite Plateaus, welche HPO-Probleme von herkömmlichen Black-Box-Optimierungsproblemen unterscheiden.

Zweitens entwickeln wir neuartige Algorithmen für HPO in multi-kriteriellen Szenarien sowie in Quality-Diversity Szenarien. Wir schlagen einen neuen Ansatz vor, um Modellperformanz und Interpretierbarkeit gleichzeitig zu optimieren, wobei wir letztere anhand von Metriken wie Merkmalsparsamkeit, Sparsamkeit von Interaktionseffekten und Sparsamkeit nicht-monotoner Merkmale quantifizieren. Zudem verknüpfen wir das Forschungsfeld Quality-Diversity Optimierung mit HPO und neuronaler Architektursuche, um innerhalb eines einzigen Optimierungslaufs diverse, aber gleichzeitig leistungsstarke neuronale Architekturen zu entdecken, die unterschiedlichen Hardwarebeschränkungen gerecht werden.

Drittens nutzen wir Benchmarking für algorithmische Innovationen und neue Erkenntnisse. Wir präsentieren YAHPO Gym, eine skalierbare surrogatbasierte Benchmarking-Suite, die sowohl ein- als auch multi-kriterielle HPO-Probleme sowie multi-fidelity Optimierung unterstützt. Mit diesem Framework definieren wir neue Quality-Diversity Probleme, die von HPO-Problemen abgeleitet sind und entwickeln einen innovativen multi-fidelity HPO-Algorithmus, der von den Prinzipien des Programming by Optimization geleitet wird. Darüber hinaus analysieren wir eine bewährte neuronale Architektursuche-Methode und bewerten den Einfluss der einzelnen Komponenten systematisch. Außerdem stellen wir eine Methode zur Konstruktion synthetischer Black-Box-Funktionen vor, die das Ziel hat, dass deren Optimierungslandschaften vorab definierte, bestimmte Eigenschaften aufweisen.

Durch ein vertieftes Verständnis grundlegender Prinzipien von HPO, die Entwicklung neuartiger multi-kriterieller und Quality-Diversity Optimierungsstrategien sowie die Bereitstellung skalierbarer Benchmarking-Tools leistet diese Dissertation einen bedeutenden Beitrag zur Effizienz und Effektivität von HPO für vielfältige ML-Anwendungen.

Summary

Hyperparameter optimization (HPO) is a fundamental aspect of machine learning (ML), directly influencing model performance and adaptability. As a computationally expensive black-box optimization problem, HPO requires efficient algorithms to identify optimal hyperparameter configurations. This thesis advances the field of HPO along three key dimensions: foundational insights, HPO in the presence of more than one objective, and algorithmic innovations through benchmarking.

First, we revisit resampling strategies for performance estimation, demonstrating both theoretically and empirically that reshuffling resampling splits across hyperparameter configurations enhances generalization. Additionally, we conduct an in-depth analysis of HPO validation landscapes, revealing characteristics such as low multimodality and broad plateaus that differentiate them from conventional black-box optimization benchmarks.

Second, we introduce novel algorithms for HPO in multi-objective and quality diversity settings. We propose a new approach for simultaneously optimizing model performance and interpretability, quantifying interpretability through feature sparsity, sparsity of interaction effects, and sparsity of non-monotone features. Furthermore, we bridge the field of quality diversity optimization with HPO, which allows us to discover diverse yet well-performing neural architectures that satisfy varying hardware constraints within a single optimization run.

Third, we use benchmarking to drive algorithmic innovation and insights in HPO. We present YAHPO Gym, a scalable benchmarking suite supporting single-objective, multi-fidelity, and multi-objective HPO via surrogate benchmarks. Using this framework, we define new quality diversity problems inspired by HPO and develop a novel multi-fidelity optimization algorithm guided by programming by optimization principles. Additionally, we ablate a state-of-the-art neural architecture search algorithm to assess the impact of individual components and introduce a systematic approach for constructing synthetic black-box functions that admit specific optimization landscape properties.

By deepening our general understanding of HPO, proposing novel multi-objective and quality diversity optimization strategies, and developing scalable benchmarking tools, this thesis enhances the efficiency and effectiveness of HPO across diverse ML applications.

Contents

I. Introduction and Background	1
1. Introduction	3
1.1. Outline	3
1.2. How to read this Thesis	5
2. Background	7
2.1. Hyperparameter Optimization	9
2.1.1. Single-Objective Hyperparameter Optimization	11
2.1.2. Neural Architecture Search	20
2.1.3. Multi-Fidelity Optimization	22
2.1.4. Hyperparameter Optimization with more than one Objective	25
2.1.5. Parallel Optimization	32
2.1.6. Configuring an Optimization Algorithm	33
2.2. Benchmarking	34
2.2.1. Categorizing Hyperparameter Optimization Benchmarks	35
2.2.2. Landscape Analysis	36
II. Contributions	39
3. Contributions to the Foundations of Hyperparameter Optimization	41
3.1. Reshuffling Resampling Splits can Improve Generalization of Hyperparameter Optimization	43
3.2. HPO x ELA: Investigating Hyperparameter Optimization Landscapes by Means of Exploratory Landscape Analysis	92
4. Multi-Objective and Quality Diversity Hyperparameter Optimization	109
4.1. Multi-Objective Optimization of Performance and Interpretability of Tabular Supervised Machine Learning Models	111
4.2. Tackling Neural Architecture Search with Quality Diversity Optimization	122
5. Algorithmic Innovations Informed Through Benchmarking	153
5.1. YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization	155
5.2. A Collection of Quality Diversity Optimization Problems Derived from Hyperparameter Optimization of Machine Learning Models	195
5.3. Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers	203
5.4. Mutation is all you need	219
5.5. Neural Networks as Black-Box Benchmark Functions Optimized for Exploratory Landscape Features	230
III. Conclusion	243
6. Concluding Remarks	245

IV. References, Excluded Publications and Contributing Publications	249
References	251
Excluded Publications	273
Contributing Publications	275

List of Figures

- 2.1. Illustration of the RBF Kernel. Lighter colors indicate higher values. 14
- 2.2. Branin (log) function and principles of BO. 19
- 2.3. Sampling behavior of different black-box optimizers on Branin (log). 21
- 2.4. Exemplary visualization of an HB bracket starting with $n = 8$ configurations at lowest fidelity $r = 0.125$ and using a halving parameter $\eta = 2.0$ 24
- 2.5. Exemplary Pareto set and front of a two-objective optimization problem. 27
- 2.6. Objectives in HPO. 28
- 2.7. Exemplary quality diversity optimization problem. 31

- 6.1. Normalized interest in the search term “hyperparameter optimization” from January 2016 to March 2025 according to Google Trends. 247

List of Tables

- 2.1. Exemplary HB schedule with a halving parameter of $\eta = 2.0$ 24
- 2.2. Comparison of real-world, synthetic, tabular, and surrogate HPO benchmarks. . . 36

Abbreviations

Throughout this thesis, we use the following common abbreviations:

ML	Machine Learning
AutoML	Automated Machine Learning
HPO	Hyperparameter Optimization
NAS	Neural Architecture Search
HPC	Hyperparameter Configuration
RS	Random Search
GS	Grid Search
EA	Evolutionary Algorithm
BO	Bayesian Optimization
GP	Gaussian Process
SH	Successive Halving
HB	Hyperband
AC	Algorithm Configuration

Part I.

Introduction and Background

1. Introduction

1.1. Outline

Machine Learning (ML) is a branch of Artificial Intelligence that develops algorithms which learn from experience. An algorithm or computer program, i.e., the *machine*, is considered to *learn* when it gains experience from past data, allowing it to improve performance on a given task over time (Mitchell, 1997, Chapter 1, p. 2). In other words, if an algorithm improves its ability to complete a task based on prior interactions or observations, it exhibits learning.

In recent years, ML has provided radically new approaches to application fields such as computer vision (Krizhevsky et al., 2012; He et al., 2016; Dosovitskiy et al., 2021), natural language processing (Vaswani et al., 2017; Devlin et al., 2019; Brown et al., 2020), game playing (Silver et al., 2016) and drug discovery (Vamathevan et al., 2019). ML, however, is not one single algorithm. Instead, many different learning algorithms exist, each varying in how they approach solving a task and how learning from data is performed. Crucially, each algorithm has its own *hyperparameters* - control parameters that determine how the algorithm learns from data and influence the final model that will be used to solve a task.

Choosing ML algorithms and their hyperparameters so they perform well on a task is a non-trivial problem and requires expert knowledge and manual trial and error over many iterations. This hinders the adoption of ML (McKinsey & Company, 2018; Baier et al., 2019; Shankar et al., 2022; Paleyes et al., 2022).

The field of Automated Machine Learning (AutoML) is concerned with democratizing ML. It aims to make ML accessible, efficient, and scalable by reducing the need for manual intervention and expert knowledge. A core component of AutoML is given by Hyperparameter Optimization (HPO) that subsumes different methods to automatically identify well-performing hyperparameter configurations (HPCs) of an ML algorithm for a task at hand.

HPO can be characterized by different dimensions, including the search space, type and number of objective functions, evaluation methodology, optimization algorithms, and computational efficiency. The search space defines the hyperparameters to optimize ranging from low-dimensional numeric to high-dimensional mixed spaces with categorical parameters and potentially hierarchical dependencies. In full AutoML pipelines, this extends beyond the hyperparameters of a single algorithm to include algorithm selection and pre- and post-processing steps.

The primary goal of HPO is to optimize an ML model's performance, typically measured by an estimate of its generalization error. To ensure unbiased estimation of a model's generalization error, resampling techniques such as holdout and cross-validation need to be used to mimic evaluating models on unseen data. Since the generalization error lacks an analytical expression and gradient information, HPO is fundamentally a *black-box* optimization problem. The costly nature

of model evaluation further complicates it, making HPO *expensive*, especially in deep learning, where evaluating a single HPC can take hours or even days.

Beyond performance, real-world ML applications often require optimizing multiple objectives such as resource efficiency or measures related to the interpretability of models. This has made multi-objective HPO an increasingly important and actively researched field.

Over the past decade, the AutoML community has focused on efficient HPO methods, moving beyond grid and random search to adaptive strategies like Bayesian Optimization. Multi-fidelity optimization has further improved efficiency by using cheaper approximations of the objective function.

Since HPO algorithms themselves are modular and configurable, and due to the need to empirically validate improvements of new algorithms, the HPO and AutoML community has demonstrated the necessity for representative benchmarks. The effectiveness of empirical research and benchmarking in HPO relies on access to diverse, representative, and reproducible benchmarks that are easy and ideally cheap to use and accurately reflect real-world HPO problems.

Despite significant advancements in HPO over the past decade, several challenges remain. The optimization landscape of HPO problems is still not well understood. The role of resampling strategies in estimating generalization errors and their impact on the HPO process further requires deeper investigation. Moreover, real-world applications increasingly demand efficient optimization beyond predictive performance, incorporating objectives such as interpretability, fairness, and computational efficiency. However, multi-objective or constrained optimization might not always be the most efficient approach as they try to approximate the whole Pareto front or only return a single solution satisfying a constraint. Finally, benchmarking in HPO remains a relevant topic, as existing benchmarks often lack scalability, representativeness, and reproducibility.

This thesis addresses these challenges through three key contributions:

1. Foundations of HPO: We investigate the impact of resampling strategies and using the same or different train validation splits for each HPC on generalization error. We further analyze the optimization landscape of HPO and contrast its properties with traditional black-box optimization problems.
2. Multi-objective and quality diversity HPO: We develop novel algorithms that extend HPO beyond single-objective optimization, introducing strategies to balance predictive performance with other objectives such as interpretability of models and resource efficiency.
3. Benchmarking in HPO: We introduce a large-scale surrogate-based benchmarking suite that offers efficient, reproducible, and scalable evaluation tools for HPO research. To showcase its practicality, we apply it to the algorithm configuration of a multi-fidelity HPO method. Additionally, we highlight the value of ablation studies in gaining deeper insights into algorithm components.

1.2. How to read this Thesis

This is a thesis *by publication*, which combines several works published in leading venues following research questions concerned with advancing HPO. Before listing the contributed publications, the current Part I sets the general stage, and we already briefly outlined ML and HPO in the previous Section 1.1. In the following Chapter 2 we provide a general background on ML and HPO and introduce notation and concepts of the methods underlying HPO in ML. This part further contains a systematic overview of recent developments in HPO, providing motivation for the contributions of this thesis at a higher level of abstraction. Contributed publications are then categorized into three groups and presented in Part II: In Chapter 3, we present contributions concerned with the foundations of HPO. In Chapter 4, we present contributions to HPO in the presence of more than one objective, namely multi-objective and quality diversity optimization. In Chapter 5, we focus on benchmarking in HPO. Within each chapter introducing a group of contributions, we briefly motivate the contributions and summarize them. Finally, we discuss our contributions and the field of HPO and its future in Chapter 6 in Part III concluding the thesis. Some publications of the author have not been included in this thesis. We briefly list them in Part IV along with other references and the contributed publications selected for this thesis.

2. Background

The following background closely follows the terminology and notation introduced in [Bischl et al. \(2023\)](#). The goal of supervised ML is to infer a model based on training data whose predictions generalize well to unseen test data. To formalize this, let $\mathcal{D} := \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$ be a labeled dataset composed of n observations. Each observation $(\mathbf{x}^{(i)}, y^{(i)})$ consists of a p dimensional feature vector $\mathbf{x}^{(i)} \in \mathcal{X}$ and a target label $y^{(i)} \in \mathcal{Y}$. We will not cover tensor data or multi-label outputs or the unsupervised setting to keep the notation simple. We assume that \mathcal{D} is an i.i.d. sample from an underlying, unknown distribution, i.e., $\mathcal{D} \sim (\mathbb{P}_{xy})^n$. An ML model f is a function that maps each feature vector to a prediction $f : \mathcal{X} \rightarrow \mathbb{R}^g$. The two most popular tasks of supervised ML are regression and classification. For regression, the prediction is a single number and the codomain is given by the real numbers \mathbb{R} . For the classification of g classes, the prediction is usually given by g decision scores or posterior probabilities, and the codomain is given by \mathbb{R}^g . For binary classification ($\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, 1\}$), the prediction function is often scalar-valued ($g = 1$), representing the estimated probability of the positive class.

To obtain a model, a learning algorithm or inducer \mathcal{I} is trained on a dataset and outputs the model or prediction function. Almost any learning algorithm can be configured by hyperparameters that influence this training process. To formalize this, let $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ denote hyperparameters that configure the inducer. For now, we simply acknowledge that hyperparameters $\boldsymbol{\lambda}$ influence the inducers' behavior. In [Section 2.1](#), we introduce hyperparameters and their optimization in more detail, after having covered the basics of supervised ML and performance evaluation.

The inducer maps a dataset to a model \hat{f} , or its parameter vector $\hat{\boldsymbol{\theta}}$:

$$\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \rightarrow \mathcal{H}, \quad (\mathcal{D}, \boldsymbol{\lambda}) \mapsto \hat{f} \quad (2.1)$$

Here, $\mathbb{D} := \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n$ is the set of all datasets, $\boldsymbol{\Lambda}$ is the domain of hyperparameters and \mathcal{H} denotes the hypothesis space of prediction functions. Note that model parameters $\hat{\boldsymbol{\theta}}$ are the result of the training process, whereas hyperparameters $\boldsymbol{\lambda}$ govern this training process on a higher level.

To quantify how close the model output is to the true label, we rely on a loss function $L : \mathcal{Y} \times \mathbb{R}^g \rightarrow \mathbb{R}_0^+$. For example, a well-known loss function for binary classification is given by the log loss: $L_{\log}(y, f(\mathbf{x})) = -[y \log(f(\mathbf{x})) + (1 - y) \log(1 - f(\mathbf{x}))]$, assuming $f(\mathbf{x})$ to be the posterior probability for the positive class. A widely used training concept is empirical risk minimization (ERM). In general, we would like to find the model \hat{f} or parameter vector $\hat{\boldsymbol{\theta}}$ that in expectation performs best for data from the distribution (\mathbb{P}_{xy}) , minimizing the following risk:

$$\mathcal{R}(f) := \mathbb{E}_{(\mathbf{x}, y) \sim \mathbb{P}_{xy}} [L(y, f(\mathbf{x}))] \quad (2.2)$$

In practice, however, we can only approximate this expectation given our finite training data \mathcal{D} available. This turns the risk function into the empirical risk function. Still, the conceptual goal remains the same, and we want to find the risk-optimal model \hat{f} or parameter vector $\hat{\theta}$:

$$\mathcal{R}_{\text{emp}}(f) := \frac{1}{n} \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})), \quad \hat{f} = \arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) \quad (2.3)$$

HPO in contrast operates on a higher, second level and is concerned with the problem of identifying the optimal HPC of the inducer. As we seek to optimize an unbiased expected performance of a model produced by an inducer \mathcal{I} , we need to evaluate the model on new, unseen test data. Again, assuming a point-wise loss for simplicity¹ $L(y, f(\mathbf{x}))$, the generalization error, i.e., the expected performance of a model produced by an inducer configured by an HPC and trained on training data of a given size, states as:

$$\text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, L) = \mathbb{E}_{\mathcal{D}_{\text{train}}, (\mathbf{x}, y) \sim \mathbb{P}_{xy}} [L(y, \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D}_{\text{train}})(\mathbf{x}))] \quad (2.4)$$

Note that the expectation is taken both over the training dataset $\mathcal{D}_{\text{train}}$ and test point (\mathbf{x}, y) . Again, this quantity cannot be computed in practice, and we usually must estimate the generalization error from a single given dataset \mathcal{D} . We therefore fall back to an empirical estimate of the generalization error based on a resampling strategy.

Resampling strategies split the available data into training and test sets. The inducer configured by an HPC is then (potentially repeatedly) trained on a train set and the performance of the model is evaluated on the test set. For a simple holdout resampling based on a single random split, $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ can be represented as index vectors $J_{\text{train}} \in \{1, \dots, n\}^{n_{\text{train}}}$ and $J_{\text{test}} \in \{1, \dots, n\}^{n_{\text{test}}}$ that partition the dataset. We can proceed to define the holdout estimator for Equation (2.4) as:

$$\widehat{\text{GE}}_{J_{\text{train}}, J_{\text{test}}}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, L) = \frac{1}{n_{\text{test}}} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{test}}} L(y, \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D}_{\text{train}})(\mathbf{x})) \quad (2.5)$$

Formally, we can identify any resampling strategy with a vector of corresponding splits where each split $\mathcal{J} = ((J_{\text{train},1}, J_{\text{test},1}), \dots, (J_{\text{train},B}, J_{\text{test},B}))$ where $J_{\text{train},i}, J_{\text{test},i}$ again represent index vectors and B is the number of splits or cross-validation folds. This results in an estimator for Equation (2.4) as:

$$\widehat{\text{GE}}(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{J}, L) = \text{agr} \left(\widehat{\text{GE}}_{J_{\text{train},1}, J_{\text{test},1}}(\mathcal{I}, \boldsymbol{\lambda}, |J_{\text{train},1}|, L), \dots, \widehat{\text{GE}}_{J_{\text{train},B}, J_{\text{test},B}}(\mathcal{I}, \boldsymbol{\lambda}, |J_{\text{train},B}|, L) \right) \quad (2.6)$$

Here, the aggregation function $\text{agr} : \mathbb{R}_0^+ \times \dots \times \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ is often simply the mean. Note that formally, Equation (2.4) requires as input the size of the training set, n_{train} . For the general resampling-based estimator in Equation (2.6), we do not explicitly state the training set size. Rather the implicit assumption is that it holds that $n_{\text{train}} \approx n_{\text{train},1} \approx \dots \approx n_{\text{train},B}$. A popular variant is k -fold cross-validation where the available data is split into k partitions of the same size and for each partition, the model is evaluated on the complement. For an overview of different loss

¹Note that this loss or in general performance metric must not be the same as the one used for the inner ERM problem. Moreover, for notation suitable for a more general case of a set-based performance metric such as the area under the receiver operating characteristic curve, we refer to [Bischl et al. \(2023\)](#).

2.1 Hyperparameter Optimization

functions and performance metrics, we refer to [Bischl et al. \(2023\)](#). For a discussion of different resampling methods, we refer to [Boulesteix et al. \(2008\)](#), [Hastie et al. \(2009, Chapter 7\)](#), and [Bischl et al. \(2012b\)](#).

2.1. Hyperparameter Optimization

Almost any ML algorithm can be configured by hyperparameters. These hyperparameters influence the training procedure and the resulting generalization performance of the learning algorithm. For example, in neural networks, the learning rate determines how quickly the model updates weights during training. Values too small can lead to slow convergence, whereas values too large can lead to divergence ([Goodfellow et al., 2016, Chapter 8](#)). Decision trees rely on a maximum depth hyperparameter that controls model complexity to prevent overfitting ([Breiman et al., 1984](#)). Random forests that ensemble decision trees can for example be configured by the number of trees and the maximum number of features considered for a split that impact bias and variance ([Breiman, 2001](#)). Gradient boosting systems like XGBoost ([Chen and Guestrin, 2016](#)) involve hyperparameters such as learning rate, maximum depth, and regularization strength. Support vector machines employ regularization hyperparameters and can be used with different kernel functions, which determine the trade-off between margin maximization and error and influence the decision boundary ([Cortes and Vapnik, 1995](#)).

Identifying optimal hyperparameters is essential for good model performance ([Bergstra et al., 2011](#); [Feurer and Hutter, 2019](#); [Yang and Shami, 2020](#); [Bischl et al., 2023](#)) and many ML algorithms exhibit strong potential to benefit from HPO ([Probst et al., 2019](#); [van Rijn and Hutter, 2018](#)).

The general HPO problem is defined as:

$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \tilde{\boldsymbol{\Lambda}}} c(\boldsymbol{\lambda}) = \arg \min_{\boldsymbol{\lambda} \in \tilde{\boldsymbol{\Lambda}}} \widehat{\text{GE}}(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{J}, L) \quad (2.7)$$

Here, $c(\boldsymbol{\lambda})$ is a compact notation for the estimated generalization error when the inducer \mathcal{I} , resampling \mathcal{J} , and performance metric or loss function L are fixed. Expressing Equation (2.7) in words, we optimize the estimated generalization error of a learner $\mathcal{I}_{\boldsymbol{\lambda}}$, with respect to an HPC $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_l)$. Note that the objective is usually referred to as the validation error or loss - although we introduced the resampling estimate in the previous section based on a train and test split. This terminology originates from the fact that to obtain an unbiased performance estimate of the inducer configured by the optimal HPC, we require nested resampling which makes use of an outer test set ([Bischl et al., 2023](#)).

The search space $\tilde{\boldsymbol{\Lambda}} \subset \boldsymbol{\Lambda}$ contains all hyperparameters that are to be optimized and their respective ranges:

$$\tilde{\boldsymbol{\Lambda}} = \tilde{\boldsymbol{\Lambda}}_1 \times \tilde{\boldsymbol{\Lambda}}_2 \times \dots \times \tilde{\boldsymbol{\Lambda}}_l \quad (2.8)$$

Here $\tilde{\boldsymbol{\Lambda}}_i$ is a bounded subset of the domain of the i -th hyperparameter $\boldsymbol{\Lambda}_i$. Hyperparameters can be continuous (e.g., learning rate), discrete (e.g., number of layers), or categorical (e.g., choice of activation function). This results in $\tilde{\boldsymbol{\Lambda}}$ potentially being a so-called mixed space. Moreover, certain hyperparameters can depend on other hyperparameters. This can result in a tree-like hierarchical search space with dependencies that can be represented by directed acyclic graphs. For example, consider the choice of first-order optimizer as a hyperparameter which we optimize but also parameters of the optimizer, e.g., stochastic gradient descent ([Robbins and Monro, 1951](#);

Bottou et al., 2018) and its learning rate or Adam (Kingma and Ba, 2015) and its learning rate and β_1 , β_2 parameters used for computing running averages of the gradient and its square. Here, the β_1 , β_2 hyperparameters depend on the optimizer choice Adam.

In general, $c(\boldsymbol{\lambda})$ in Equation (2.7) is a black-box function: Usually, there is no closed-form mathematical expression and analytic gradient information is not available. Moreover, evaluating $c(\boldsymbol{\lambda})$ can take significant time due to the resampling procedure. Therefore, the minimization of $c(\boldsymbol{\lambda})$ forms an *expensive black-box* optimization problem. As such, this generally rules out the usage of standard first-order or second-order optimization methods that require gradients or additional (approximated) Hessians. At this point, we do have to note, however, that gradient-based methods for HPO do exist. These methods rely on purely numeric search spaces and derive the hypergradient based on the implicit function theorem (Pedregosa, 2016) or via iterative differentiation making use of automatic differentiation (Maclaurin et al., 2015; Franceschi et al., 2017). Coming back to treating the HPO problem as a black-box, due to the potentially high cost of evaluation, sample efficiency (i.e., identifying a well-performing configuration with only a few evaluations) is a highly desirable property that many derivative-free meta-heuristic optimization techniques lack. Finally, as $c(\boldsymbol{\lambda})$ is a resampling-based estimate of the true generalization error and due to the non-determinism of learning algorithms in general, $c(\boldsymbol{\lambda})$ can be considered a stochastic objective. Looking at the related literature, however, Bischl et al. (2023) point out that many HPO algorithms may overlook this issue or address it by simply assuming that sufficient resampling replications will average out the randomness. Note that to ensure an unbiased estimate of the performance of the inducer configured by the optimal HPC, it is necessary to use nested resampling (Cawley and Talbot, 2010; Bischl et al., 2023). Without nested resampling, performance estimates may be overly optimistic, as the same data used to select the best HPC may inadvertently influence the final evaluation (Bischl et al., 2023).

The remaining parts of this background chapter are structured as follows. In Section 2.1.1 an overview of widely used algorithms for tackling single-objective HPO as formulated in Equation (2.7) is presented with an emphasis on Bayesian Optimization (BO). In Section 2.1.2, the optimization problem of Neural Architecture Search (NAS) is introduced and formulated as a special case of HPO. As training neural networks is costly, many works of the related literature in the last few years have been concerned with making HPO and NAS more efficient. In Section 2.1.3 we briefly introduce the concept of multi-fidelity optimization suited to speed up HPO and NAS and present widely used algorithms. So far, our introduction of HPO involved a single performance metric, i.e., minimizing the estimated generalization error. However, multiple objectives can be of interest when deploying models in practice. While constraints can arise in the form of other objectives to consider that affect the feasible set of HPCs, there is also the need to simultaneously consider more than one objective which is to be minimized. For example, consider the resource usage of a model depending on its size, or the complexity or simplicity of the prediction function of the model which is directly tied to concepts related to model interpretability. In practice, we can be interested in minimizing both the generalization error and complexity of a model. Since low resource usage or low complexity often conflicts with achieving high performance, this necessitates multi-objective HPO. We provide a brief introduction to this topic in Section 2.1.4, along with an overview of popular optimization algorithms. As an alternative to multi-objective optimization, we further briefly introduce the comparably novel field of quality diversity optimization. Here, the goal is not to simultaneously optimize multiple objectives, but instead optimize a single objective while obtaining diverse solutions for other so-called behavior functions. Finally, for the sake of completeness, we briefly discuss algorithms and methods suitable to speed up HPO via

2.1 Hyperparameter Optimization

parallelization in Section 2.1.5. Several open-source HPO toolboxes such as SMAC (Hutter et al., 2011; Lindauer et al., 2022), Spearmint (Snoek et al., 2012), Hyperopt (Bergstra et al., 2013), Optuna (Akiba et al., 2019), Dragonfly (Kandasamy et al., 2020), BoTorch (Balandat et al., 2020) or HEBO (Cowen-Rivers et al., 2022) provide efficient implementations of often both single-objective and multi-objective HPO techniques, usually focusing on BO algorithms and often also supporting multi-fidelity evaluations. As the focus of this thesis is not software, we will not provide an overview of these HPO toolboxes but refer to Bischl et al. (2023) and Karl et al. (2023) for an overview. Similarly, since this thesis focuses specifically on HPO, we will not go detailed into general AutoML systems such as Auto-WEKA (Thornton et al., 2013) or Auto-sklearn (Feurer et al., 2015a), but instead refer to Part II of Hutter et al. (2019), which provides a comprehensive introduction of traditional AutoML systems.

2.1.1. Single-Objective Hyperparameter Optimization

The problem of single-objective HPO is directly given by Equation (2.7). The two core challenges are (1) $c(\boldsymbol{\lambda})$ being an expensive black-box function and (2) the potential hierarchical mixed search space $\tilde{\Lambda}$. The expensiveness of function evaluations naturally calls for so-called sample-efficient algorithms. An algorithm is sample efficient if few function evaluations are needed to identify well-performing configurations. Below, we introduce popular HPO algorithms for vanilla single-objective HPO and highlight similarities and differences. In later sections, we will see that most of the algorithms can be extended to different settings, such as multi-fidelity optimization and multi-objective optimization.

Popular Algorithms

Most HPO algorithms are iterative and follow the same core principle: They iteratively *propose* HPCs $\boldsymbol{\lambda}^+$ and *evaluate* them. Let $\mathcal{A} = ((\boldsymbol{\lambda}^{(1)}, c(\boldsymbol{\lambda}^{(1)})), (\boldsymbol{\lambda}^{(2)}, c(\boldsymbol{\lambda}^{(2)})), \dots)$, denote the archive we store configurations and their objective values with $\mathcal{A}^{[t+1]} = \mathcal{A}^{[t]} \cup (\boldsymbol{\lambda}^+, c(\boldsymbol{\lambda}^+))$. In general, iterative or sequential HPO algorithms therefore follow the scheme as depicted in Algorithm 1. The core difference between algorithms is how they propose configurations. Note that Line 2 and Line 7 and 8 can differ in practice, i.e., how to terminate HPO and what should be returned as the final result. For introductory discussions on these topics, we refer to Bischl et al. (2023). HPO and in general black-box optimization algorithms can be described based on their search behavior. An important concept is given by *exploration* vs. *exploitation* (Feurer and Hutter, 2019; Bischl et al., 2023) which can be traced back as early as foundational work on sequential decision-making such as multi-armed bandits (Robbins, 1952). Exploration describes the behavior of an optimizer to propose new configurations in unexplored search space areas to obtain information that reduces uncertainty. Exploitation refers to evaluating configurations that we believe to perform well or are in areas of configurations that perform well, which potentially allows for incremental performance improvement.

Grid Search. A grid search (GS) simply discretizes the search space. For each hyperparameter domain $\tilde{\Lambda}_i$, a finite set of configurations is considered and the full grid is constructed by taking the Cartesian product. For numeric domains, this can be achieved by picking values equidistant between the lower and upper bound. For categorical domains, an exhaustive inclusion of all

Algorithm 1 Pseudocode for Sequential HPO

```

1:  $\mathcal{A} \leftarrow \emptyset$ 
2: while not terminated do
3:    $\lambda^+ \leftarrow \text{PROPOSECONFIGURATION}(\tilde{\Lambda}, \mathcal{A})$ 
4:    $c(\lambda^+) \leftarrow \text{EVALUATECONFIGURATION}(\lambda^+)$ 
5:    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\lambda^+, c(\lambda^+))\}$ 
6: end while
7:  $(\lambda^*, c(\lambda^*)) \leftarrow \arg \min_{(\lambda, c(\lambda)) \in \mathcal{A}} c(\lambda)$ 
8: return  $(\lambda^*, c(\lambda^*))$ 

```

potential values is possible, or the inclusion of a subset. GS can naturally handle mixed and hierarchical search spaces. However, it is affected by the curse of dimensionality, as the grid grows exponentially with the dimensionality of the search space when keeping the resolution per parameter constant. This implies that grids with many configurations are needed to properly cover search spaces resulting in poor sample efficiency.

Random Search. In its simplest variant, a random search (RS) proposes HPCs by sampling each hyperparameter independently of the others following a given distribution, e.g., the uniform one. RS naturally can handle mixed and also hierarchical search spaces by respecting dependencies, e.g., in a post-hoc manner. However, similar to GS, it suffers from the curse of dimensionality, as sample coverage of high-dimensional search spaces needs many configurations. In contrast to GS, however, RS has the advantage that marginal per hyperparameter more unique values will be generated (Bergstra et al., 2011). This results in slightly higher efficiency as, e.g., in the case of some hyperparameters having no effect at all on the objective function more unique values will have been tried for the other relevant ones.

Population Based Algorithms. As the name suggests, population-based algorithms work by maintaining a population of configurations over generations. Examples are given by evolutionary algorithms (EAs) and evolution strategies such as CMA-ES (Hansen and Ostermeier, 1996; Hansen, 2023) or MIES (Li et al., 2013) but also particle swarm optimization (Kennedy and Eberhart, 1995) or differential evolution (Bilal et al., 2020). Inspired by biological evolution, these algorithms start with an initial population of configurations, from which offspring are generated. In the case of EAs, this is done via parent selection and recombination of parents via so-called mutation and crossover operators. In the case of evolution strategies that are based on a sampling procedure such as CMA-ES, offspring are generated after refining a sampling distribution based on well-performing individuals of the population. Both, EAs and evolution strategies usually also involve elitist mechanisms to only keep top-performing individuals in the population when moving from one generation to another. For an in-depth introduction to population-based algorithms and their building blocks, e.g., parent selection, mutation and crossover, and survival selection, we refer to Simon (2013). While evolution strategies often can perform well for HPO and are more efficient than RS and GS they are not sample efficient and usually require many iterations to identify good configurations (Bischi et al., 2023).

2.1 Hyperparameter Optimization

Bayesian Optimization. So far, all algorithms introduced are sampling-based and model-free. BO originates from a different idea of modeling the unknown black-box function $c(\boldsymbol{\lambda})$ that is to be optimized. As BO can be considered the de-facto state-of-the-art algorithm for vanilla HPO and has been a strong focus of the last years of research, we will introduce its core ideas and components in more detail. We refer to Garnett (2023) for a more in-depth introduction to BO.

Assume we want to optimize an expensive black-box function without analytical expression and gradient information. A reasonable approach is to have a prior belief about how the function could look like, collect data by evaluating configurations and update our prior belief in a Bayesian manner. Formally, this implies building a probabilistic model of the function based on observed data and iteratively proposing new points that based on our current belief of the function provide utility for optimizing the function and refining our belief. More specifically, BO consists of two core building blocks: (1) a so-called surrogate model that is used to model the black-box function based on observed data, and (2) a so-called acquisition function that is defined on the surrogate and quantifies the utility of candidate points to guide the next proposal of points.

The standard surrogate model of BO for purely numeric search spaces is given by a Gaussian Process (GP). A GP is a flexible non-parametric regression model. In the following, we introduce it using standard supervised learning notation established above following Williams and Rasmussen (2006). Readers familiar with GPs may skip the following introduction.

Formally, a GP is a random process where any point $\mathbf{x} \in \mathcal{X}$ is assigned a random variable $f(\mathbf{x})$ and the joint distribution of any finite number of these random variables is Gaussian:

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}). \quad (2.9)$$

Here, $\mathbf{f} = (f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)}))$, $\boldsymbol{\mu} = (m(\mathbf{x}^{(1)}), \dots, m(\mathbf{x}^{(n)}))$ and $\mathbf{K}_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. $m : \mathcal{X} \rightarrow \mathbb{R}$ is the mean function commonly set to zero and $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a positive definite kernel.

Given a training dataset with noise-free function observations \mathbf{f} at inputs \mathbf{X} , a GP prior can be updated to a posterior $p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f})$ that allows for predictions at new test inputs \mathbf{X}_* .

As by definition of a GP the joint distribution of observed values \mathbf{f} and predictions \mathbf{f}_* is again Gaussian, one can obtain the posterior predictive distribution under the zero mean GP as

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \quad (2.10)$$

$$\boldsymbol{\mu}_* = \mathbf{K}_*^\top \mathbf{K}^{-1} \mathbf{f}, \quad (2.11)$$

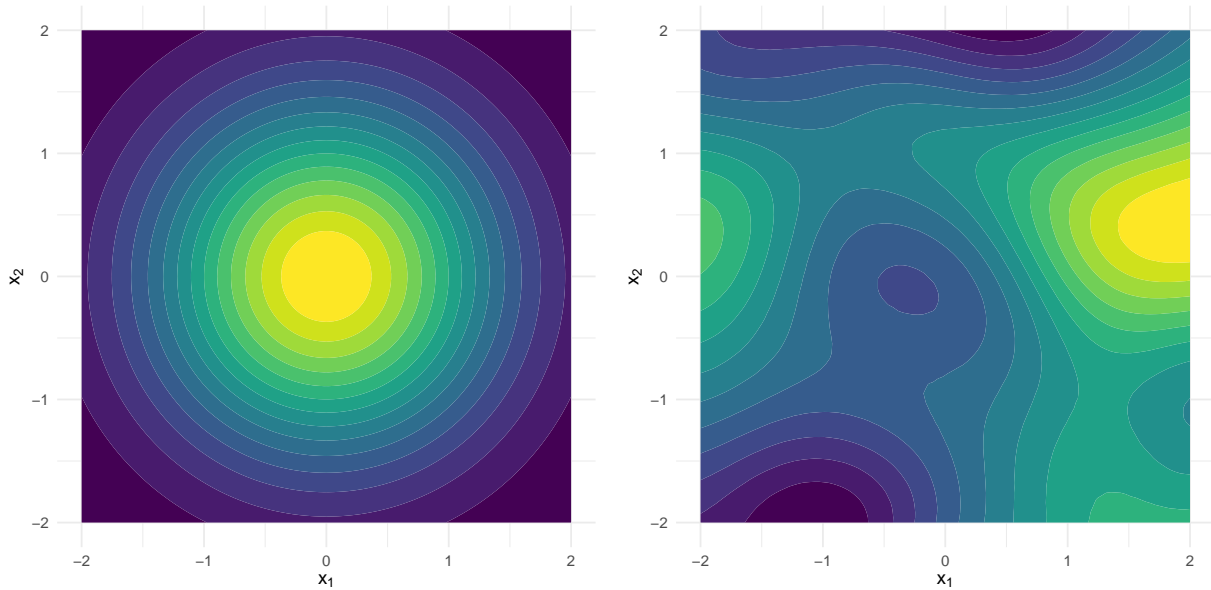
$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^\top \mathbf{K}^{-1} \mathbf{K}_*, \quad (2.12)$$

where \mathbf{K} is the $n \times n$ kernel matrix on training data, \mathbf{K}_* is the $n \times n_*$ kernel matrix of training data and test inputs and \mathbf{K}_{**} is the $n_* \times n_*$ kernel matrix of test inputs.

A kernel function measures the similarity between two points. We will not go into detail with respect to the properties of kernel functions but only provide some examples. The radial basis function (RBF) kernel with a single lengthscale parameter ℓ is given by:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_s^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell}\right),$$

where σ_s^2 is the signal variance.



(a) RBF kernel in two dimensions $k((x_1, x_2), (0, 0))$. (b) Sample from a zero-mean GP prior with RBF kernel.

Figure 2.1.: Illustration of the RBF Kernel. Lighter colors indicate higher values.

For a visualization of the contour lines of RBF kernel in two dimensions with a signal variance $\sigma_s^2 = 1$ and a lengthscale parameter $\ell = 1$, see Figure 2.1a. We can observe a Gaussian bell centered around the reference point $\mathbf{x}' = (0, 0)$ with high kernel (similarity) values close to 1 for points close to the reference point. Figure 2.1 visualizes a sample from a zero-mean GP prior using said RBF kernel.

The Matérn kernel is a generalization of the RBF kernel that introduces a smoothness parameter ν , allowing control over the function's differentiability. A common choice is $\frac{5}{2}$ or $\frac{3}{2}$:

$$k_\nu(\mathbf{x}, \mathbf{x}') = \sigma_s^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} \|\mathbf{x} - \mathbf{x}'\|}{\ell} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu} \|\mathbf{x} - \mathbf{x}'\|}{\ell} \right), \quad (2.13)$$

where K_ν is the modified Bessel function of the second kind, and Γ is the gamma function.

To allow further flexibility in accounting for different variations in each input dimension, it is common to use automatic relevance determination (ARD) kernels that incorporate different lengthscale parameters for each parameter dimension.

One central property of kernel functions is whether they only depend on the relative position of its two inputs, and not on their absolute location. This property is referred to *stationary* which has implications on the properties of the GP when modeling a function. For example, a GP with a stationary kernel holds the assumption that the mean and smoothness of the function are consistent throughout the space it is defined on.

2.1 Hyperparameter Optimization

Moving on to the scenario of noisy objective function values $y = f(\mathbf{x}) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_f^2)$ (i.e., i.i.d and homoscedastic noise), the posterior predictive distribution under the zero mean GP is obtained as

$$p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mu_*, \Sigma_*), \quad (2.14)$$

$$\mu_* = \mathbf{K}_*^\top (\mathbf{K} + \sigma_f^2 \mathbf{I}_n)^{-1} \mathbf{y}, \quad (2.15)$$

$$\Sigma_* = \mathbf{K}_{**} - \mathbf{K}_*^\top (\mathbf{K} + \sigma_f^2 \mathbf{I}_n)^{-1} \mathbf{K}_* \quad (2.16)$$

Trivially, for a single test point \mathbf{x}^* , the posterior predictive distribution is

$$p(f_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mu_*, \sigma_*), \quad (2.17)$$

$$\mu_* = \mathbf{k}_*^\top (\mathbf{K} + \sigma_f^2 \mathbf{I}_n)^{-1} \mathbf{y}, \quad (2.18)$$

$$\sigma_* = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_*^\top (\mathbf{K} + \sigma_f^2 \mathbf{I}_n)^{-1} \mathbf{k}_*, \quad (2.19)$$

where \mathbf{k}^* is the n dimensional vector of covariances between the test point \mathbf{x}^* and the training points \mathbf{X} .

Kernel and likelihood parameters of the GP are typically obtained via maximum likelihood or maximum a posteriori estimates. GPs are potent surrogate models due to their flexible modeling capabilities, strong extrapolation capability, and well-calibrated uncertainty estimates.

To keep notation standard, we introduced GPs with conventional supervised learning notation using features $\mathbf{x} \in \mathcal{X}$ and targets $y \in \mathcal{Y}$. Translating into the context of HPO, we generally assume noisy objective function evaluations, and we now model HPCs and their estimated generalization error which we obtain in the archive $\mathcal{A} = ((\boldsymbol{\lambda}^{(1)}, c(\boldsymbol{\lambda}^{(1)})), \dots, (\boldsymbol{\lambda}^{(t)}, c(\boldsymbol{\lambda}^{(t)})))$. Therefore, $\boldsymbol{\lambda}$ now takes the role of \mathbf{x} and $c(\boldsymbol{\lambda})$ the role of y .

Based on the surrogate model as our belief of the functional relationship between hyperparameters and their performance, BO makes use of an acquisition function defined on the posterior predictive distribution to intelligently propose the next candidate point that is to be evaluated.

Popular acquisition functions are given the Expected Improvement (EI; Moćkus 1974; Jones et al. 1998), Probability of Improvement (PI; Kushner 1964), and Lower Confidence Bound (LCB, also better known as the UCB in the case of maximization; Srinivas et al. 2010). The EI measures for a candidate HPC $\boldsymbol{\lambda}$ how much we can expect it to improve over the current best-observed function value of the so-called incumbent, given that we only allow for improvement (notice the max operator). Formally,

$$\alpha_{\text{EI}}(\boldsymbol{\lambda}) := \mathbb{E}_{C(\boldsymbol{\lambda})} [\max(c_{\min} - C(\boldsymbol{\lambda}), 0)] \quad (2.20)$$

where c_{\min} is the best observed function value so far and $C(\mathbf{x})$ is the random variable associated with the HPC $\boldsymbol{\lambda}$ following the posterior predictive distribution of the surrogate model. In contrast to the original black-box function c , the EI is cheap to compute, and optimizing it gives us the next candidate HPC for evaluation. Note that, formally, the EI as stated in Equation (2.20) is only sensible in the context of noise-free observations. Nevertheless, it is frequently used as the default acquisition function. For a discussion and a literature survey of how to extend EI to noisy observations, see Garnett (2023, Chapter 7).

Given a GP as a surrogate model (or rather any model with a normal posterior predictive distribution), the EI can be computed in closed form as

$$\alpha_{\text{EI}}(\boldsymbol{\lambda}) = (c_{\min} - \mu(\boldsymbol{\lambda})) \Phi\left(\frac{c_{\min} - \mu(\boldsymbol{\lambda})}{\sigma(\boldsymbol{\lambda})}\right) + \sigma(\boldsymbol{\lambda}) \phi\left(\frac{c_{\min} - \mu(\boldsymbol{\lambda})}{\sigma(\boldsymbol{\lambda})}\right) \quad (2.21)$$

Here, Φ and ϕ denote the cumulative density function and the probability density function of the standard normal distribution. $\mu(\boldsymbol{\lambda})$ and $\sigma(\boldsymbol{\lambda})$ denote the mean and standard deviation of the posterior predictive distribution under the GP.

The PI measures the probability of improvement of a candidate HPC $\boldsymbol{\lambda}$ over the current best-observed function value. Formally,

$$\alpha_{\text{PI}}(\boldsymbol{\lambda}) := \mathbb{P}(C(\boldsymbol{\lambda}) < c_{\min}) \quad (2.22)$$

Given a surrogate model with a normal posterior predictive distribution, the PI can be computed in closed form as

$$\alpha_{\text{PI}}(\boldsymbol{\lambda}) = \Phi\left(\frac{c_{\min} - \mu(\boldsymbol{\lambda})}{\sigma(\boldsymbol{\lambda})}\right) \quad (2.23)$$

The LCB is defined as the lower confidence bound based on the mean and standard deviation prediction of the surrogate model. Formally,

$$\alpha_{\text{LCB}}(\boldsymbol{\lambda}) := \mu(\boldsymbol{\lambda}) - \tau\sigma(\boldsymbol{\lambda}), \quad (2.24)$$

where $\tau \in \mathbb{R}^+$ is a control parameter guiding exploration vs. exploitation.

The EI, PI, and LCB are “short-sighted” acquisition functions in the sense that they focus on short-term improvement (i.e., policies based on EI, PI, and LCB are constructed to perform well given that one additional candidate point can be evaluated). This is also called myopic (Garnett, 2023, Chapter 5). Other notable non-myopic acquisition functions are based on ideas from information gain, i.e., entropy search (Hennig and Schuler, 2012), predictive entropy search (Hernández-Lobato et al., 2014), max value entropy search (Wang and Jegelka, 2017) and joint entropy search (Hvarfner et al., 2022a) that try to reduce the uncertainty for the location of the optimum, the value of the optimum, or both. Another popular non-myopic acquisition function is given by knowledge gradient (Frazier et al., 2009) which measures the expected decrease in the minimum posterior mean of the objective function after obtaining a new sample and thereby quantifies the value of information gained from evaluating a given point.

So far, we mainly focused on the surrogate model and acquisition function as building blocks of BO. Two other important building blocks are (1) the mechanism to generate the initial design and (2) the optimization technique used for the acquisition function optimization.

The initial design obtained before making the first model-based proposal should provide the surrogate with rich information about the black-box function that is to be optimized. Methods to generate initial designs exist plenty, e.g., a random design, grid design, or pseudo-random designs such as Sobol’ sequences (Sobol’, 1967) or Latin hypercube designs (McKay et al., 1979). We do not discuss similarities and differences in detail but want to note that on top of the choice of the initial design method, its size can have a bigger impact on BO performance as a small initial design may result in a poor initial surrogate model, wasting subsequent model-based proposals, whereas a large initial design may result in a good initial surrogate model, however, more function

2.1 Hyperparameter Optimization

evaluations were used to construct it. A heuristic can be to set the initial sample size to a constant times the dimensionality of the search space, e.g., between two to ten times (Jones et al., 1993; Snoek et al., 2012; Frazier, 2018).

Regarding acquisition function optimization, the applicability of optimization techniques depends on the search space and the surrogate model. In the case of purely numeric search spaces, global but also local numerical optimization methods are popular. Popular zero-th order methods include CMA-ES (Hansen, 2023) and DIRECT (Jones et al., 1998). If gradients of the acquisition function can be computed (e.g., possible in the case of a GP as a surrogate model), local second-order methods such as L-BFGS-B (Byrd et al., 1995; Zhu et al., 1997) are widely used, typically in combination with a multi-start procedure and a global search procedure for identifying promising starting points (Balandat et al., 2020). In the case of mixed spaces, popular methods include iterated local search variants (Hutter et al., 2007, 2009) or EAs both of which naturally could also be applied to numerical search spaces. Finally, RS with a large budget can in principle always be used as a fallback of last resort although usually not recommended.

So far, we have introduced BO with GPs as a surrogate model as they can be considered the de-facto gold standard for low to moderate dimensional, purely numeric search spaces. HPO, however, can involve the optimization of mixed search spaces and potentially include hierarchical dependencies. For GPs to work in such spaces, this requires the careful design of appropriate kernel functions (Jenatton et al., 2017; Ru et al., 2020; Garrido-Merchán and Hernández-Lobato, 2020). Another downside of GPs is that vanilla implementations scale cubic in the number of data points. Therefore, other types of surrogate models have gained popularity when using BO for HPO, namely Random Forests (RFs), Tree-Structured Parzen Estimators (TPEs), and Bayesian Neural Networks (BNNs). RFs are ensemble models consisting of multiple decision trees trained on different subsets of data. In BO, RFs can provide an estimate of the mean and variance of the objective function under the assumption of each node prediction in a tree following a normal distribution, allowing for uncertainty quantification needed for acquisition functions. Tree-based models are non-parametric, can handle mixed and hierarchical spaces, and are efficient and robust to train (Kim and Choi, 2022). RFs are popularly used as surrogate models in BO by SMAC (Hutter et al., 2011; Lindauer et al., 2022). TPE models the probability of good and bad HPCs separately using kernel density estimation. Instead of optimizing an explicit acquisition function, TPE selects promising hyperparameters based on their likelihood under the good configuration distribution (Bergstra et al., 2011). TPEs are widely recognized as surrogate models following their adoption in the popular Optuna (Akiba et al., 2019) toolbox for HPO. BNNs on the other hand introduce probabilistic weights into deep neural networks, allowing for uncertainty estimation, making them applicable to BO (Springenberg et al., 2016). They are typically trained using Hamiltonian Monte Carlo (HMC) which can be considered the gold standard (Li et al., 2024) but is computationally expensive (Neal, 1996, Chapter 3) or via cheaper variants such as stochastic gradient HMC (Chen et al., 2014). A simpler approach to BNNs is given by deep ensembles that can approximate fully Bayesian inference by averaging model predictions of standard deep neural networks trained with different randomly initialized weights (Lakshminarayanan et al., 2017). BNNs can act as flexible, expressive function approximators capable of handling complex relationships in hyperparameter spaces. With appropriate preprocessing, they can further handle mixed and hierarchical search spaces. Moreover, in contrast to vanilla GPs, they directly allow for modeling non-stationary functions. Note that for BO with GPs with stationary kernels, the possibility to handle non-stationary functions is further given by input warping (Snoek et al., 2014; Cowen-Rivers et al., 2022) or deep kernel learning (Wilson et al., 2016). A comparably cheap and

robust approach to make use of BNNs as surrogate models in BO consists of combining Bayesian linear regression with the features learned by a deep neural network (Snoek et al., 2015).

We have seen that BO is not a single algorithm but describes a class of optimization algorithms that use a surrogate model to propose promising candidates for evaluation. BO has building blocks such as the surrogate model, the acquisition function, and the acquisition function optimizer and is further configurable by the size and concrete choice of initial design or frequency of interleaving configurations that are proposed uniformly at random. In Section 2.1.6 we will briefly introduce the topic of Algorithm Configuration (AC) that can be used to automatically choose well-performing components for a BO algorithm for a class of problems at hand.

We will not introduce theoretical concepts of convergence and bounds on (simple or cumulative) regret in the context of BO in this thesis. Nevertheless, the interested reader is referred to Srinivas et al. (2010) proving a sublinear cumulative regret bound of a global BO algorithm with a GP as a surrogate model and LCB as an acquisition function assuming noisy function evaluations. In the noiseless setting, Bull (2011) proves near optimal convergence rate of a global BO algorithm with a GP as a surrogate model and EI as an acquisition function in the case of a fixed GP prior. De Freitas et al. (2012) proved exponential convergence for global BO with a GP as a surrogate model and LCB as an acquisition function under the assumption of the objective function being locally quadratic in a neighborhood around the global optimum. Garnett (2023, Chapter 10) provides a detailed introduction to the topic of theoretical analysis in BO and outlines ideas and proofs of key literature.

Moreover, we will not discuss topics of warm-starting BO via meta-learning (Feurer et al., 2015b) or transfer-learning for BO (Bai et al., 2023). For a general introduction to the topics of meta-learning and transfer-learning, we refer to Vanschoren (2019).

Having introduced BO and its components, we will continue with an illustration. Assume we want to minimize the Branin function (Branin, 1972), a popular synthetic test function for black-box optimization algorithms. On the domain $[-5, 10] \times [0, 15]$, the Branin function has three global minima at $(-\pi, 12.275)$, $(\pi, 2.275)$, $(9.42578, 2.475)$ with a value of 0.397887. In the following, we will minimize the Branin function on the log scale to allow for prettier visualization of its contour lines. We perform BO with an initial design of eight points drawn uniformly at random, as shown in Figure 2.2a. The posterior mean of a GP with a Matérn $\frac{5}{2}$ kernel trained on the initial design is shown in Figure 2.2b, and the posterior uncertainty is shown in Figure 2.2c. We can see that the GP has reasonably good mean predictions and good uncertainty estimates, with uncertainty being low in areas close to the design points used for training. In Figure 2.2d, we visualize the EI defined on the posterior predictive distribution of the GP. The EI is high in areas where the mean prediction is reasonably low, and the posterior distribution suggests uncertainty. We observe that the EI landscape is multimodal, with two promising areas. In Figure 2.2e, we analogously visualize the PI. The PI is high for candidate points close to design points with a low posterior mean, which increases the probability of improving over the current best function value. Similarly to the EI, the PI landscape is multimodal. Finally, in Figure 2.2f we analogously visualize the LCB ($\tau = 1$). Note that for the LCB, lower values are to be preferred in contrast to the EI and PI which we maximize. Similarly to the EI and PI, the LCB landscape is multimodal, with two promising areas.

2.1 Hyperparameter Optimization

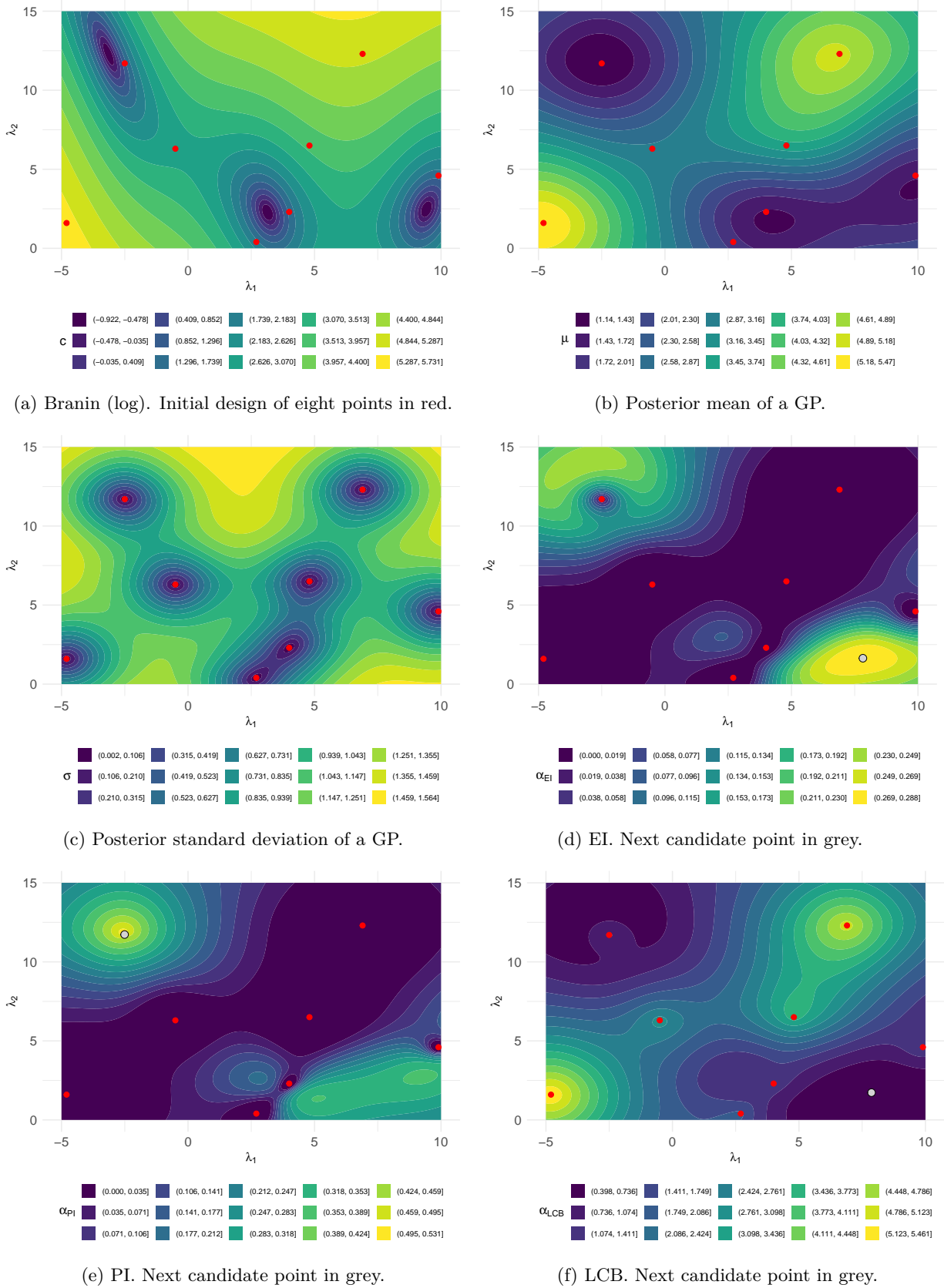


Figure 2.2.: Branin (log) function and principles of BO.

To conclude this introductory section on methods for standard single-objective HPO, we will briefly compare the sampling behavior of different black-box optimization methods suitable for single-objective HPO introduced in this section. Recall that core characteristics are *exploration* vs. *exploitation*. We will again rely on the Branin function (on the log scale) as a synthetic test function, which we want to minimize. We compare GS, RS, vanilla CMA-ES without restarts, and BO with a GP with a Matérn $\frac{5}{2}$ kernel, EI as acquisition function, and an iterated local search as acquisition function optimizer. Each method is allowed 100 function evaluations as a budget, and BO uses the same initial design generated uniformly at random as depicted in Figure 2.2. Each sub-figure in Figure 2.3 visualizes the sampling behavior of each method. For methods that ignore previous function evaluations during their proposals (i.e., RS and GS) we visualize all 100 points in grey. For the sequential methods CMA-ES and BO we visualize the 100 points in a color gradient from grey to blue, where lighter blue colors indicate later proposals during the sequential optimization process. The histograms on the x - and y -axis depict marginal distributions of the points proposed by the methods. For GS, histograms indicate uniform coverage but we observe the downside of GS only evaluating a few unique points per parameter due to the construction of the grid. GS by construction explores the search space reasonably well but falls short in exploitation. For RS, we observe reasonably uniformly distributed marginal distributions, indicating good exploration of the search space. However, RS fails to exploit well-performing regions. CMA-ES as an iterative approach first explores the space before committing to a promising region, exploiting information obtained during optimization, sampling plenty of points around one of the global minima. However, it fails to sample around the other two global minima. This is a nice demonstration, of why CMA-ES is usually run with a restart mechanism (Auger and Hansen, 2005) although this comes at the cost of additional function evaluations. BO demonstrates well-balanced exploration and exploitation, often proposing points in areas around the three global minima but also exploring the search space. In this example, BO is the only method capable of proposing many points in areas around all three global minima using only a few function evaluations.

2.1.2. Neural Architecture Search

While HPO refers to optimizing the hyperparameters of general ML algorithms, NAS specifically focuses on optimizing hyperparameters that define the architecture of deep neural networks. These include the number and types of layers, network depth, and connections. For a general background on NAS, we refer to Elsken et al. (2019a); Wistuba et al. (2019); White et al. (2023).

Although NAS primarily targets neural network architectures, it can be regarded as a special case of HPO. According to Elsken et al. (2019a), NAS consists of three key components: (1) search space, (2) search strategy, and (3) performance estimation strategy. The definition of the search space strongly impacts the feasibility of NAS (Wistuba et al., 2019). NAS differs from standard HPO in two major ways: (1) the vast search space, as allowing arbitrarily large and deep networks with flexible operations and connections leads to an immense number of possibilities, and (2) the computational cost of performance estimation, which is prohibitively expensive for neural architectures, necessitating more efficient estimation techniques than standard resampling methods.

A significant breakthrough in NAS search space design was the introduction of cell-based search spaces (Zoph et al., 2018) in the context of computer vision tasks. In search spaces such as the popular NASNet space, architectures are constructed using two types of cells. Normal cells maintain

2.1 Hyperparameter Optimization

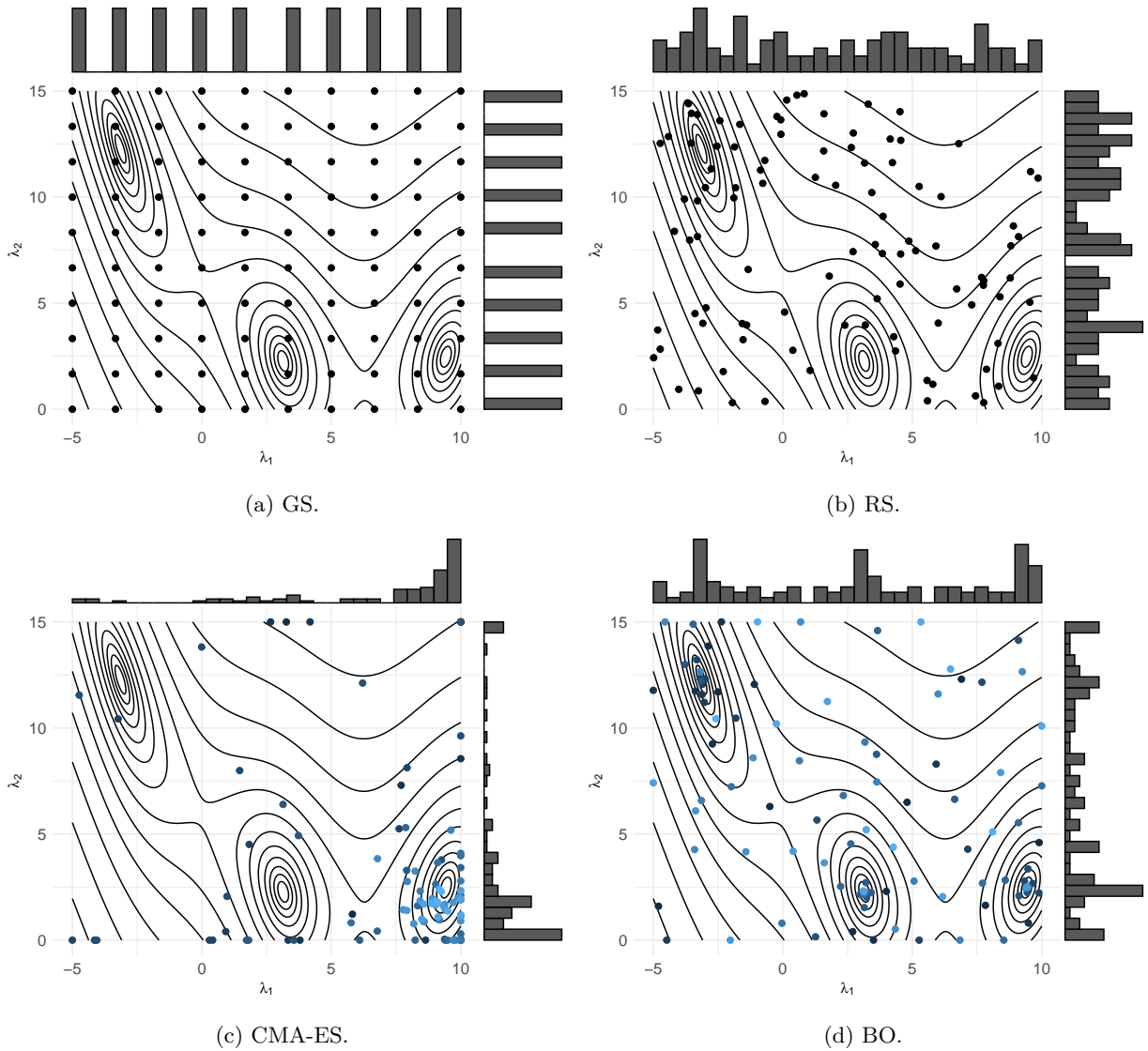


Figure 2.3.: Sampling behavior of different black-box optimizers on Branin (log).

spatial dimensions and reduction cells decrease spatial dimensions. These cells can be sequentially stacked or combined into more complex structures, such as multi-branch architectures.

The key advantage of cell-based search spaces is that NAS can be used to optimize individual cells rather than entire architectures, resulting in a more feasible optimization problem. Moreover, cells optimized on smaller datasets (e.g., CIFAR-10) often generalize well to larger datasets (e.g., ImageNet). Similar to recurrent networks or residual networks, repeating well-designed building blocks improves the efficiency and performance of architectures.

NAS search strategies largely can be grouped into EA approaches, BO approaches, reinforcement learning approaches, differentiable NAS, and One-Shot NAS and Zero-Shot NAS (Elsken et al., 2019a). *EA approaches* (Real et al., 2017; Liu et al., 2018; Real et al., 2019) evolve a population of architectures over time, similarly as introduced in Section 2.1. Here, mutations are usually local operations, e.g., adding or removing a layer, or altering hyperparameters of layers and their

training hyperparameters (Elsken et al., 2019a). *BO approaches* (Bergstra et al., 2013; Domhan et al., 2015; Kandasamy et al., 2018; White et al., 2021a) as usual build a surrogate model to map architectures to performance estimates. However, two peculiarities in the context of NAS are how to encode neural architectures and the choice of a surrogate model. For example, Kandasamy et al. (2018) define a distance metric based on the computation at each layer, the types of operations, and structural connections relying on principles of optimal transport, which allows the usage of a GP as a surrogate model based on this distance metric as a kernel function. Other approaches involve performing a path encoding of architectures (White et al., 2020, 2021a) on which, e.g., an ensemble of neural networks can be trained as a surrogate model (White et al., 2021a,c). *Reinforcement Learning approaches* frame NAS as a reinforcement learning problem in which an agent selects actions (architecture components) and receives rewards based on generalization performance. Key pioneer works include Baker et al. (2017); Zoph and Le (2017). *Gradient-Based or Differentiable NAS approaches* make use of a continuous relaxation of the search space, which enables direct gradient-based optimization of architectures. This idea was popularly introduced by Liu et al. (2019) via the DARTS algorithm. DARTS formulates NAS as a continuous optimization problem, enabling efficient search via bilevel optimization. It constructs a supernet with mixed operations, iteratively updates network weights and architecture parameters, and selects the final architecture by discretizing the learned weights. One-shot NAS and weight sharing similarly rely on a single super-network (the one-shot model) which allows for efficient evaluation of different architectures as its subgraphs. Key pioneer works include ENAS (Pham et al., 2018) and ProxylessNAS (Cai et al., 2019). Finally, *Zero-Shot NAS* approaches try to predict the performance of architectures without any training at all. They rely on proxy metrics based on weight initialization (Tanaka et al., 2020) or graph-based structure (Mellor et al., 2021) that can be indicative of the performance of a trained architecture (Xiang et al., 2023).

Concerning performance evaluation, a notable difference between NAS and vanilla HPO for general ML algorithms is that training an architecture and evaluating it is much more computationally intensive. Usually, evaluating architectures during optimization via full training is infeasible and NAS relies on approximate performance estimation techniques: lower fidelity estimates, learning curve extrapolation, weight inheritance and network morphisms, and one-shot NAS or weight sharing. Among these, we introduce lower fidelity estimates in the context of multi-fidelity optimization in more detail in the next section.

2.1.3. Multi-Fidelity Optimization

In HPO, we optimize an estimate of the generalization performance of a learning algorithm configured by hyperparameters (Equation (2.7)). As obtaining this performance estimate can be computationally expensive (recall that resampling methods involve repeatedly producing a model on training data and evaluating it on test data), the HPO literature of the last years has been actively concerned with speeding up performance evaluation.

Multi-Fidelity Optimization describes a class of algorithms that rely on lower-cost so-called lower fidelity estimates of the objective function that is to be optimized alongside higher-cost high fidelity evaluations to balance the efficiency and accuracy of evaluations. For example, training a neural network for only a few epochs results in speed-up but already can be indicative of the performance of the network trained for many more epochs. Another example of such a fidelity parameter is given by the size of the training set.

2.1 Hyperparameter Optimization

Formally, one can introduce a fidelity parameter $r \in (0, 1]$ (after scaling the range of the fidelity parameter) that influences the resource requirements of evaluating the estimated generalization performance in Equation (2.6). Assuming that a high fidelity parameter (e.g., using all training data or using the maximum number of epochs to be considered) results in better performance, the black-box full-fidelity HPO problem as in Equation (2.7) turns into the following grey-box multi-fidelity HPO problem:

$$\boldsymbol{\lambda}^* \in \arg \min_{(\boldsymbol{\lambda} \in \tilde{\Lambda}, r=1)} c(\boldsymbol{\lambda}, r) = \arg \min_{(\boldsymbol{\lambda} \in \tilde{\Lambda}, r=1)} \widehat{\text{GE}}(\mathcal{I}, \boldsymbol{\lambda}, \boldsymbol{\lambda}_C(r), \mathcal{J}(r), L) \quad (2.25)$$

Here $\mathcal{J}(r)$ denotes that the resampling strategy may be directly influenced by the fidelity parameter r (e.g., via subsampling of the training data) and $\boldsymbol{\lambda}_C(r) \in \tilde{\Lambda}_C$ denotes potential hyperparameters that are influenced by the fidelity parameter and are not optimized over but are given exogenously ($\tilde{\Lambda} \setminus \tilde{\Lambda}_C = \emptyset$). Usually, r only affects either the resampling \mathcal{J} or exogenous hyperparameters $\boldsymbol{\lambda}_C(r)$ and if it affects $\boldsymbol{\lambda}_C$ it often only affects a single hyperparameter.

As the relationship between the fidelity parameter and performance can be complicated and a larger fidelity parameter must not always result in better performance, we can also optimize jointly over the optimal fidelity parameter without restricting it to the maximum:

$$(\boldsymbol{\lambda}^*, r^*) \in \arg \min_{(\boldsymbol{\lambda} \in \tilde{\Lambda}, r \in (0, 1])} c(\boldsymbol{\lambda}, r) = \arg \min_{(\boldsymbol{\lambda} \in \tilde{\Lambda}, r \in (0, 1])} \widehat{\text{GE}}(\mathcal{I}, \boldsymbol{\lambda}, \boldsymbol{\lambda}_C(r), \mathcal{J}(r), L) \quad (2.26)$$

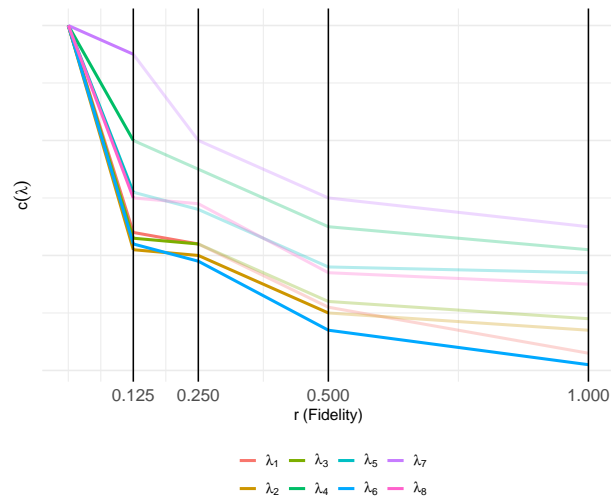
Multi-fidelity algorithms for HPO can be categorized based on how they make use of the fidelity parameter during optimization of Equation (2.25) or Equation (2.26). The following grouping of and introduction of popular multi-fidelity algorithms follows [Feurer and Hutter \(2019\)](#).

Learning curves represent performance trends of a model, such as its performance on increasingly larger dataset subsets or across iterations of an algorithm (e.g., epochs used to train a neural network or boosting iterations of gradient boosting algorithms). Predictive termination uses learning curve models to extrapolate observed curves and stop training if the configuration is unlikely to outperform the best model so far. Instead of simply terminating training of likely poor-performing configurations, training can also be frozen and continued. Key works are given by [Domhan et al. \(2015\)](#); [Klein et al. \(2017b\)](#); [Kadra et al. \(2023\)](#); [Rakotoarison et al. \(2024\)](#).

Bandit based methods frame multi-fidelity HPO as a multi-armed bandit problem (for a general introduction to multi-armed bandits, the reader is referred to [Bubeck and Cesa-Bianchi 2012](#)). Successive halving (SH) is a simple bandit algorithm that can both be deployed in the stochastic ([Karnin et al., 2013](#)) and non-stochastic ([Jamieson and Talwalkar, 2016](#)) setting. The non-stochastic setting suitable for HPO can be characterized by the following: Losses (i.e., generalization performances) are real numbers chosen by an oblivious adversary. Each arm (HPC) has a limit of its loss (i.e., generalization performance) sequence as the fidelity parameter increases. The goal is to identify the arm with the lowest loss limit. The mechanism of how SH works is by evaluating a given number of HPCs for a given (low) fidelity level, observing their performance estimates, and discarding the worst-performing half of HPCs. The top-performing half is promoted to the next stage and evaluated on a higher fidelity level, double the old one and this process repeats until a single HPC is left that has been evaluated on the maximum fidelity. While SH is conceptually simple, it suffers from the “budget vs. the number of configurations” dilemma, since a priori it is unclear whether many configurations should be evaluated on a comparably low

Table 2.1.: Exemplary HB schedule with a halving parameter of $\eta = 2.0$.

Bracket (s)	Stage (i)	Fidelity (r)	#Configurations (n)
3	0	0.125	8
3	1	0.250	4
3	2	0.500	2
3	3	1.000	1
2	0	0.250	6
2	1	0.500	3
2	2	1.000	1
1	0	0.500	4
1	1	1.000	2
0	0	1.000	4

Figure 2.4.: Exemplary visualization of an HB bracket starting with $n = 8$ configurations at lowest fidelity $r = 0.125$ and using a halving parameter $\eta = 2.0$.

fidelity level or if fewer configurations should be evaluated on a comparably higher fidelity level. Hyperband (HB; Li et al. 2018) addresses SH’s trade-off issue by dividing the total optimization budget into multiple settings, where different numbers of configurations are tested at different budgets. HB uses SH as a subroutine in each of its brackets, with different numbers of starting configurations and their starting fidelity level.

In Table 2.1 we illustrate an exemplary HB schedule. Figure 2.4 exemplarily visualizes the third bracket inspired by visualizations of Feurer and Hutter (2019) and Bischl et al. (2023). This most explorative third bracket starts evaluating eight configurations at a fidelity level of 0.125. Each bracket uses a total budget of four full fidelity evaluations.

Neither SH nor HB learn from previous evaluations of configurations but sample configurations uniformly at random. BOHB (Falkner et al., 2018) improves upon HB by replacing the random proposal mechanism with a BO proposal. At its core, BOHB’s surrogate model resembles TPE (Bergstra et al., 2011) but uses multidimensional kernel density estimators. During optimization,

2.1 Hyperparameter Optimization

the model is trained on the highest fidelity subset of the archive for which at least $|\tilde{\Lambda}| + 1$ evaluations have been performed. Therefore, BOHB over time trains its surrogate model on higher fidelity observations but continues using lower fidelity evaluations via the SH subroutine in each bracket. Another possibility to improve over vanilla HB is given by DEHB (Awad et al., 2021) which combines differential evolution with the HB schedule by running multiple subpopulations at different fidelity levels. The best-performing configurations at lower fidelity levels serve as parents for generating new configurations to be evaluated at higher fidelity levels.

Finally, methods that rely on *adaptive fidelity* selection dynamically decide which fidelities to evaluate HPCs on based on prior observations. Key works take inspiration from multi-task BO that uses a multi-task GP to model the performance of related tasks and to learn the correlation of tasks during the optimization process. In the context of multi-fidelity HPO, different tasks come from evaluating different fidelity levels. Similarly, as bandit-based or learning curve approaches, however, this requires the a priori specification of a discrete set of fidelities on which hyperparameters should be evaluated. To make use of the usually smooth dependence on the fidelity, such as, e.g., size of the data subset used (Feurer and Hutter, 2019), approaches as described in Klein et al. (2017a); Kandasamy et al. (2017); Wistuba et al. (2022) treat the fidelity as continuous and learn a surrogate model using custom kernels.

2.1.4. Hyperparameter Optimization with more than one Objective

So far, we have only been concerned with optimizing a single objective in HPO, namely the estimated generalization error. However, in practice, often more than one objective can be of interest and objectives can go beyond performance measures only.

Below, we will briefly introduce three scenarios of HPO with more than one objective: (1) constrained optimization (2) multi-objective optimization, and (3) quality diversity optimization.

Constrained Optimization While good performance is often a primary objective, other objectives of interest can arise, for example in the form of *constraints*. Examples of constraints are given by memory consumption, training time, prediction time and energy usage. Formally, HPO under constraints can be described as solving the following optimization problem:

$$\begin{aligned} \arg \min_{\lambda \in \tilde{\Lambda}} \quad & c(\lambda) & (2.27) \\ \text{subject to} \quad & k_1(\lambda) = 0, \dots, k_n(\lambda) = 0 \quad (\text{equality constraints}), \\ & \tilde{k}_1(\lambda) \geq 0, \dots, \tilde{k}_{\tilde{n}}(\lambda) \geq 0 \quad (\text{inequality constraints}) \end{aligned}$$

Here $k_1(\lambda), \dots, k_n(\lambda)$ denote n functions related to equality constraints and $\tilde{k}_1(\lambda), \dots, \tilde{k}_{\tilde{n}}(\lambda)$ denote \tilde{n} functions related to inequality constraints. Especially in the field of Hardware-Aware NAS (Benmeziane et al., 2021), constrained optimization problems are frequently encountered and formulated to identify architectures that do not exceed a given number of trainable parameters or memory when deployed on an edge device or prediction latency or energy consumption not exceeding a given threshold.

In HPO, constraints are often hidden or unknown. An example of a hidden constraint where we can only make a binary observation (constraint is satisfied or not) is given by memory and time constraints of a computing system during HPO (Feurer and Hutter, 2019). For example, when training a deep neural network fails due to an out-of-memory error, we generally cannot observe the amount of memory requested before the crash (Franceschi et al., 2024) but only notice that training failed. An example of an unknown constraint is for example the prediction time of an SVM, which can only be measured after having trained the model as it depends on the number of support vectors (Feurer and Hutter, 2019). In this sense, the constraint itself is a black-box function that must be evaluated (Gramacy and Lee, 2011) and can be as expensive to evaluate as the primary objective function given by the estimated generalization error.

One way to handle constraints in HPO is to assign a penalty value to configurations not meeting constraints (Golovin et al., 2017). Another approach especially in the context of BO is to model the probability of a configuration violating constraints and encouraging to search for configurations with good performance that are likely to satisfy constraints. This is usually achieved by extending established acquisition functions such as EI, PES, or MES (Gramacy and Lee, 2011; Gardner et al., 2014; Gelbart et al., 2014; Hernández-Lobato et al., 2015; Picheny et al., 2016; Letham et al., 2019; Perrone et al., 2019). Moreover, BO approaches, especially information-theoretic ones, can allow for a decoupled evaluation of the objective function and constraints (Gelbart et al., 2014; Hernández-Lobato et al., 2015, 2016b) which is beneficial if their evaluation can take different amounts of time.

Multi-Objective Optimization Besides constraints, one can also be interested in optimizing *multiple objectives* simultaneously. For example, consider the estimated generalization error as one objective and the size in memory of a neural network when deployed on an edge device as another objective. Instead of treating memory as a constraint, we now treat it as an additional objective. Both objectives of good performance and low memory are generally in conflict, as well-performing architectures usually involve more trainable parameters and therefore require more memory when being deployed. Following Karl et al. (2023) we define the multi-objective HPO problem as

$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} \mathbf{c}(\boldsymbol{\lambda}) = \arg \min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} (c_1(\boldsymbol{\lambda}), \dots, c_m(\boldsymbol{\lambda})), \quad (2.28)$$

where $\mathbf{c} : \tilde{\Lambda} \rightarrow \mathbb{R}^m$, $\boldsymbol{\lambda} \mapsto (c_1(\boldsymbol{\lambda}), \dots, c_m(\boldsymbol{\lambda}))$ maps an HPC to $m \geq 2$ objectives.

In contrast to single-objective optimization, there usually does not exist a single optimal HPC $\boldsymbol{\lambda}^*$. Rather, multi-objective HPO relies on the concept of Pareto optimality and dominance to judge different HPCs concerning multiple objectives. An objective vector \mathbf{c} is said to Pareto dominate another objective vector \mathbf{c}' if the objective values are equally good among all objectives and strictly better in at least one objective. Formally:

$$\mathbf{c} \prec \mathbf{c}' \iff (c_i \leq c'_i \ \forall i \in \{1, \dots, m\}) \wedge (\exists j \in \{1, \dots, m\} : c_j < c'_j) \quad (2.29)$$

Analogously, one defines an HPC $\boldsymbol{\lambda}$ to dominate (\prec) another configuration $\boldsymbol{\lambda}'$ if and only if $\mathbf{c}(\boldsymbol{\lambda}) \prec \mathbf{c}(\boldsymbol{\lambda}')$. “Optimal” HPCs are given by non-dominated configurations: A configuration $\boldsymbol{\lambda}$ is called non-dominated or (Pareto) optimal if and only if there is no other $\boldsymbol{\lambda}'$ that dominates

2.1 Hyperparameter Optimization

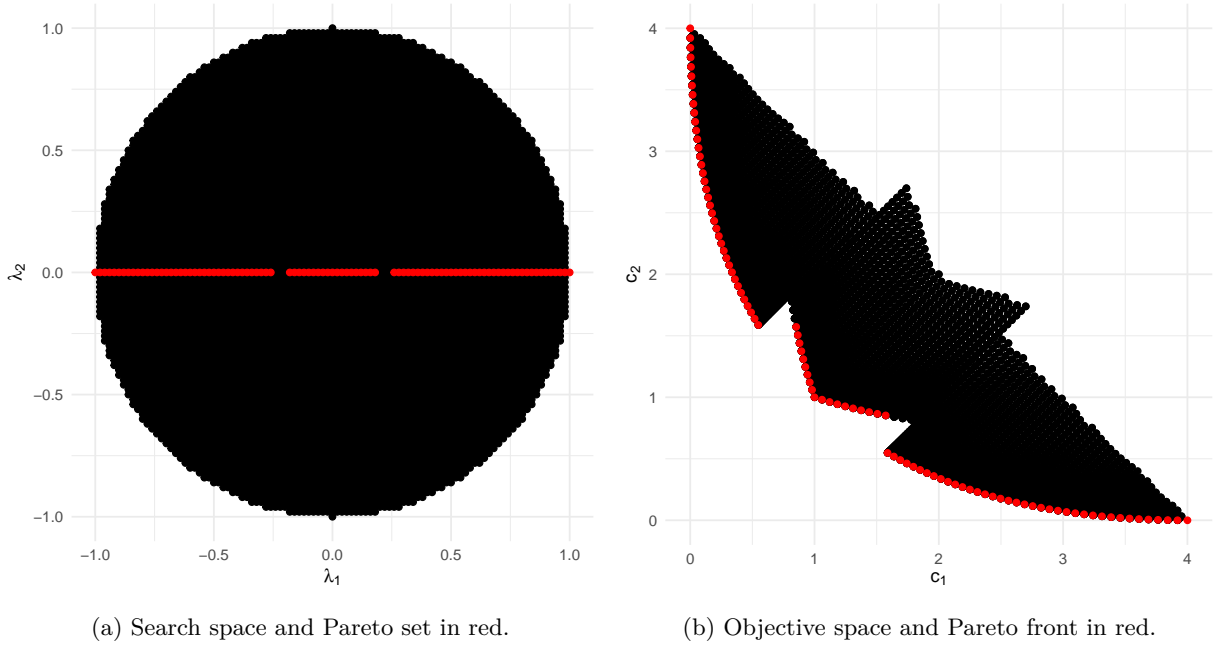


Figure 2.5.: Exemplary Pareto set and front of a two-objective optimization problem.

λ . Pareto dominance defines only a partial order over \mathbb{R}^m . Formally, the set of non-dominated configurations is called the Pareto set:

$$\mathcal{P} := \{\lambda \in \tilde{\Lambda} \mid \nexists \lambda' \in \tilde{\Lambda} \text{ s.t. } \lambda' \prec \lambda\} \quad (2.30)$$

In objective space, the Pareto set can be mapped to the so-called Pareto front $\mathcal{F} := \mathbf{c}(\mathcal{P})$. The general goal of multi-objective optimization is to identify a Pareto set $\hat{\mathcal{P}}$ whose Pareto front $\mathbf{c}(\hat{\mathcal{P}})$ sufficiently well approximates the true Pareto front.

To illustrate this, consider an exemplary Pareto set and front depicted in Figure 2.5 inspired by visualizations of Karl et al. (2023). The search space is given by the unit ball concerning λ_1 and λ_2 . The Pareto set in red (left figure) represents feasible, non-dominated solutions in the search space. The Pareto front in red (right figure) shows how these solutions translate into trade-offs between the two objectives.

The result of multi-objective optimization is not a single optimal solution but the Pareto set of non-dominated solutions. Therefore criteria and so-called quality indicators are needed to quantify characteristics of an approximate Pareto set and its corresponding Pareto front. A popular quality indicator is given by the dominated hypervolume (Zitzler and Thiele, 1998). The hypervolume of an approximation of the Pareto front $\mathbf{c}(\hat{\mathcal{P}})$ is defined as the combined volume of the dominated hypercubes of all solution points concerning a reference point $\mathbf{r} \in \mathbb{R}^m$. Higher hypervolume indicates that an approximation of the Pareto front covers a larger volume in objective space. For more details, we refer to Zitzler and Thiele (1998); Karl et al. (2023). Additionally, Karl et al. (2023) provide an overview of other criteria for comparing solution sets and quality indicators for approximated Pareto fronts.

Following Karl et al. (2023), we provide in Figure 2.6 a categorization of popular objectives in HPO that in combination give rise to multi-objective optimization problems. Often, one of the primary

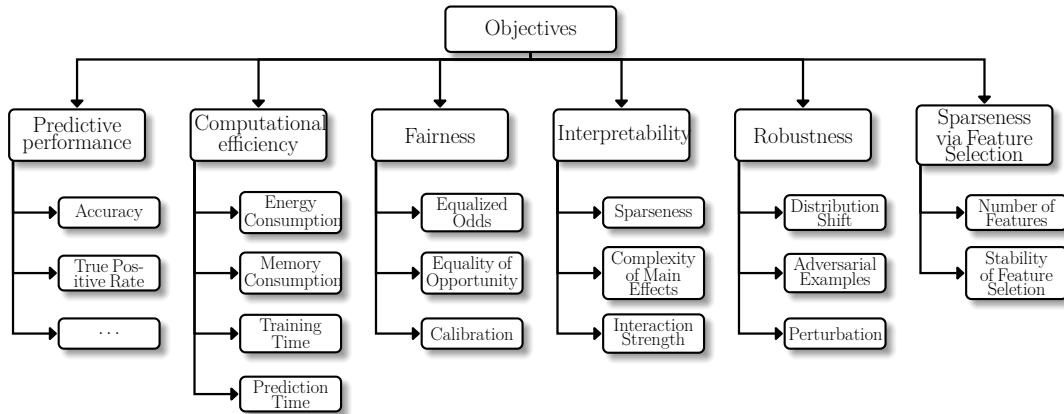


Figure 2.6.: Objectives in HPO.

objectives is given by predictive performance, i.e., maximizing accuracy in classification tasks or maximizing true positive rate. Other objectives are related to computational efficiency and are usually tied to the size of the model or the number of trainable parameters (Tan et al., 2019; Howard et al., 2019). Computational efficiency metrics are core objectives in Hardware-aware NAS (Benmeziane et al., 2021). Fairness (Corbett-Davies et al., 2023), interpretability (Freitas, 2019; Molnar et al., 2020), robustness (Hancox-Li, 2020), and sparseness via feature selection (Li et al., 2022) are related to accountability and transparency in ML and are highly relevant in many practical applications where models are deployed that must fulfill certain legal and ethical standards.

Following Karl et al. (2023) multi-objective optimization methods can be categorized into a priori and a posteriori methods. A posteriori methods return a set of configurations $\hat{\mathcal{P}}$ that try to approximate the true Pareto set \mathcal{P} well, whereas a priori methods return a single configuration that is optimal according to preferences specified before optimization.

Scalarization as an a priori method converts a multi-objective optimization problem into a single-objective problem using a function $s : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$. For example, a weighted sum approach considers a linear combination of the objectives:

$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} \sum_{i=1}^m \alpha_i c_i(\boldsymbol{\lambda}) \quad (2.31)$$

However, scalarization in multi-objective optimization has two key challenges (Karl et al., 2023): (1) Selecting appropriate scalarization weights α_i . (2) A single solution usually cannot fully capture the complexity of conflicting objectives.

A posteriori methods in multi-objective optimization build upon principles that have been well-established in single-objective optimization (see Section 2.1.1). These methods extend optimization strategies to handle multiple conflicting objectives and aim to approximate the Pareto set effectively.

For example, GS or RS in principle can also naturally be used for multi-objective HPO. Since configuration evaluations are independent, adapting these methods simply involves returning the non-dominated solutions. Similar to single-objective HPO, both methods serve as baselines for assessing more advanced optimization techniques.

2.1 Hyperparameter Optimization

Population-based algorithms such as evolutionary approaches are popular for multi-objective optimization methods. As core building blocks of evolutionary approaches are parent selection and recombination of parents via so-called mutation and crossover operators and the selection of members of the population for survival, one can efficiently design algorithms suitable for multi-objective optimization by adapting these mechanisms. Mutation and crossover operators are generally not affected when moving from one to multiple objectives. Parent and survival selection mechanisms however must be adapted as individuals no longer simply can be ranked according to their single-objective value (Karl et al., 2023). According to Emmerich and Deutz (2018), multi-objective EAs largely make use of (1) Pareto dominance for ranking, (2) scalarization involving several single-objective sub-problems with varying parametrizations, and (3) scalar indicators, such as the dominated hypervolume.

Among the most popular algorithms for multi-objective optimization is NSGA-II (Deb et al., 2002). NSGA-II relies on the concept of Pareto dominance combined with a so-called crowding distance mechanism to ensure convergence towards the Pareto front and maintain diversity among solutions. First, an initial population of candidates is generated and evaluated on all objective functions. Individuals are then ranked on different fronts based on Pareto dominance. The best-ranked front corresponds to the non-dominated set and subsequent fronts contain non-dominated individuals given the exclusion of individuals belonging to previous fronts. To preserve diversity, a crowding distance value is assigned to each individual that is given by the sum of differences between an individual’s left and right neighbor in each objective, where large crowding distances are preferred. Individuals who are optimal concerning one objective get assigned an infinite crowding distance and are ranked best. Parents for recombination are selected via binary tournaments, favoring individuals with better ranks according to the non-dominated sorting criterion and higher crowding distances (for tie breaking). The same criteria are used during the survival step of individuals in the population.

BO methods for multi-objective optimization can be categorized (Horn et al., 2015) based on whether they follow a scalarization approach of objectives or whether they rely on surrogate models for each objective. In the latter case, they proceed to use an aggregating acquisition function or formulate the acquisition function optimization problem itself as a multi-objective optimization problem.

A popular BO method based on scalarization is ParEGO (Knowles, 2006). ParEGO relies on the so-called augmented Tchebycheff function to scalarize the objectives:

$$c_{\text{scalarized}}(\boldsymbol{\lambda}) = \max_{i \in \{1, \dots, m\}} (\alpha_i c_i(\boldsymbol{\lambda})) + \rho[\boldsymbol{\alpha} \cdot \mathbf{c}(\boldsymbol{\lambda})] \quad (2.32)$$

Here, ρ is a small positive constant, and $\boldsymbol{\alpha}$ is a weight vector drawn uniformly at random from a set of $\binom{s+m-1}{m-1}$ different weight vectors with s determining the total number of weight vectors:

$$\left\{ \boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_m) \mid \sum_{i=1}^m \alpha_i = 1 \wedge \alpha_i = \frac{l}{s}, l \in \{0, 1, \dots, s\} \right\} \quad (2.33)$$

In each iteration, ParEGO therefore performs a different scalarization of objectives. A surrogate model is trained on the scalarized objective and an acquisition function suitable for single-objective optimization, such as the EI is used to propose the next candidate point for evaluation.

Another BO method based on an aggregating acquisition function relying on a surrogate model for each objective function is given by the Expected Hypervolume Improvement (EHVI; [Emmerich et al. 2006](#)). The EHVI of a candidate point is calculated as the expectation of the hypervolume improvement based on the posterior predictive distributions of the surrogate models. As it involves the evaluation of an expectation, a multidimensional integral must be computed and the related literature contains several works concerned with efficient ways to calculate the EHVI ([Emmerich et al., 2011](#); [Yang et al., 2019](#); [Emmerich et al., 2016](#)). Similarly, as the EHVI extends the EI to multiple objectives, the Probability of Hypervolume Improvement extends the PI to multiple objectives ([Keane, 2006](#)).

For an overview of other multi-objective BO approaches, such as SMS-EGO ([Ponweiser et al., 2008](#)), Multi-EGO ([Jeong and Obayashi, 2005](#)), and information-theoretic acquisition functions such as PESMO ([Hernández-Lobato et al., 2016a](#)), MESMO ([Belakaria et al., 2019](#)), and JES ([Tu et al., 2022](#)) we refer to [Karl et al. \(2023\)](#).

Last but not least, as in single-objective HPO, multi-objective HPO methods can be extended to work in a multi-fidelity setting as introduced in Section 2.1.3. For example, [Schmucker et al. \(2020\)](#) propose HB with random scalarizations as a multi-fidelity multi-objective method and [Salinas et al. \(2021\)](#) in a similar spirit extend HB to work with a non-dominated sorting rule. [Belakaria et al. \(2020\)](#) propose a BO algorithm relying on information-theoretic concepts similarly as MESMO to obtain the next candidate point and fidelity level for evaluation.

Quality Diversity Optimization In contrast to multi-objective optimization where we want to simultaneously optimize more than one competing objective, optimization in the presence of more than one objective can also be framed as optimizing one primary objective but desiring diversity of solutions with respect to other secondary objectives. This may at first feel somewhat similar to constrained optimization. However, whereas constrained optimization returns a single optimal solution that is feasible, quality diversity optimization seeks to return a set of optimal solutions that are well-performing but behaviorally diverse concerning other functions.

Let $c_1 : \tilde{\Lambda} \rightarrow \mathbb{R}$ denote the primary objective that we want to minimize. Behavioral diversity is formalized via so-called niches that partition the search space based on ($i \in \{2, \dots, k\}, k \geq 2$) secondary feature functions $c_i : \tilde{\Lambda} \rightarrow \mathbb{R}$. So-called behavioral niches $N_j \subseteq \tilde{\Lambda}$ ($j \in \{1, \dots, l\}$) are defined on these feature functions via niche-specific boundaries $\mathbf{b}_{ij} = [\text{lower}_{ij}, \text{upper}_{ij}]$. A solution belongs to a behavioral niche N_j if and only if its feature function values fall within the boundaries of each feature function that make up said niche:

$$\begin{aligned} \boldsymbol{\lambda} \in N_j &\iff \forall i \in \{2, \dots, k\} : c_i(\boldsymbol{\lambda}) \in \mathbf{b}_{ij} \\ &\iff (\text{lower}_{2j} \leq c_2(\boldsymbol{\lambda}) < \text{upper}_{2j}) \wedge \dots \wedge (\text{lower}_{kj} \leq c_k(\boldsymbol{\lambda}) < \text{upper}_{kj}) \end{aligned} \quad (2.34)$$

The formal goal of quality diversity optimization is then to find for each behavioral niche N_j the optimal solution:

$$\boldsymbol{\lambda}_j^* \in \arg \min_{\boldsymbol{\lambda} \in N_j} c_1(\boldsymbol{\lambda}) \quad (2.35)$$

The solutions for each niche, also called the elites, are returned as a set $\{\boldsymbol{\lambda}_1^*, \dots, \boldsymbol{\lambda}_l^*\}$.

In Figure 2.7 we visualize an exemplary quality diversity optimization problem. The search space is given by the unit ball with respect to λ_1 and λ_2 . A single feature function c_2 defines three

2.1 Hyperparameter Optimization

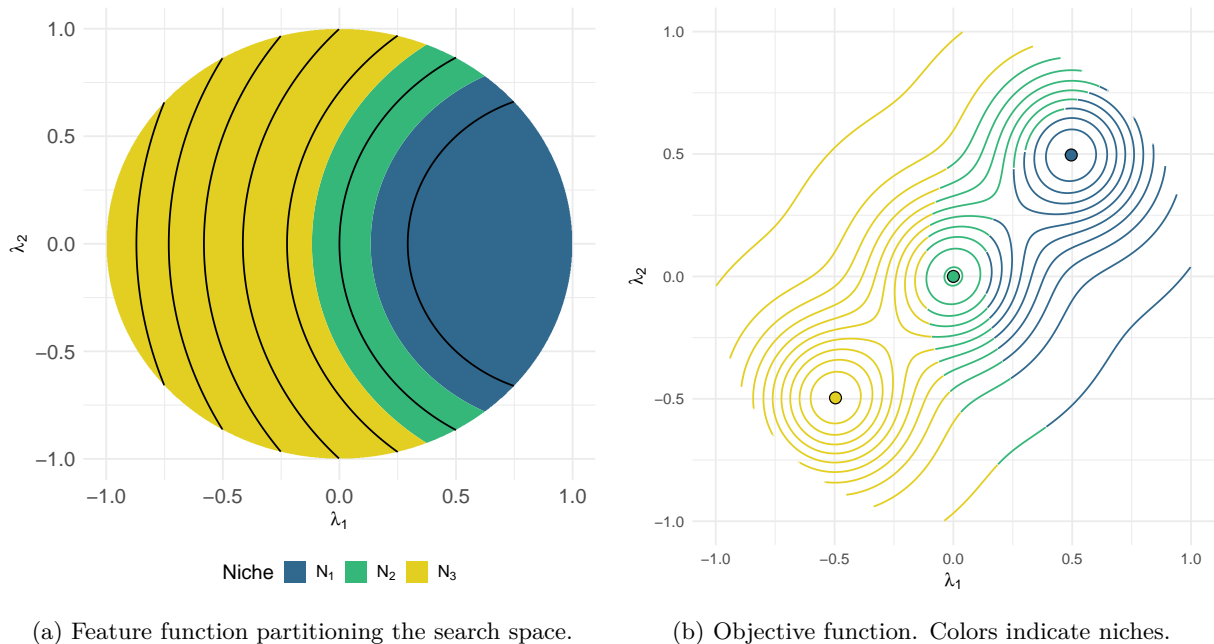


Figure 2.7.: Exemplary quality diversity optimization problem.

niches on the search space, as depicted in Figure 2.7a. The goal is to find for each niche N_1 , N_2 , and N_3 the best point for the objective function c_1 depicted in Figure 2.7b where contour lines are colored according to niches and optimal solutions are indicated via colored points.

In quality diversity optimization, it is often assumed that niches fully partition the search space, $\cup_{j \in \{1, \dots, l\}} N_j = \tilde{\mathbf{A}}$ and that niches are pairwise disjoint, i.e., $\forall j \in \{1, \dots, l\}, j' \in \{1, \dots, l\}, j \neq j' : N_j \cap N_{j'} = \emptyset$. Moreover, in the simplest case of feature functions c_2, \dots, c_k being projections to parameters of the search space, quality diversity optimization directly allows for straightforward diversity of solutions with respect to niches defined on the search space and can be seen as related to multimodal optimization (for an introduction to multimodal optimization in the context of EAs, see Preuss 2015).

Quality diversity optimization is beneficial in applications where a diverse set of solutions is required. Examples include training robotic movements, where a repertoire of behaviors must be learned (Cully et al., 2015), air-foil design (Gaier et al., 2017), developing game-playing agents that employ varied strategies (Perez-Liebana et al., 2021), illuminating latent spaces of generative adversarial networks (Fontaine and Nikolaidis, 2021; Fontaine et al., 2021) and discovering reinforcement learning policies (Batra et al., 2024). The ideas of quality diversity optimization trace back to concepts of Novelty Search. Novelty Search (Lehman and Stanley, 2011a) explores whether diversity alone can yield a good set of solutions. Despite not explicitly optimizing for an objective, Novelty Search has performed surprisingly well in some settings. This led to the development of Novelty Search with Local Competition (Lehman and Stanley, 2011b), the first true quality diversity optimization algorithm. A widely used evolutionary quality diversity optimization algorithm is given by MAP-Elites (Mouret and Clune, 2015).

Quality diversity optimization has gained significant attention in recent years, leading to the development of various novel methods. One such approach, BOP-Elites (Kent and Branke, 2020;

Kent et al., 2023), integrates BO into the quality diversity optimization framework. BOP-Elites employs surrogate models for the objective and feature functions and utilizes an acquisition function over a structured archive. This enables high sample efficiency, even when feature functions are treated as black-boxes.

2.1.5. Parallel Optimization

As HPO is an expensive black-box optimization problem and sequential HPO algorithms are designed to only evaluate a single HPC in each iteration, the related literature has been concerned with designing methods that allow for the evaluation of HPCs in parallel.

Following Bischl et al. (2023) we first want to note that HPO can be parallelized at different levels. When parallelizing iterations, batches of HPCs are proposed and evaluated in parallel. Each HPC evaluation itself further can be parallelized across resampling iterations, while individual train-test splits within, e.g., cross-validation can also be executed independently. Additionally, certain models, such as random forests, allow for parallelization during model fitting itself. In the case of performing nested resampling to obtain an unbiased performance estimate of the model induced by the learning algorithm configured to use the best-performing HPC, parallelization additionally can be performed on the outermost resampling level (Bischl et al., 2023).

When looking at parallelizing iterations, the choice of the HPO algorithm strongly impacts the types of parallelization possible. RS and GS, for example, are considered to be “embarrassingly parallel” (Bischl et al., 2023) as the evaluation of each HPC is independent of the evaluation of all other HPCs. In contrast, iterative methods such as EAs, or BO generally can be adapted to work in a batch parallel setting.

Especially in the context of BO the related work has been concerned with designing *batch-parallel* algorithms. A straightforward approach described in Hutter et al. (2012) proposes to sample the exploration vs. exploitation parameter of the LCB acquisition function from an exponential distribution and optimize each LCB variant to obtain a batch of candidates. Another possibility is given by performing multi-objective optimization of the posterior mean and uncertainty (Bischl et al., 2014) to generate a batch of candidates from the resulting Pareto set. Ginsbourger et al. (2010) describe the constant liar and Kriging believer approach that both fill in function values for previous points proposed in a batch. The constant liar strategy assumes that the unknown function values at the selected batch points are the same (for example the mean of function values) or follow some simple heuristic. The Kriging believer strategy assumes that the surrogate model’s prediction at the new points is correct and uses the predicted mean as the fill-in function value. For myopic acquisition functions, one can further also define so-called q -variants, e.g., the qEI is a truly multivariate acquisition function based on the expected improvement over q points jointly (Ginsbourger et al., 2010; Chevalier and Ginsbourger, 2013). While these acquisition functions can be expensive to optimize for larger batches, a reparameterization trick, and sub-modularity property can allow for a more efficient greedy optimization (Wilson et al., 2017, 2018). Another efficient batch-parallel acquisition function based on the LCB is proposed in Desautels et al. (2014) making use of a confidence bound adjusted for pending observations and hallucinated feedback. Gonzalez et al. (2016) introduce a maximization-penalization approach for batch-parallel BO that modifies the acquisition function to discourage batch points closely together. Their approach estimates a Lipschitz constant of the objective function from the GP and uses it to define exclusion zones around selected batch points.

2.1 Hyperparameter Optimization

Looking at *fully asynchronous parallelization*, existing work in the context of BO is given by [Egelé et al. \(2023\)](#) proposing asynchronous-decentralized BO, wherein each worker runs a sequential BO making use of the LCB acquisition function with differently sampled exploration vs. exploitation trade-off and asynchronously communicates its results through a shared archive. In the context of multi-fidelity optimization, [Li et al. \(2020\)](#) introduce asynchronous SH and [Klein et al. \(2020\)](#) propose a model-based extension based on the concepts of BO. Another fundamentally different approach to fully asynchronous training of neural networks and their hyperparameters is given by population-based training ([Jaderberg et al., 2017](#)). Here, a population of models is evolved asynchronously by interleaving training with evolutionary updates, dynamically adjusting hyperparameters through exploitation (copying successful configurations), and exploration (mutating hyperparameters) to create an adaptive learning schedule.

2.1.6. Configuring an Optimization Algorithm

So far, we have introduced two levels of inference in the context of HPO. On the first level, the inducer maps a dataset to a model \hat{f} , or its parameter vector $\hat{\theta}$ (as introduced in Equation (2.1) which we restate here):

$$\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}, \quad (\mathcal{D}, \lambda) \mapsto \hat{f}$$

We identify the risk-optimal model via ERM (as introduced in Equation (2.3)):

$$\hat{f} = \arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f)$$

The inducer itself is configured by hyperparameters $\lambda \in \tilde{\Lambda} \subseteq \Lambda$. On the second level, an HPO algorithm returns an (estimated) optimal HPC while internally solving the first-level ERM for each candidate HPC. As a result of HPO, we identify the (estimated) optimal HPC (solving the optimization problem already introduced in Equation (2.7)):

$$\hat{\lambda} = \arg \min_{\lambda \in \tilde{\Lambda}} c(\lambda)$$

HPO is a bilevel inference mechanism ([Bischl et al., 2023](#)) consisting of first-level and second-level inference ([Guyon et al., 2010](#)), sometimes also referred to as inner and outer inference ([Franceschi et al., 2018](#)).

Acknowledging that an HPO algorithm itself can be configured by control parameters gives rise to a third level. We now focus on identifying the best configuration of the HPO algorithm \mathcal{O} configured by control parameters $\gamma \in \Gamma$ (these could, for example, be the choice of surrogate model or acquisition function in the context of BO) for a class of HPO problem instances at hand. Formally, the HPO algorithm \mathcal{O} maps a dataset to the optimal HPC:

$$\mathcal{O} : \mathbb{D} \times \Gamma \rightarrow \tilde{\Lambda}, \quad (\mathcal{D}, \gamma) \mapsto \hat{\lambda}$$

We can now use a meta-optimization algorithm on a third level to map a collection of HPO problem instances to the optimal HPO algorithm configuration, while solving the second-level optimization problem for each HPO problem conditional on an HPO algorithm configuration:

$$\hat{\gamma} = \arg \min_{\gamma \in \Gamma} \kappa(\gamma)$$

Here, κ is a function that takes in a configuration γ of the HPO algorithm \mathcal{O} (and a collection of HPO problem instances which we suppress here for clarity) and outputs the value of a meta-objective over the collection of HPO problem instances. An example of such a meta-objective is given by the average final generalization error of the best-performing HPC identified for each HPO problem instance when running the HPO algorithm for a given optimization budget. The difference between this Algorithm Configuration (AC) problem on the third level and HPO on the second level is that in HPO we search for the optimal HPC for a given task. In AC, we optimize over multiple problem instances and search for optimal control parameters that perform well in expectation over these multiple instances.

We will not discuss the general field of AC here but instead, refer to [Hoos \(2012b\)](#); [Eggenberger et al. \(2019\)](#); [Schede et al. \(2022\)](#) for additional background and a more standard introduction of AC. An example of AC for BO in HPO is given by [Lindauer et al. \(2019\)](#) who investigate the impact of optimizing BO’s own hyperparameters such as the choice of surrogate model and its hyperparameters and transformations of the objective function prior to using a surrogate model. AC is closely linked to the concept of programming by optimization ([Hoos, 2012a](#)), an approach that advocates delaying design choices in software development while using ML and AC to automatically optimize software and configurations of algorithms.

2.2. Benchmarking

ML is a field that can be approached both as a formal science and an empirical science. Theoretical research on the one hand focuses on the mathematical foundations of learning algorithms, exploring their limitations and proving properties under assumptions. Empirical research on the other hand emphasizes practical relevance and largely relies on *benchmarking* algorithms on real-world problems and datasets to assess their effectiveness. To bridge both perspectives, researchers often derive hypotheses about algorithm behavior and validate them through benchmarking on problem instances, or they simply try to demonstrate that their algorithms improve over a current state-of-the-art algorithm.

Modern ML research therefore relies heavily on benchmarking, where standardized datasets and evaluation metrics and protocols serve as shared resources in research communities. In general computing, a benchmark is defined as “a problem that has been designed to evaluate the performance of a system [which] is subjected to a known workload and the performance of the system against this workload is measured. Typically, the purpose is to compare the measured performance with that of other systems that have been subject to the same benchmark test.” ([Butterfield et al., 2016](#)). Following [Schlangen \(2021\)](#) and [Koch et al. \(2021\)](#), we refer to ML benchmarks as community resources used to evaluate ML models or algorithms. Benchmarks typically consist of a particular task through a dataset (or multiple tasks and datasets) and a quantitative metric used for evaluation, as well as an evaluation protocol specifying, e.g., compute budget and how to compare algorithms. [Koch et al. \(2021\)](#) point out that, historically, the practice of benchmarking was partially advocated to the ML community after the “AI Winter” of the 1980s to assess the outcome and value of grants stressing the importance of practical applicability of algorithms going beyond theoretical insight and properties, see also [Church \(2018\)](#); [Koch and Peterson \(2024\)](#); [Orr and Kang \(2024\)](#).

2.2 Benchmarking

In recent years, ML communities have strongly evolved to facilitate benchmarking (Koch et al., 2021). This includes the development and collection of open-access datasets, for example MNIST (LeCun et al., 1998), CIFAR-10 and CIFAR-100 (Krizhevsky and Hinton, 2009), ImageNet (Deng et al., 2009), GLUE (Wang et al., 2018), OpenML CC18 (Bischl et al., 2021), competitions and challenges (e.g., the Netflix competition, see Bennett and Lanning 2007, the ChaLearn AutoML challenge, see Guyon et al. 2015, and Kaggle challenges in general) and benchmarking software and platforms (Vanschoren et al., 2014; Eggenberger et al., 2021; Gijssbers et al., 2024), allowing for benchmarking with comparably little effort.

Benchmarking is important because it allows for tracking scientific progress over time through empirical validation. Agreed-upon datasets and tasks, evaluation metrics, and protocols facilitate fair and standardized comparisons of algorithms and can drive innovation, i.e., by illustrating why certain algorithms excel or fail and improving upon them. Best practices include ablation studies of components of algorithms, i.e., changing one factor or component of an algorithm at a time, rerunning benchmarks, and performing statistical analysis of performance differences attributed to factors in an isolated fashion (Saltelli, 2002; Hoos, 2012a; Hutter et al., 2014; Fawcett and Hoos, 2016).

Having said that, current benchmarking practices in ML research cannot be considered without flaws. For a general critical discussion of benchmarks in ML over the last years and its often narrow focus on few datasets and tasks that are reused in different sub-communities and resulting risks such as the illusion of progress towards performance improvements that generalize see Koch et al. (2021); Raji et al. (2021), but also Recht et al. (2018, 2019). Moreover, Herrmann et al. (2024) identify issues in past and current empirical ML research, i.e., most work lacking empirical rigor and reproducibility, SOTA-hacking and biased benchmarks, risking distorted scientific progress, much research being framed as confirmatory albeit operating in an exploratory setting and call out for more neutral comparison studies, replications studies and meta-studies. For best practices for benchmarking black-box optimization in general but also specifically HPO and NAS, we refer to Bartz-Beielstein et al. (2020), Lindauer and Hutter (2020), and Bischl et al. (2023).

2.2.1. Categorizing Hyperparameter Optimization Benchmarks

The fields of AutoML and more specifically HPO have always been benchmarking and engineering heavy, focusing on the construction and design of algorithms that show good performance on tasks deemed to be practically relevant (Bergstra et al., 2011, 2013). This is sensible, as a research field that is concerned with democratizing ML and making it accessible to anybody without expert knowledge should, without doubt, be able to demonstrate good performance on real-world problems.

Focusing on HPO, the first standardized HPO benchmarks were introduced through HPOLib (Eggenberger et al., 2013). HPOLib consists of a mix of synthetic black-box functions (e.g., Branin and Hartmann), low-dimensional HPO problems (e.g. logistic regression), medium-dimensional HPO problems (e.g., neural networks), and high-dimensional HPO problems (e.g., optimizing the AutoWEKA framework introduced in Thornton et al. 2013 which combines algorithm selection and HPO in a so-called CASH problem).

Benchmarks used in HPO research can be categorized into four groups: (1) Real-world HPO problems where actual HPO is performed by training and evaluating a learning algorithm via

	Real-World HPO	Synthetic	Tabular	Surrogate
Objective Function	Performing actual HPO	Mathematically defined functions (e.g., Branin, Ackley, BBOB)	Computed performance values of real HPO stored in a lookup table	Regression model of real HPO data mapping configurations to performance
Computational Cost	High	Low	Low (lookup)	Low (prediction)
Real-World Relation	Gold standard	Low	High	Moderate to high (depends on surrogate)
Noise	Includes all sources of real-world HPO noise	Usually deterministic, unless noise is artificially added	Reflects noise in original data	Reflects noise in original data
Search Space	Original HPO problem	Often low dimensional purely numeric but can be extended	Matches original HPO problem, but discretized search space	Matches original HPO problem
Reproducibility	Depends on setup, may vary due to stochasticity in training and hardware	Highly reproducible (deterministic mathematical functions)	Highly reproducible (fixed lookup table)	Highly reproducible
Usability	Low (computationally expensive, requires real HPO)	High (fast and easy to use)	High (simple table lookup, no actual HPO)	High (fast queries, no actual HPO, but quality depends on surrogate model)
Use Cases	Realistic HPO evaluation, understanding true ML overheads	Theoretical development, debugging, optimizer testing	Rapid benchmarking, standardized comparisons	Approximate large-scale HPO experiments at reduced cost
Drawbacks	High cost, resource-intensive, hard to isolate individual factors	May not generalize to real HPO landscapes	Limited to stored configurations, no extrapolation, discretized search space	Strong dependence on surrogate model quality

Table 2.2.: Comparison of real-world, synthetic, tabular, and surrogate HPO benchmarks.

a resampling method. (2) Synthetic mathematical functions, e.g. Branin, Ackley, or black-box benchmarking suites such as BBOB functions (Hansen et al., 2009). (3) Tabular HPO problems (e.g., Ying et al. 2019) where real-world HPO problems were evaluated with many HPCs, and the results have been stored. Therefore, during benchmarking, instead of evaluating a configuration via resampling, one can simply perform a lookup of the pre-collected data. (4) Surrogate HPO Problems (e.g., Eggenesperger et al. 2015) where a surrogate model, e.g., a regression model has been trained on pre-collected tabular HPO data so that during benchmarking, instead of evaluating a configuration via resampling, one can simply perform a prediction with the surrogate model. We provide an overview of the similarities and differences of these types of benchmarks with respect to different dimensions such as the objective function, computational cost, reproducibility, and usability in Table 2.2.

2.2.2. Landscape Analysis

In black-box optimization, no explicit knowledge about the problem structure is available beyond input-output evaluations. To address this, past works have developed techniques to extract numerical features that characterize optimization landscapes. Exploratory landscape analysis (ELA)

is a framework for characterizing the properties of an optimization problem’s landscape solely relying on computationally cheap, low-level numeric features to distinguish different optimization problems (Mersmann et al., 2011) instead of relying on manually crafted high-level features. The core implication of the idea of ELA is that if a problem’s characteristics are well understood, one can match it with an appropriate optimization algorithm. ELA relies on a comparably small sample of function evaluations, such as random sampling or a Latin hypercube design. It then applies statistical measures and ML techniques to compute low-level numerical features that describe the optimization landscape. In the original work of Mersmann et al. (2011), these features can be grouped into meta-model features (e.g., the adjusted R^2 of an estimated linear regression model with and without interactions), convexity features (e.g., the mean deviation from linearity), distribution features of the function values (e.g., skewness and kurtosis), level set features (e.g., misclassification errors of a linear or quadratic discriminant analysis for function values split by quantiles), local search features (e.g., the number of estimated local optima), and curvature features (e.g., minimum and maximum and quantile values of the euclidean norm of an estimated numerical gradient). These low-level features relate to high-level features such as global structure, multimodality, separability, global to local optima contrast, search space homogeneity, plateaus, variable scaling and basin² size homogeneity. High-level features abstractly describe problem characteristics, whereas ELA low-level features are numeric indicators computed directly from function samples. Computing these ELA features for the BBOB functions allowed Mersmann et al. (2011) to separate the BBOB problem groups solely based on ELA features with only a few low-cost features being required. This paved the way for automated algorithm selection based on ELA features (Bischl et al., 2012a; Kerschke and Trautmann, 2019). For example Bischl et al. (2012a) frame algorithm selection on the BBOB problems as a cost-sensitive classification problem relying on ELA features, where misclassification costs depend on performance differences. Over the last years, ELA has grown to its own research field within the black-box optimization community with recent papers proposing additional features based on concepts such as cell mapping (Kerschke et al., 2014), information content (Muñoz et al., 2015) or nearest better clustering (Kerschke et al., 2015). For example, the idea of cell mapping features is based on the observation that a continuous search space can be partitioned in every dimension thus achieving a discretization of the original search space into cells. The discretization of the original sample into cells then allows the computation of additional features. These features help to characterize the global structure and multimodality of an optimization problem.

²A basin or basin of attraction (Asenjo et al., 2013) is a region in the search space where an optimization algorithm, when initialized within that region, will converge to a certain local or global optimum.

Part II.

Contributions

3. Contributions to the Foundations of Hyperparameter Optimization

In this chapter, we present contributions to the foundations of HPO. By contributions to the *foundations* of HPO, we refer to research that deepens our general understanding of the core components of HPO, focusing on theoretical insights and empirical findings that try to explain how the HPO problem behaves as an optimization task. Despite the importance of HPO in ML, many aspects of its optimization landscape and the role of resampling strategies remain poorly understood and underexplored.

A central component of HPO is the estimation of the generalization error through resampling methods such as holdout or cross-validation. While recent works have been concerned with statistical properties of resampling methods, especially cross-validation (Nadeau and Bengio, 1999; Bengio and Grandvalet, 2003; Borra and Di Ciaccio, 2010; Austern and Zhou, 2020; Bayle et al., 2020; Bates et al., 2024), little work has explicitly addressed how resampling interacts with the optimization process itself. Generally, the bias and variance trade-offs of different resamplings (Bischi et al., 2012b) should be considered, however, in the context of HPO we do usually care more about reducing variance when trying to identify a well-performing HPC (Cawley and Talbot, 2010) unless we are also interested in obtaining an unbiased performance estimate of the learning algorithm configured by the best HPC. In this case, we must resort to nested resampling (Cawley and Talbot, 2010; Bischi et al., 2023). A common assumption in HPO is that all HPCs should be evaluated on the same resampling, i.e., the holdout split is performed once before optimization and each HPC is evaluated via the same train and validation split. While this makes sense from a statistical perspective of paired comparisons, one can ask the question of how choosing different resampling splits for each HPC would affect HPO and the generalization error of the model of the best-performing HPC. To our best knowledge, this has only been briefly touched upon in a thesis by Lévesque (2018) demonstrating that for holdout and cross-validation, this can improve generalization error when using BO or CMA-ES as HPO algorithms.

Beyond resampling, a deeper understanding of HPO landscapes is critical for designing better optimization algorithms. While the black-box optimization community has developed ELA techniques to analyze synthetic optimization functions, HPO has largely been treated as a generic black-box problem without a deeper examination of its landscape properties. In the related field of AC, Pushak and Hoos (2018) demonstrated that many numerical parameter landscapes are often uni-modal and even convex, suggesting that simpler optimization methods might be more effective than previously thought. In NAS, related work is given by White et al. (2021b) who investigate the role of noise (for a general discussion of the variance and sources of variation in ML benchmark experiments, see also Bouthillier et al. 2021) and show that reducing noise in the training pipeline can make NAS easier by reducing the number of local minima, resulting in simple local search algorithms to outperform more complex NAS algorithms. As understanding the landscape of HPO problems is crucial for efficient algorithm development, gaining a better

understanding of its landscape is highly relevant. Moreover, as black-box optimization algorithms are often benchmarked on synthetic functions but later on still applied to HPO it is important to compare the landscape properties of HPO to the ones of existing black-box functions.

In the first contribution article *Reshuffling resampling splits can improve generalization of hyperparameter optimization* we systematically examine the effect of reshuffling (e.g., choosing a new train and validation split for each HPC) on HPO generalization error. We show theoretically that reshuffling resampling splits during HPO can result in a final configuration with better overall expected generalization error and relate these observations to the properties of the loss landscape, namely curvature and noisiness of the estimated generalization error. In both controlled simulation studies and a large-scale realistic HPO benchmark, we demonstrate that reshuffling resampling splits indeed can lead to real-world improvement of HPO. Especially holdout can strongly benefit from reshuffling and this observation holds independently of the choice of HPO algorithm, notably also for a simple RS.

In the second contributing article *HPO \times ELA: Investigating hyperparameter optimization landscapes by means of exploratory landscape analysis* we systematically investigate similarities and differences of HPO and synthetic black-box optimization problems given by the BBOB functions. We compare the performance of different black-box optimizers, compute ELA features for both HPO and BBOB problems, and examine how HPO problems position themselves in ELA feature space. Using a few features such as kurtosis we can tell HPO problems apart from BBOB problems. In a cluster analysis we demonstrate that HPO problems mostly position themselves with BBOB problems of little multimodality.

3.1. Reshuffling Resampling Splits can Improve Generalization of Hyperparameter Optimization

Contributing article:

T. Nagler*, L. Schneider*, B. Bischl, and M. Feurer. Reshuffling resampling splits can improve generalization of hyperparameter optimization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 40486–40533, 2024. https://proceedings.neurips.cc/paper_files/paper/2024/hash/47811ee68103bfcde7ca2223fccefb3a-Abstract-Conference.html.

Copyright information:

This article is licensed under copyright and license as per Curran Associates regulations.

Author contributions:

Thomas Nagler and Lennart Schneider share the first authorship. Their overall contributions can be described as follows. The initial project idea was originally developed by Lennart Schneider and Matthias Feurer, who identified the potential benefits of reshuffling during HPO in benchmark experiments. The conceptualization of the project was further refined through contributions from Bernd Bischl and Thomas Nagler. Thomas Nagler developed the theoretical framework, performed the formal analysis, and authored all proofs presented in the manuscript. Lennart Schneider and Thomas Nagler collaboratively designed the simulation study, which was executed and analyzed by Lennart Schneider. The benchmark design was a joint effort by Lennart Schneider, Bernd Bischl, and Matthias Feurer, with Lennart Schneider implementing the code for the benchmarks, conducting all experiments, and analyzing the results. Lennart Schneider drafted the initial versions of the Introduction, Simulation Study, Benchmark Experiments, and Discussion sections. He also drafted the Extended Related Work, Details Regarding Benchmark Experiments, and Additional Benchmark Results Visualizations sections. Thomas Nagler wrote the Theoretical Analysis and Proofs of the Main Results sections, as well as the Additional Results on the Density of Random HPO Grids and Selected Validation Schemes sections. Matthias Feurer conducted the literature search and review, integrating the findings into the Introduction and the Extended Related Work section. Matthias Feurer, Bernd Bischl, and Thomas Nagler further iterated on the Introduction, Discussion, and all appendix sections, providing feedback and refinements. All authors contributed to the revision process of the manuscript.

Supplementary material available at:

- Code: https://github.com/slds-lmu/paper_2024_resuffling
- Data: <https://doi.org/10.6084/m9.figshare.27627504.v1>
- arXiv version: <https://arxiv.org/abs/2405.15393>

Reshuffling Resampling Splits Can Improve Generalization of Hyperparameter Optimization

Thomas Nagler*
t.nagler@lmu.de

Lennart Schneider*

Bernd Bischl

Matthias Feurer

Department of Statistics, LMU Munich
Munich Center for Machine Learning (MCML)

Abstract

Hyperparameter optimization is crucial for obtaining peak performance of machine learning models. The standard protocol evaluates various hyperparameter configurations using a resampling estimate of the generalization error to guide optimization and select a final hyperparameter configuration. Without much evidence, paired resampling splits, i.e., either a fixed train-validation split or a fixed cross-validation scheme, are often recommended. We show that, surprisingly, reshuffling the splits for every configuration often improves the final model’s generalization performance on unseen data. Our theoretical analysis explains how reshuffling affects the asymptotic behavior of the validation loss surface and provides a bound on the expected regret in the limiting regime. This bound connects the potential benefits of reshuffling to the signal and noise characteristics of the underlying optimization problem. We confirm our theoretical results in a controlled simulation study and demonstrate the practical usefulness of reshuffling in a large-scale, realistic hyperparameter optimization experiment. While reshuffling leads to test performances that are competitive with using fixed splits, it drastically improves results for a single train-validation holdout protocol and can often make holdout become competitive with standard CV while being computationally cheaper.

1 Introduction

Hyperparameters have been shown to strongly influence the performance of machine learning models (van Rijn & Hutter, 2018; Probst et al., 2019). The primary goal of hyperparameter optimization (HPO; also called tuning) is the identification and selection of a hyperparameter configuration (HPC) that minimizes the estimated generalization error (Feurer & Hutter, 2019; Bischl et al., 2023). Typically, this task is challenged by the absence of a closed-form mathematical description of the objective function, the unavailability of an analytic gradient, and the large cost to evaluate HPCs, categorizing HPO as a noisy, black-box optimization problem. An HPC is evaluated via resampling, such as a holdout split or M -fold cross-validation (CV), during tuning.

These resampling splits are usually constructed in a fixed and instantiated manner, i.e., the same training and validation splits are used for the internal evaluation of all configurations. On the one hand, this is an intuitive approach, as it should facilitate a fair comparison between HPCs and reduce the variance in the comparison.¹ On the other hand, such a fixing of train and validation splits might steer the optimization, especially after a substantial budget of evaluations, towards favoring HPCs

^{*}Equal contribution.

¹This approach likely originates from the concept of paired statistical tests and the resulting variance reduction, but in our literature search we did not find any references discussing this in the context of HPO. For example, when comparing the performance of two classifiers on one dataset, paired tests are commonly

which are specifically tailored to the chosen splits. Such and related effects, where we "overoptimize" the validation performance without effective reward in improved generalization performance have been sometimes dubbed "overtuning" or "oversearching". For a more detailed discussion of this topic, including related work, see Section 5 and Appendix B. The practice of reshuffling resampling splits during HPO is generally neither discussed in the scientific literature nor HPO software tools.² To the best of our knowledge, only Lévesque (2018) investigated reshuffling train-validation splits for every new HPC. For both holdout and M -fold CV using reshuffled resampling splits resulted in, on average, slightly lower generalization error when used in combination with Bayesian optimization (BO, Garnett, 2023) or CMA-ES (Hansen & Ostermeier, 2001) as HPO algorithms. Additionally, reshuffling was used by a solution to the NeurIPS 2006 performance prediction challenge to estimate the final generalization performance (Guyon et al., 2006). Recently, in the context of evolutionary optimization, reshuffling was applied after every generation (Larcher & Barbosa, 2022).

In this paper, we systematically examine the effect of reshuffling on HPO performance. Our contributions can be summarized as follows:

1. We show theoretically that reshuffling resampling splits during HPO can result in finding a configuration with better overall generalization performance, especially when the loss surface is rather flat and its estimate is noisy (Section 2).
2. We confirm these theoretical insights through controlled simulation studies (Section 3).
3. We demonstrate in realistic HPO benchmark experiments that reshuffling splits can lead to a real-world improvement of HPO (Section 4). Especially in the case of reshuffled holdout, we find that the final generalization performance is often on par with 5-fold CV under a wide range of settings.

We discuss results, limitations, and avenues for future research in Section 5.

2 Theoretical Analysis

2.1 Problem Statement and Setup

Machine learning (ML) aims to fit a model to data, so that it generalizes well to new observations of the same distribution. Let $\mathcal{D} = \{\mathbf{Z}_i\}_{i=1}^n$ be the observed dataset consisting of *i.i.d.* random variables from a distribution P , i.e., in the supervised setting $\mathbf{Z}_i = (\mathbf{X}_i, Y_i)$.^{3,4} Formally, an inducer g configured by an HPC $\lambda \in \Lambda$ maps a dataset \mathcal{D} to a model from our hypothesis space $\mathcal{H} = g_\lambda(\mathcal{D}) \in \mathcal{H}$. During HPO, we want to find a HPC that minimizes the expected generalization error, i.e., find

$$\lambda^* = \arg \min_{\lambda \in \Lambda} \mu(\lambda), \quad \text{where} \quad \mu(\lambda) = \mathbb{E}[\ell(\mathbf{Z}, g_\lambda(\mathcal{D}))],$$

where $\ell(\mathbf{Z}, h)$ is the loss of model h on a fresh observation \mathbf{Z} . In practice, there is usually a limited computational budget for each HPO run, so we assume that there is only a finite number of distinct HPCs $\Lambda = \{\lambda_1, \dots, \lambda_j\}$ to be evaluated, which also simplifies the subsequent analysis. Naturally, we cannot optimize the generalization error directly, but only an estimate of it. To do so, a resampling is constructed. For every HPC λ_j , draw M random sets $\mathcal{I}_{1,j}, \dots, \mathcal{I}_{M,j} \subset \{1, \dots, n\}$ of validation indices with $n_{\text{valid}} = \lceil \alpha n \rceil$ instances each. The random index draws are assumed to be independent of the observed data. The data is then split accordingly into pairs $\mathcal{V}_{m,j} = \{\mathbf{Z}_i\}_{i \in \mathcal{I}_{m,j}}, \mathcal{T}_{m,j} = \{\mathbf{Z}_i\}_{i \notin \mathcal{I}_{m,j}}$ of disjoint validation and training sets. Define the validation loss on the m -th fold

$$L(\mathcal{V}_{m,j}, g_{\lambda_j}(\mathcal{T}_{m,j})) = \frac{1}{n_{\text{valid}}} \sum_{i \in \mathcal{I}_{m,j}} \ell(\mathbf{Z}_i, g_{\lambda_j}(\mathcal{T}_{m,j})),$$

¹employed that implicitly assume that differences between the performance of classifiers on a given CV fold are comparable (Dietterich, 1998; Nadeau & Bengio, 1999, 2003; Demšar, 2006).

²In Appendix B, we present an overview of how resampling is addressed in tutorials and examples of standard HPO libraries and software. We conclude that usually fixed splits are used or recommended.

³Throughout, we use bold letters to indicate (fixed and random) vectors.

⁴We provide a notation table for symbols used in the main paper in Table 2 in the appendix.

and the M -fold validation loss as

$$\widehat{\mu}(\boldsymbol{\lambda}_j) = \frac{1}{M} \sum_{m=1}^M L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j})).$$

Since μ is unknown, we minimize $\widehat{\boldsymbol{\lambda}} = \arg \min_{\boldsymbol{\lambda} \in \Lambda} \widehat{\mu}(\boldsymbol{\lambda})$, hoping that $\mu(\widehat{\boldsymbol{\lambda}})$ will also be small. Typically, the same splits are used for every HPC, so $\mathcal{I}_{m,j} = \mathcal{I}_m$ for all $j = 1, \dots, J$ and $m = 1, \dots, M$. In the following, we investigate how reshuffling train-validation splits (i.e., $\mathcal{I}_{m,j} \neq \mathcal{I}_{m,j'}$ for $j \neq j'$) affects the HPO problem.

2.2 How Reshuffling Affects the Loss Surface

We first investigate how different validation and reshuffling strategies affect the empirical loss surface $\widehat{\mu}$. In particular, we derive the limiting distribution of the sequence $\sqrt{n}(\widehat{\mu}(\boldsymbol{\lambda}_j) - \mu(\boldsymbol{\lambda}_j))_{j=1}^J$. This limiting regime will not only reveal the effect of reshuffling on the loss surface, but also give us a tractable setting to study HPO performance.

Theorem 2.1. *Under regularity conditions stated in Appendix C.1, it holds*

$$\sqrt{n}(\widehat{\mu}(\boldsymbol{\lambda}_j) - \mu(\boldsymbol{\lambda}_j))_{j=1}^J \rightarrow \mathcal{N}(0, \Sigma) \quad \text{in distribution,}$$

where

$$\Sigma_{i,j} = \tau_{i,j,M} K(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j), \quad \tau_{i,j,M} = \lim_{n \rightarrow \infty} \frac{1}{nM^2\alpha^2} \sum_{s=1}^n \sum_{m=1}^M \sum_{m'=1}^M \Pr(s \in \mathcal{I}_{m,i} \cap \mathcal{I}_{m',j}),$$

and

$$K(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j) = \lim_{n \rightarrow \infty} \text{Cov}[\bar{\ell}_n(\mathbf{Z}', \boldsymbol{\lambda}_i), \bar{\ell}_n(\mathbf{Z}', \boldsymbol{\lambda}_j)], \quad \bar{\ell}_n(\mathbf{z}, \boldsymbol{\lambda}) = \mathbb{E}[\ell(\mathbf{z}, g_{\boldsymbol{\lambda}}(\mathcal{T}))] - \mathbb{E}[\ell(\mathbf{Z}, g_{\boldsymbol{\lambda}}(\mathcal{T}))],$$

where the expectation is taken over a training set \mathcal{T} of size n and two fresh samples \mathbf{Z}, \mathbf{Z}' from the same distribution.

The regularity conditions are rather mild and discussed further in Appendix C.1. The kernel K reflects the (co-)variability of the losses caused by validation samples. The contribution of training samples only has a higher-order effect. The validation scheme enters the distribution through the quantities $\tau_{i,j,M}$. In what follows, we compute explicit expressions for some popular examples. The following list provides formal definitions for the index sets $\mathcal{I}_{m,j}$.

- (i) (holdout) Let $M = 1$ and $\mathcal{I}_{1,j} = \mathcal{I}_1$ for all $j = 1, \dots, J$, and some size- $\lceil \alpha n \rceil$ index set \mathcal{I}_1 .
- (ii) (reshuffled holdout) Let $M = 1$ and $\mathcal{I}_{1,1}, \dots, \mathcal{I}_{1,J}$ be independently drawn from the uniform distribution over all size- $\lceil \alpha n \rceil$ subsets from $\{1, \dots, n\}$.
- (iii) (M -fold CV) Let $\alpha = 1/M$ and $\mathcal{I}_1, \dots, \mathcal{I}_M$ be a disjoint partition of $\{1, \dots, n\}$, and $\mathcal{I}_{m,j} = \mathcal{I}_m$ for all $j = 1, \dots, J$.
- (iv) (reshuffled M -fold CV) Let $\alpha = 1/M$ and $(\mathcal{I}_{1,j}, \dots, \mathcal{I}_{M,j}), j = 1, \dots, J$, be independently drawn from the uniform distribution over disjoint partitions of $\{1, \dots, n\}$.
- (v) (M -fold holdout) Let $\mathcal{I}_m, m = 1, \dots, M$, be independently drawn from the uniform distribution over size- $\lceil \alpha n \rceil$ subsets of $\{1, \dots, n\}$ and set $\mathcal{I}_{m,j} = \mathcal{I}_m$ for all $m = 1, \dots, M, j = 1, \dots, J$.
- (vi) (reshuffled M -fold holdout) Let $\mathcal{I}_{m,j}, m = 1, \dots, M, j = 1, \dots, J$, be independently drawn from the uniform distribution over size- $\lceil \alpha n \rceil$ subsets of $\{1, \dots, n\}$.

The value of $\tau_{i,j,M}$ for each example is computed explicitly in Appendix E. In all these examples, we in fact have

$$\tau_{i,j,M} = \begin{cases} \sigma^2, & i = j \\ \tau^2 \sigma^2, & i \neq j. \end{cases} \quad (1)$$

for some method-dependent parameters σ, τ shown in Table 1. The parameter σ^2 captures any increase in variance caused by omitting an observation from the validation sets. The parameter τ quantifies a potential decrease in correlation in the loss surface due to reshuffling. More precisely,

Table 1: Exemplary parametrizations in Equation (1) for resamplings; see Appendix E for details.

Method	σ^2	τ^2
holdout (HO)	$1/\alpha$	1
reshuffled HO	$1/\alpha$	α
M -fold CV	1	1
reshuffled M -fold CV	1	1
M -fold HO (subsampling / Monte Carlo CV)	$1 + (1 - \alpha)/M\alpha$	1
reshuffled M -fold HO	$1 + (1 - \alpha)/M\alpha$	$1/(1 + (1 - \alpha)/M\alpha)$

the observed losses $\hat{\mu}(\lambda_i), \hat{\mu}(\lambda_j)$ at distinct HPCs $\lambda_i \neq \lambda_j$ become less correlated when τ is small. Generally, an increase in variance leads to worse generalization performance. The effect of a correlation decrease is less obvious and is studied in detail in the following section.

We make the following observations about the differences between methods in Table 1:

- M -fold CV incurs no increase in variance ($\sigma^2 = 1$) and — because every HPC uses the same folds — no decrease in correlation. Interestingly, the correlation does not even decrease when reshuffling the folds. In any case, all samples are used exactly once as validation and training instance. At least asymptotically, this leads to the same behavior, and reshuffling should have almost no effect on M -fold CV.
- The two (1-fold) holdout methods bear the same $1/\alpha$ increase in variance. This is caused by only using a fraction α of the data as validation samples. Reshuffled holdout also decreases the correlation parameter τ^2 . In fact, if HPCs $\lambda_i \neq \lambda_j$ are evaluated on largely distinct samples, the validation losses $\hat{\mu}(\lambda_i)$ and $\hat{\mu}(\lambda_j)$ become almost independent.
- M -fold holdout also increases the variance, because some samples may still be omitted from validation sets. This increase is much smaller for large M . Accordingly, the correlation is also decreased by less in the reshuffled variant.

2.3 How Reshuffling Affects HPO Performance

In practice, we are mainly interested in the performance of a model trained with the optimal HPC $\hat{\lambda}$. To simplify the analysis, we explore this in the large-sample regime derived in the previous section. Assume

$$\hat{\mu}(\lambda_j) = \mu(\lambda_j) + \epsilon(\lambda_j) \quad (2)$$

where $\epsilon(\lambda)$ is a zero-mean Gaussian process with covariance kernel

$$\text{Cov}(\epsilon(\lambda), \epsilon(\lambda')) = \begin{cases} K(\lambda, \lambda) & \text{if } \lambda = \lambda', \\ \tau^2 K(\lambda, \lambda') & \text{else.} \end{cases} \quad (3)$$

Let $\Lambda \subseteq \{\lambda \in \mathbb{R}^d : \|\lambda\| \leq 1\}$ with $|\Lambda| = J < \infty$ be the set of hyperparameters. Theorem 2.2 ahead gives a bound on the expected regret $\mathbb{E}[\mu(\hat{\lambda}) - \mu(\lambda^*)]$. It depends on several quantities characterizing the difficulty of the HPO problem. The constant

$$\kappa = \sup_{\|\lambda\|, \|\lambda'\| \leq 1} \frac{|K(\lambda, \lambda) - K(\lambda, \lambda')|}{K(\lambda, \lambda) \|\lambda - \lambda'\|^2}.$$

can be interpreted as a measure of correlation of the process ϵ . In particular, $\text{Corr}(\epsilon(\lambda), \epsilon(\lambda')) \geq 1 - \kappa \|\lambda - \lambda'\|^2$. The constant is small when ϵ is strongly correlated, and large otherwise. Further, define η as the minimal number such that any η -ball contained in $\{\|\lambda\| \leq 1\}$ contains at least one element of Λ . It measures how densely the set of candidate HPCs Λ covers set of all possible HPCs. If Λ is a deterministic uniform grid, we have about $\eta \approx J^{-1/d}$. Similarly, Lemma D.1 in the Appendix shows that $\eta \lesssim J^{-1/2d}$ when randomly sampling HPCs. Finally, the constant

$$m = \sup_{\lambda \in \Lambda} \frac{|\mu(\lambda) - \mu(\lambda^*)|}{\|\lambda - \lambda^*\|^2},$$

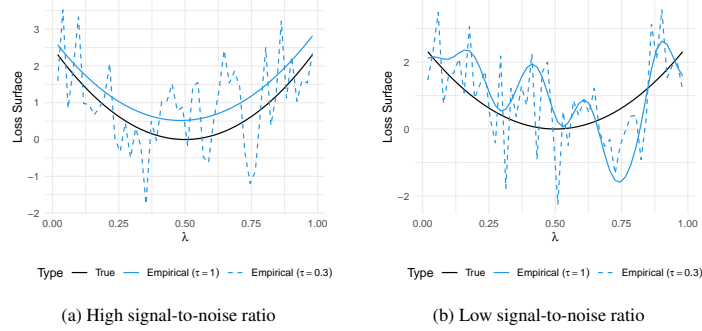


Figure 1: Example of reshuffled empirical loss yielding a worse (left) and better (right) minimizer.

measures the local curvature at the minimum of the loss surface μ . Finding an HPC λ close to the theoretical optimum λ^* is easier when the minimum is more pronounced (large m). On the other hand, the regret $\mu(\lambda) - \mu(\lambda^*)$ is also punishing mistakes more quickly. Defining $\log(x)_+ = \max\{0, \log(x)\}$, we can now state our main result.

Theorem 2.2. *Let $\hat{\mu}$ follow the Gaussian process model (2). Suppose $\kappa < \infty$, $0 < \underline{\sigma}^2 \leq \text{Var}[\epsilon(\lambda)] \leq \sigma^2 < \infty$ for all $\lambda \in \Lambda$, and $m > 0$. Then*

$$\mathbb{E}[\mu(\hat{\lambda}) - \mu(\lambda^*)] \leq \sigma\sqrt{d}[8 + B(\tau) - A(\tau)].$$

where

$$B(\tau) = 48 \left[\sqrt{1 - \tau^2} \sqrt{\log J} + \tau \sqrt{1 + \log(3\kappa)_+} \right], \quad A(\tau) = \sqrt{1 - \tau^2} (\underline{\sigma}/\sigma) \sqrt{\log \left(\frac{\sigma}{2m\eta^2} \right)_+}.$$

The numeric constants result from several simplifications in a worst-case analysis, which lowers their practical relevance. A qualitative analysis of the bound is still insightful. The bound is increasing in σ and d , indicating that the HPO problem is harder when there is a lot of noise or there are many parameters to tune. The terms $B(\tau)$ and $A(\tau)$ have conceptual interpretations:

- The term $B(\tau)$ quantifies how likely it is to pick a bad $\hat{\lambda}$ because of bad luck: a λ far away from λ^* had such a small $\epsilon(\lambda)$ that it outweighs the increase in μ . Such events are more likely when the process ϵ is weakly correlated. Accordingly, $B(\tau)$ is decreasing in τ and increasing in κ .
- The term $A(\tau)$ quantifies how likely it is to pick a good $\hat{\lambda}$ by luck: a λ close to λ^* had such a small $\epsilon(\lambda)$ that it overshoots all the other fluctuations. Also such events are more likely when the process ϵ is weakly correlated. Accordingly, the term $A(\tau)$ is decreasing in τ .

The B , as stated, is unbounded, but a closer inspection of the proof shows that it is upper bounded by $\sqrt{\log J}$. This bound is attained only in the unrealistic scenario when the validation losses are essentially uncorrelated across all HPCs. The term A is bounded from below by zero, which is also the worst case because the term enters our regret bound with a negative sign.

Both A and B are decreasing in the reshuffling parameter τ . There are two regimes. If $\sigma/2m\eta^2 \leq e$, then $A(\tau) = 0$ and reshuffling cannot lead to an improvement of the bound. The term $\sigma/2m\eta^2$ can be interpreted as noise-to-signal ratio (relative to the grid density). If the signal is much stronger than the noise, the HPO problem is so easy that reshuffling will not help. This situation is illustrated in Figure 1a.

If on the other hand $\sigma/2m\eta^2 > e$, the terms $A(\tau)$ and $B(\tau)$ enter the bound with opposing signs. This creates tension: reshuffling between HPCs increases $B(\tau)$, which is countered by a decrease in $A(\tau)$. So which scenarios favor reshuffling? When the process ϵ is strongly correlated, κ is small and reshuffling (decreasing τ) incurs a high cost in $B(\tau)$. This is intuitive: When there is strong

correlation, the validation loss surface $\hat{\mu}$ is essentially just a vertical shift of μ . Finding the optimal λ is then almost as easy as if we would know μ , and decorrelating the surface through reshuffling would make it unnecessarily hard. When ϵ is less correlated (κ large) however, reshuffling does not hurt the term $B(\tau)$ as much, but we can reap all the benefits of increasing $A(\tau)$. Here, the effect of reshuffling can be interpreted as hedging against the catastrophic case where all $\hat{\mu}(\lambda)$ close to the optimal λ^* are simultaneously dominated by a region of bad hyperparameters. This is illustrated in Figure 1b.

3 Simulation Study

To test our theoretical understanding of the potential benefits of reshuffling resampling splits during HPO, we conduct a simulation study. This study helps us explore the effects of reshuffling in a controlled setting.

3.1 Design

We construct a univariate quadratic loss surface function $\mu : \Lambda \subset \mathbb{R} \mapsto \mathbb{R}, \lambda \rightarrow m(\lambda - 0.5)^2/2$ which we want to minimize. The global minimum is given at $\mu(0.5) = 0$. Combined with a kernel for the noise process ϵ as in Equation (3), this allows us to simulate an objective as observed during HPO by sampling $\hat{\mu}(\lambda) = \mu(\lambda) + \epsilon(\lambda)$. We use a squared exponential kernel $K(\lambda, \lambda') = \sigma_\epsilon^2 \exp(-\kappa(\lambda - \lambda')^2/2)$ that is plugged into the covariance kernel of the noise process ϵ in Equation (3). The parameters m and κ in our simulation setup correspond exactly to the curvature and correlation constants from the previous sections. Recall that Theorem 2.2 states that the effect of reshuffling strongly depends on the curvature m of the loss surface μ (a larger m implies a stronger curvature) and the constant κ as a measure of correlation of the noise ϵ (a larger κ implies weaker correlation). Combined with the possibility to vary τ in the covariance kernel of ϵ , we can systematically investigate how curvature of the loss surface, correlation of the noise and the extent of reshuffling affect optimization performance. In each simulation run, we simulate the observed objective $\hat{\mu}(\lambda)$, identify the minimizer $\hat{\lambda} = \arg \min_{\lambda \in \Lambda} \hat{\mu}(\lambda)$, and calculate its true risk, $\mu(\hat{\lambda})$. We repeat this process 10000 times for various combinations of τ , m , and κ .

3.2 Results

Figure 2 visualizes the true risk of the configuration $\hat{\lambda}$ that minimizes the observed objective. We observe that for a loss surface with low curvature (i.e., $m \leq 2$), reshuffling is beneficial (lower values of τ resulting in a better true risk of the configuration that optimizes the observed objective) as long as the noise process is not too correlated (i.e., $\kappa \geq 1$). As soon as the noise process is more strongly correlated, even flat valleys of the true risk μ remain clearly visible in the observed risk $\hat{\mu}$, and reshuffling starts to hurt the optimization performance. Moving to scenarios of high curvature, the general relationship of m and κ remains the same, but reshuffling starts to hurt optimization performance already with weaker correlation in the noise. In summary, the simulations show that in cases of low curvature of the loss surface, reshuffling (reducing τ) tends to improve the true risk of the optimized configuration, especially when the loss surface is flat (small m) and the noise is not strongly correlated (i.e., κ is large). This exactly confirms our theoretical predictions from the previous section.

4 Benchmark Experiments

In this section, we present benchmark experiments of real-world HPO problems where we investigate the effect of reshuffling resampling splits during HPO. First, we discuss the experimental setup. Second, we present results for HPO using random search (Bergstra & Bengio, 2012). Third, we also show the effect of reshuffling when applied in BO using HEBO (Cowen-Rivers et al., 2022) and SMAC3 (Lindauer et al., 2022). Recall that our theoretical insight suggests that 1) reshuffling might be beneficial during HPO and 2) holdout should be affected the most by reshuffling and other resamplings should only be affected to a lesser extent.

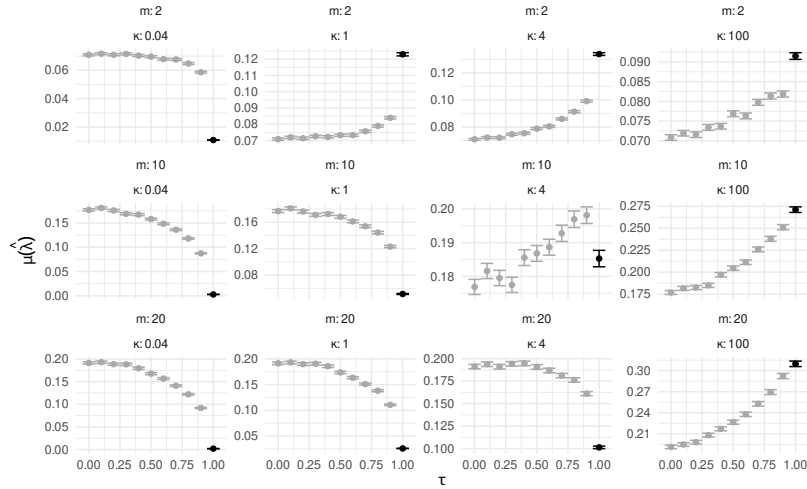


Figure 2: Mean true risk (lower is better) of the configuration minimizing the observed objective systematically varied with respect to curvature m , correlation strength κ of the noise (a larger κ implying weaker correlation), and extent of reshuffling τ (lower τ increasing reshuffling). A τ of 1 indicates no reshuffling. Error bars represent standard errors.

4.1 Experimental Setup

As benchmark tasks, we use a set of standard HPO problems defined on small- to medium-sized tabular datasets for binary classification. We suspect the effect of the resampling variant used and whether the resampling is reshuffled to be larger for smaller datasets, where the variance of the validation loss estimator is naturally higher. Furthermore, from a practical perspective, this also ensures computational feasibility given the large number of HPO runs in our experiments. We systematically vary the learning algorithm, optimized performance metric, resampling method, whether the resampling is reshuffled, and the size of the dataset used for training and validation during HPO. Below, we outline the general experimental design and refer to Appendix F for details.

We used a subset of the datasets defined by the AutoML benchmark (Gijsbers et al., 2024), treating these as data generating processes (DGPs; Hothorn et al., 2005). We only considered datasets with less than 100 features to reduce the required computation time and required the number of observations to be between 10000 and 1000000; for further details see Appendix F.1. Our aim was to robustly measure the generalization performance when varying the size n , which, as defined in Section 2 denotes the size of the combined data for model selection, so one training and validation set combined. First, we sampled 5000 data points per dataset for robust assessment of the generalization error; these points are not used during HPO in any way. Then, from the remaining points we sampled tasks with $n \in \{500, 1000, 5000\}$.

We selected CatBoost (Prokhorenkova et al., 2018) and XGBoost (Chen & Guestrin, 2016) for their state-of-the-art performance on tabular data (Grinsztajn et al., 2022; Borisov et al., 2022; McElfresh et al., 2023; Kohli et al., 2024). Additionally, we included an Elastic Net (Zou & Hastie, 2005) to represent a linear baseline with a smaller search space and a funnel-shaped MLP (Zimmer et al., 2021) as a cost-effective neural network baseline. We provide details regarding training pipelines and search spaces in Appendix F.2.

We conduct a random search with 500 HPC evaluations for every resampling strategy we described in Table 1, for both fixed and reshuffled splits. We always use 80/20 train-validation splits for holdout

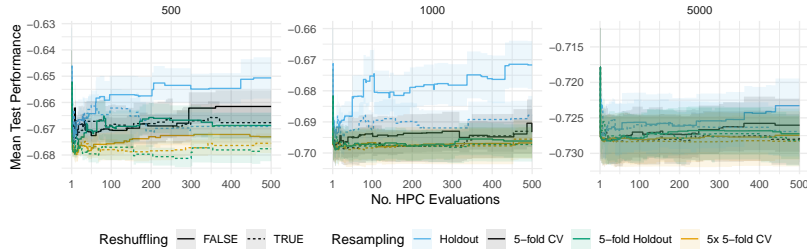


Figure 3: Average test performance (negative ROC AUC) of the incumbent for XGBoost on dataset albert for increasing n (train-validation sizes, columns). Shaded areas represent standard errors.

and 5-fold CVs, so that training set size (and negative estimation bias) are the same. Anytime test performance of an HPO run is assessed by re-training the current incumbent (i.e. the best HPC until the current HPO iteration based on validation performance) on all available train and validation data and evaluating its performance on the outer test set. Note we do this for scientific evaluation in this experiment; obviously, this is not possible in practice. Using random search allows us to record various metrics and afterwards simulate optimizing for different ones, specifically, we recorded accuracy, area under the ROC curve (ROC AUC) and logloss.

We also investigated the effect of reshuffling on two state-of-the-art BO variants (Eggensperger et al., 2021; Turner et al., 2021), namely HEBO (Cowen-Rivers et al., 2022) and SMAC3 (Lindauer et al., 2022). The experimental design was the same as for random search, except for the budget, which we reduced from 500 HPCs to 250 HPCs, and only optimized ROC AUC.

4.2 Experimental Results

In the following, we focus on the results obtained using ROC AUC. We present aggregated results over different tasks, learning algorithms and replications to get a general understanding of the effects. Unaggregated results and results involving accuracy and logloss can be found in Appendix G.

Results of Reshuffling Different Resamplings For each resampling (holdout, 5-fold holdout, 5-fold CV, and 5x 5-fold CV), we empirically analyze the effect of reshuffling train and validation splits during HPO.

In Figure 3 we exemplarily show how test performance develops over the course of an HPO run on a single task for different resamplings (with and without reshuffling). Naturally, test performance does not necessarily increase in a monotonic fashion, and especially holdout without reshuffling tends to be unstable. Its reshuffled version results in substantially better test performance.

Next, we look at the relative *improvement* (compared to standard 5-fold CV, which we consider our baseline) with respect to *test* ROC AUC performance of the incumbent over time in Figure 4, i.e., the difference in test performance of the incumbent between standard 5-fold CV and a different resampling protocol; hence a positive difference tells us how much better in test error we are, if we would have chosen the other protocol instead 5-fold CV. We observe that reshuffling generally results in equal or better performance compared to the same resampling protocol without reshuffling. For 5-fold holdout and especially 5-fold CV and 5x 5-fold CV, reshuffling has a smaller effect on relative test performance improvement, as expected. Holdout is affected the most by reshuffling and results in substantially better relative test performance compared to standard holdout. We also observe that an HPO protocol based on reshuffled holdout results in similar final test performance as standard 5-fold CV while overall being substantially cheaper due to requiring less model fits per HPC evaluation. In Appendix G.2, we further provide an ablation study on the number of folds when using M -fold holdout, where we observed that – in line with our theory – the more folds are used, the less reshuffling affects M -fold holdout.

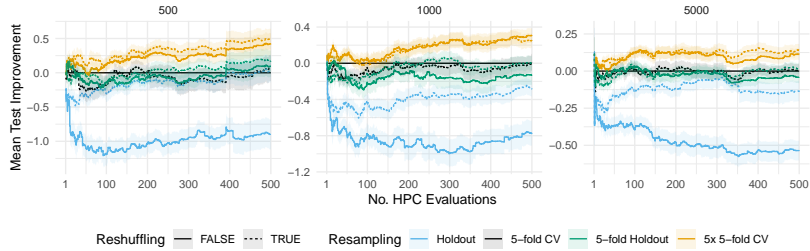


Figure 4: Average improvement (compared to standard 5-fold CV) with respect to test performance (ROC AUC) of the incumbent over different tasks, learning algorithms and replications separately for increasing n (train-validation sizes, columns). Shaded areas represent standard errors.

However, this general trend can vary for certain combinations of classifier and performance metric, see Appendix G. Especially for logloss, we observed that reshuffling rarely is beneficial; see the discussion in Section 5. Finally, the different resamplings generally behave as expected. The more we are willing to invest compute resources into a more intensive resampling like 5-fold CV or 5x 5-fold CV, the better the generalization performance of the final incumbent.

Results for BO and Reshuffling Figure 5 shows that, generally HEBO and SMAC3 outperform random search with respect to generalization performance (i.e., comparing HEBO and SMAC3 to random search under standard holdout, or comparing under reshuffled holdout). More interestingly, HEBO, SMAC3 and random search all strongly benefit from reshuffling. Moreover, the performance gap between HEBO and random search but also SMAC3 and random search narrows when the resampling is reshuffled, which is an interesting finding of its own: As soon as we are concerned with generalization performance of HPO and not only investigate validation performance during optimization, the choice of optimizer might have less impact on final generalization performance compared to other choices such as whether the resampling is reshuffled during HPO or not. We present results for BO and reshuffling for different resamplings in Appendix G.

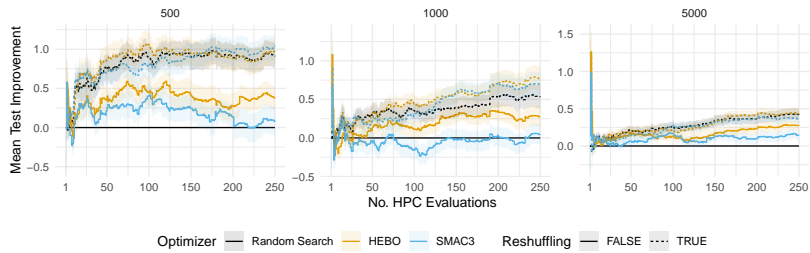


Figure 5: Average improvement (compared to random search on standard holdout) with respect to test performance (ROC AUC) of the incumbent over tasks, learning algorithms and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

5 Discussion

In the previous sections, we have shown theoretically and empirically that reshuffling can enhance generalization performance of HPO. The main purpose of this article is to draw attention to this

surprising fact about a technique that is simple but rarely discussed. Our work goes beyond a preliminary experimental study on reshuffling (Lévesque, 2018), in that we also study the effect of reshuffling on random search, multiple metrics and learning algorithms, and most importantly, for the first time, we provide a theoretical analysis that explains why reshuffling can be beneficial.

Limitations To unveil the mechanisms underlying the reshuffling procedures, our theoretical analysis relies on an asymptotic approximation of the empirical loss surface. This allows us to operate on Gaussian loss surfaces, which exhibit convenient concentration and anti-concentration properties required in our proof. The latter are lacking for general distributions, which explains our asymptotic approach. The analysis was further facilitated by a loss stability assumption regarding the learning algorithms that is generally rather mild; see the discussion in Bayle et al. (2020). However, it typically fails for highly sensitive losses, which has practical consequences. In fact, Figure 9 in Appendix G shows that reshuffling usually hurts generalization for the logloss and small sample sizes. It is still an open question whether this problem can be fixed by less naive implementations of the technique. Another limitation is our focus on generalization after search through a fixed, finite set of candidates. This largely ignores the dynamic nature of many HPO algorithms, which would greatly complicate our analysis. Finally, our experiments are limited in that we restricted ourselves to tabular data and binary classification and we avoided extremely small or large datasets.

Relation to Overfitting The fact that generalization performance can decrease during HPO (or computational model selection in general) is sometimes known as oversearching, overtuning, or overfitting to the validation set (Quinlan & Cameron-Jones, 1995; Escalante et al., 2009; Koch et al., 2010; Igel, 2012; Bischl et al., 2023), but has arguably not been studied very thoroughly. Given recent theoretical (Feldman et al., 2019) and empirical (Purucker & Beel, 2023) findings, we expect less overtuning on multi-class datasets, making it interesting to see how reshuffling would affect the generalization performance.

Several works suggest strategies to counteract this effect. First, LOOCVCV proposes a conservative choice of incumbents (Ng, 1997) at the cost of leave-one-out analysis or an additional hyperparameter. Second, it is possible to use an extra *selection set* (Igel, 2012; Lévesque, 2018; Mohr et al., 2018) at the cost of reduced training data, which was found to lead to reduced overall performance (Lévesque, 2018). Third, by using early stopping one can stop hyperparameter optimization before the generalization performance degrades again. This was so far demonstrated to be able to save compute budget at only marginally reduced performance, but also requires either a sensitivity hyperparameter or correct estimation of the variance of the generalization estimate and was only developed for cross-validation so far (Makarova et al., 2022). Reshuffling itself is orthogonal to these proposals and a combination with the above-mentioned methods might result in further improvements.

Outlook Generally, the related literature detects overfitting to the validation set either visually (Ng, 1997) or by measuring it (Koch et al., 2010; Igel, 2012; Fabris & Freitas, 2019). Developing a unified formal definition of the above-mentioned terms and thoroughly analyzing the effect of decreased generalization performance after many HPO iterations and how it relates to our measurements of the validation performance is an important direction for future work.

We further found, both theoretically and experimentally, that investing more resources when evaluating each HPC can result in better final HPO performance. To reduce the computational burden on HPO again, we suggest further investigating the use of adaptive CV techniques, as proposed by AutoWEKA (Thornton et al., 2013) or under the name Lazy Paired Hyperparameter Tuning (Zheng & Bilenko, 2013). Designing more advanced HPO algorithms exploiting the reshuffling effect should be a promising avenue for further research.

Acknowledgments and Disclosure of Funding

We thank Martin Binder and Florian Karl for helpful discussions. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8). Lennart Schneider acknowledges funding from the LMU Mentoring Program of the Faculty of Mathematics, Informatics and Statistics.

References

- Arlot, S. and Celisse, A. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40 – 79, 2010. B
- Austern, M. and Zhou, W. Asymptotics of cross-validation. *arXiv:2001.11111 [math.ST]*, 2020. C.1
- Awad, N., Mallik, N., and Hutter, F. DEHB: Evolutionary hyperband for scalable, robust and efficient Hyperparameter Optimization. In Zhou, Z. (ed.), *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pp. 2147–2153, 2021. B
- Bayle, P., Bayle, A., Janson, L., and Mackey, L. Cross-validation confidence intervals for test error. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.-F., and Lin, H. (eds.), *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pp. 16339–16350. Curran Associates, 2020. 5, C.1, C.1, C.1
- Bergman, E., Purucker, L., and Hutter, F. Don't waste your time: Early stopping cross-validation. In Eggensperger, K., Garnett, R., Vanschoren, J., Lindauer, M., and Gardner, J. (eds.), *Proceedings of the Third International Conference on Automated Machine Learning*, volume 256 of *Proceedings of Machine Learning Research*, pp. 9/1–31. PMLR, 2024. B
- Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. 4, B
- Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A., Deng, D., and Lindauer, M. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, pp. e1484, 2023. 1, 5, B
- Blum, A., Kalai, A., and Langford, J. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, COLT '99, pp. 203–208, 1999. B
- Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2022. 4.1
- Bouckaert, Remco and Frank, E. Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms. In Dai, H., Srikant, R., and Zhang, C. (eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 3–12. Springer, 2004. B
- Bousquet, O. and Zhivotovskiy, N. Fast classification rates without standard margin assumptions. *Information and Inference: A Journal of the IMA*, 10(4):1389–1421, 2021. C.1
- Bouthillier, X., Delaunay, P., Bronzi, M., Trofimov, A., Nichyporuk, B., Szeto, J., Sepahvand, N. M., Raff, E., Madan, K., Voleti, V., Kahou, S. E., Michalski, V., Arbel, T., Pal, C., Varoquaux, G., and Vincent, P. Accounting for variance in machine learning benchmarks. In Smola, A., Dimakis, A., and Stoica, I. (eds.), *Proceedings of Machine Learning and Systems 3*, volume 3, pp. 747–769, 2021. B
- Buczak, P., Groll, A., Pauly, M., Rehof, J., and Horn, D. Using sequential statistical tests for efficient hyperparameter tuning. *AStA Advances in Statistical Analysis*, 108(2):441–460, 2024. B
- Cawley, G. and Talbot, N. On Overfitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010. B
- Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In Krishnapuram, B., Shah, M., Smola, A., Aggarwal, C., Shen, D., and Rastogi, R. (eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*, pp. 785–794. ACM Press, 2016. 4.1
- Cowen-Rivers, A., Lyu, W., Tutunov, R., Wang, Z., Grosnit, A., Griffiths, R., Maraval, A., Jianye, H., Wang, J., Peters, J., and Ammar, H. HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349, 2022. 4, 4.1, B

- Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. 1
- Dietterich, T. G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998. 1
- Dunias, Z., Van Calster, B., Timmerman, D., Boulesteix, A.-L., and van Smeden, M. A comparison of hyperparameter tuning procedures for clinical prediction models: A simulation study. *Statistics in Medicine*, 43(6):1119–1134, 2024. B
- Eggensperger, K., Lindauer, M., Hoos, H., Hutter, F., and Leyton-Brown, K. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1):15–41, 2018. 5
- Eggensperger, K., Lindauer, M., and Hutter, F. Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, pp. 861–893, 2019. B
- Eggensperger, K., Müller, P., Mallik, N., Feurer, M., Sass, R., Klein, A., Awad, N., Lindauer, M., and Hutter, F. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In Vanschoren, J. and Yeung, S. (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran Associates, 2021. 4.1, B
- Escalante, H., Montes, M., and Sucar, E. Particle Swarm Model Selection. *Journal of Machine Learning Research*, 10:405–440, 2009. 5
- Fabris, F. and Freitas, A. Analysing the overfit of the auto-sklearn automated machine learning tool. In Nicosia, G., Pardalos, P., Umeton, R., Giuffrida, G., and Sciacca, V. (eds.), *Machine Learning, Optimization, and Data Science*, volume 11943 of *Lecture Notes in Computer Science*, pp. 508–520, 2019. 5
- Falkner, S., Klein, A., and Hutter, F. BOHB: Robust and efficient Hyperparameter Optimization at scale. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pp. 1437–1446. Proceedings of Machine Learning Research, 2018. B
- Feldman, V., Frostig, R., and Hardt, M. The advantages of multiple classes for reducing overfitting from test set reuse. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, volume 97, pp. 1892–1900. Proceedings of Machine Learning Research, 2019. 5
- Feurer, M. and Hutter, F. Hyperparameter Optimization. In Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.), *Automated Machine Learning: Methods, Systems, Challenges*, chapter 1, pp. 3 – 38. Springer, 2019. Available for free at <http://automl.org/book>. 1, B
- Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. Auto-Sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 23(261):1–61, 2022. B
- Garnett, R. *Bayesian Optimization*. Cambridge University Press, 2023. 1, B
- Gijbbers, P., Bueno, M., Coors, S., LeDell, E., Poirier, S., Thomas, J., Bischl, B., and Vanschoren, J. AMLB: an automl benchmark. *Journal of Machine Learning Research*, 25(101):1–65, 2024. 4.1
- Giné, E. and Nickl, R. *Mathematical Foundations of Infinite-Dimensional Statistical Models*, volume 40. Cambridge University Press, 2016. C.2
- Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, pp. 507–520, 2022. 4.1
- Guyon, I., Alamdari, A., Dror, G., and Buhmann, J. Performance prediction challenge. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 2006. 1
- Guyon, I., Saffari, A., Dror, G., and Cawley, G. Model selection: Beyond the Bayesian/Frequentist divide. *Journal of Machine Learning Research*, 11:61–87, 2010. B

- Guyon, I., Bennett, K., Cawley, G., Escalante, H. J., Escalera, S., Ho, T. K., Macià, N., Ray, B., Saeed, M., Statnikov, A., and Viegas, E. Design of the 2015 ChaLearn AutoML challenge. In *2015 International Joint Conference on Neural Networks (IJCNN'15)*, pp. 1–8. International Neural Network Society and IEEE Computational Intelligence Society, IEEE, 2015. B
- Guyon, I., Sun-Hosoya, L., Boullé, M., Escalante, H., Escalera, S., Liu, Z., Jajetic, D., Ray, B., Saeed, M., Sebag, M., Statnikov, A., Tu, W., and Viegas, E. Analysis of the AutoML Challenge Series 2015-2018. In Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.), *Automated Machine Learning: Methods, Systems, Challenges*, chapter 10, pp. 177–219. Springer, 2019. Available for free at <http://automl.org/book>. B
- Hansen, N. and Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary C.*, 9(2):159–195, 2001. 1
- Hothorn, T., Leisch, F., Zeileis, A., and Hornik, K. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, 14(3):675–699, 2005. 4.1, F.1
- Igel, C. A note on generalization loss when evolving adaptive pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, 17(3):345–352, 2012. 5, 5
- Jamieson, K. and Talwalkar, A. Non-stochastic best arm identification and Hyperparameter Optimization. In Gretton, A. and Robert, C. (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS'16)*, volume 51. Proceedings of Machine Learning Research, 2016. B
- Kadra, A., Janowski, M., Wistuba, M., and Grabocka, J. Scaling laws for hyperparameter optimization. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 47527–47553, 2023. B
- Kallenberg, O. *Foundations of modern probability*, volume 2. Springer, 1997. D
- Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In Singh, A. and Zhu, J. (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS'17)*, volume 54. Proceedings of Machine Learning Research, 2017. B
- Koch, P., Konen, W., Flasch, O., and Bartz-Beielstein, T. Optimizing support vector machines for stormwater prediction. Technical Report TR10-2-007, Technische Universität Dortmund, 2010. Proceedings of Workshop on Experimental Methods for the Assessment of Computational Systems joint to PPSN2010. 5, 5
- Kohli, R., Feurer, M., Bischl, B., Eggensperger, K., and Hutter, F. Towards quantifying the effect of datasets for benchmarking: A look at tabular machine learning. In *Data-centric Machine Learning (DMLR) workshop at the International Conference on Learning Representations (ICLR)*, 2024. 4.1
- Lang, M., Kotthaus, H., Marwedel, P., Weihs, C., Rahnenführer, J., and Bischl, B. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation*, 85:62–76, 2015. B
- Larcher, C. and Barbosa, H. Evaluating models with dynamic sampling holdout in auto-ml. *SN Computer Science*, 3(506), 2022. 1
- Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. B
- Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. SMAC3: A versatile bayesian optimization package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. 4, 4.1, B
- Loshchilov, I. and Hutter, F. CMA-ES for Hyperparameter Optimization of deep neural networks. In *International Conference on Learning Representations Workshop track*, 2016. Published online: iclr.cc. B

- Lévesque, J. *Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces*. PhD thesis, Université Laval, 2018. 1, 5, 5
- Makarova, A., Shen, H., Perrone, V., Klein, A., Faddoul, J., Krause, A., Seeger, M., and Archambeau, C. Automatic termination for hyperparameter optimization. In Guyon, I., Lindauer, M., van der Schaar, M., Hutter, F., and Garnett, R. (eds.), *Proceedings of the First International Conference on Automated Machine Learning*. Proceedings of Machine Learning Research, 2022. 5
- Mallik, N., Bergman, E., Hvarfner, C., Stoll, D., Janowski, M., Lindauer, M., Nardi, L., and Hutter, F. PriorBand: Practical hyperparameter optimization in the age of deep learning. In Oh, A., Neumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*. Curran Associates, 2023. B
- McElfresh, D., Khandagale, S., Valverde, J., Prasad, C., V., Ramakrishnan, G., Goldblum, M., and White, C. When do neural nets outperform boosted trees on tabular data? In Oh, A., Neumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*, pp. 76336–76369. Curran Associates, 2023. 4.1, F.2
- Mohr, F., Wever, M., and Hüllermeier, E. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8-10):1495–1515, 2018. 5, B
- Molinaro, A., Simon, R., and Pfeiffer, R. Prediction error estimation: A comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307, 2005. B
- Nadeau, C. and Bengio, Y. Inference for the generalization error. In Solla, S., Leen, T., and Müller, K. (eds.), *Proceedings of the 13th International Conference on Advances in Neural Information Processing Systems (NeurIPS'99)*. The MIT Press, 1999. 1
- Nadeau, C. and Bengio, Y. Inference for the generalization error. *Machine Learning*, 52:239–281, 2003. 1
- Ng, A. Preventing “overfitting” of cross-validation data. In Fisher, D. H. (ed.), *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, pp. 245–253. Morgan Kaufmann Publishers, 1997. 5, 5
- Pfisterer, F., Schneider, L., Moosbauer, J., Binder, M., and Bischl, B. YAHPO Gym – an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In Guyon, I., Lindauer, M., van der Schaar, M., Hutter, F., and Garnett, R. (eds.), *Proceedings of the First International Conference on Automated Machine Learning*. Proceedings of Machine Learning Research, 2022. B, 5
- Pineda Arango, S., Jomaa, H., Wistuba, M., and Grabocka, J. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. In Vanschoren, J. and Yeung, S. (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran Associates, 2021. B, 5
- Probst, P., Boulesteix, A., and Bischl, B. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019. 1
- Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A., and Gulin, A. Catboost: Unbiased boosting with categorical features. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, pp. 6639–6649. Curran Associates, 2018. 4.1
- Purucker, L. and Beel, J. CMA-ES for post hoc ensembling in autml: A great success and salvageable failure. In Faust, A., Garnett, R., White, C., Hutter, F., and Gardner, J. R. (eds.), *Proceedings of the Second International Conference on Automated Machine Learning*, volume 224 of *Proceedings of Machine Learning Research*, pp. 1/1–23. PMLR, 2023. 5
- Quinlan, J. and Cameron-Jones, R. Oversearching and layered search in empirical learning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI'95*, pp. 1019–1024, 1995. 5

- Rao, R., Fung, G., and Rosales, R. On the dangers of cross-validation. an experimental evaluation. In *Proceedings of the 2008 SIAM International Conference on Data Mining (SDM)*, pp. 588–596, 2008. B
- Salinas, D., Seeger, M., Klein, A., Perrone, V., Wistuba, M., and Archambeau, C. Syne Tune: A library for large scale hyperparameter tuning and reproducible research. In Guyon, I., Lindauer, M., van der Schaar, M., Hutter, F., and Garnett, R. (eds.), *Proceedings of the First International Conference on Automated Machine Learning*, pp. 16–1. Proceedings of Machine Learning Research, 2022. B
- Schaffer, C. Selecting a classification method by cross-validation. *Machine Learning Journal*, 13: 135–143, 1993. B
- Swersky, K., Snoek, J., and Adams, R. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stats.ML]*, 2014. B
- Talagrand, M. *The generic chaining: upper and lower bounds of stochastic processes*. Springer Science & Business Media, 2005. C.2
- Thornton, C., Hutter, F., Hoos, H., and Leyton-Brown, K. Auto-WEKA: Combined selection and Hyperparameter Optimization of classification algorithms. In Dhillon, I., Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., and Uthurusamy, R. (eds.), *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pp. 847–855. ACM Press, 2013. 5, B
- Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., and Guyon, I. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the Black-Box Optimization Challenge 2020. In Escalante, H. and Hofmann, K. (eds.), *Proceedings of the Neural Information Processing Systems Track Competition and Demonstration*, pp. 3–26. Curran Associates, 2021. 4.1
- van der Vaart, A. *Asymptotic statistics*, volume 3. Cambridge university press, 2000. C.1
- van Erven, T., Grünwald, P., Mehta, N., Reid, M., and Williamson, R. Fast rates in statistical and online learning. *Journal of Machine Learning Research*, 16(54):1793–1861, 2015. C.1
- van Rijn, J. and Hutter, F. Hyperparameter importance across datasets. In Guo, Y. and Farooq, F. (eds.), *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18)*, pp. 2367–2376. ACM Press, 2018. 1
- Vanschoren, J., van Rijn, J., Bischl, B., and Torgo, L. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2014. 4
- Wainer, J. and Cawley, G. Empirical Evaluation of Resampling Procedures for Optimising SVM Hyperparameters. *Journal of Machine Learning Research*, 18:1–35, 2017. B
- Wainwright, M. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge university press, 2019. C.2
- Wistuba, M., Schilling, N., and Schmidt-Thieme, L. Scalable Gaussian process-based transfer surrogates for Hyperparameter Optimization. *Machine Learning*, 107(1):43–78, 2018. G.1
- Wu, J., Toscano-Palmerin, S., Frazier, P., and Wilson, A. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. In Peters, J. and Sontag, D. (eds.), *Proceedings of The 36th Uncertainty in Artificial Intelligence Conference (UAI'20)*, pp. 788–798. PMLR, 2020. B
- Zheng, A. and Bilenko, M. Lazy paired hyper-parameter tuning. In Rossi, F. (ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, pp. 1924–1931, 2013. 5, B
- Zimmer, L., Lindauer, M., and Hutter, F. Auto-Pytorch: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:3079–3090, 2021. 4.1, F.2
- Zou, H. and Hastie, T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 2005. 4.1

A Notation

Table 2: Notation table. We discuss all symbols used in the main paper.

\mathbf{X}_i	Random vector, describing the features
Y_i	Random variable, describing the target
$\mathbf{Z}_i = (\mathbf{X}_i, Y_i)$	Data point
$\mathcal{D} = \{\mathbf{Z}_i\}_{i=1}^n$	Dataset consisting of <i>iid</i> random variables
n	Number of observations
g	Inducer/ML algorithm
h	Model, created by the inducer via $h = g_\lambda(\mathcal{D})$
λ	Hyperparameter configuration
Λ	Finite set of all hyperparameter configurations
J	$ \Lambda $, i.e., the number of hyperparameter configurations
g_{λ_j}	Hyperparameterized inducer
$\mu(\lambda)$	Expected loss of a hyperparameterized inducer on the distribution of a dataset
$\ell(\mathcal{Z}, h)$	Loss of a model h on a fresh observation \mathcal{Z}
M	Number of folds in M -fold cross-validation
α	Percentage of samples to be used for validation
$\mathcal{I}_{1,j}, \dots, \mathcal{I}_{M,j} \subset \{1, \dots, n\}$	M sets of validation indices, to be used for evaluating λ_j
$\mathcal{V}_{m,j}$	Validation data for fold m and configuration λ_j
$\mathcal{T}_{m,j}$	Training data for fold m and configuration λ_j
$L(\mathcal{V}_{m,j}, g_{\lambda_j}(\mathcal{T}_{m,j}))$	Validation loss for fold m and configuration λ_j
$\bar{\mu}(\lambda_j)$	M -fold validation loss
σ^2	Increase in variance of validation loss caused by resampling
τ^2	Decrease in correlation among validation losses caused by reshuffling
$\tau_{i,j,M}$	Resampling-related component of validation loss covariance
$K(\cdot, \cdot)$	Kernel capturing the covariance of the pointwise losses between two HPCs
$\epsilon(\lambda_j)$	Zero-mean Gaussian process, see Equation (2)
d	Number of hyperparameters
κ	Curvature constant of covariance kernel
η	Density of hyperparameter set Λ
m	Local curvature at the minimum of the loss surface μ
σ	Lower bound on the noise level
$B(\tau)$	Part of the regret bound penalizing reshuffling
$A(\tau)$	Part of the regret bound rewarding reshuffling

B Extended Related Work

Due to the black box nature of the HPO problem (Feurer & Hutter, 2019; Bischl et al., 2023), gradient free, zeroth-order optimization algorithms such as BO (Garnett, 2023), Evolutionary Strategies (Loshchilov & Hutter, 2016) or a simple random search (Bergstra & Bengio, 2012) have become standard optimization algorithms to tackle vanilla HPO problems.

In the last decade, most research on HPO has been concerned with constructing new algorithms that excel at finding configurations with a low estimated generalization error. Examples include BO variants such as HEBO (Cowen-Rivers et al., 2022) or SMAC3 (Lindauer et al., 2022). Another direction of HPO research has been concerned with speeding up the HPO process to allow more efficient spending of compute resources. Multifidelity HPO, for example, turns the black box optimization problem into a gray box one by making use of lower fidelity approximations to the target function, i.e., using fewer numbers of epochs or subsets of the data for cheap low-fidelity evaluations that approximate the costly high-fidelity evaluation. Examples include bandit-based budget allocation algorithms such as Successive Halving (Jamieson & Talwalkar, 2016), Hyperband (Li et al., 2018) and their extensions that use non-random search mechanisms (Falkner et al., 2018; Awad et al., 2021; Mallik et al., 2023) or algorithms making use of multi-fidelity information in the context of BO (Swersky et al., 2014; Klein et al., 2017; Wu et al., 2020; Kadra et al., 2023). Several works address the problem of speeding up cross-validation techniques and use techniques that could be described as grey box optimization techniques. Besides the ones mentioned in the main paper (Thornton et al., 2013; Zheng & Bilenko, 2013), it is possible to employ racing techniques for model selection in machine learning as demonstrated by Lang et al. (2015), and there has been a recent interest in methods that adapt the cost of running full cross-validation procedures (Bergman et al., 2024; Buczak et al., 2024).

When addressing the problem of HPO, we must acknowledge an inherent mismatch between the explicit objective we optimize – namely, the estimated generalization performance of a model – and the actual implicit optimization goal, which is to identify a configuration that yields the best

generalization performance on new, unseen data. Typically, evaluations and comparisons of different HPO algorithms focus exclusively on the final best validation performance (i.e., the objective that is directly optimized), even though an unbiased estimate of performance on an external unseen test set might be available. While this approach is logical for assessing the efficacy of an optimization algorithm based on the metric it seeks to improve, relying solely on finding an optimal validation configuration is beneficial only if there is reason to assume a strong correlation between the optimized validation performance and true generalization ability on new, unseen test data. This discrepancy can be found deeply within the HPO community, where the evaluation of HPO algorithms on standard benchmark libraries is usually done solely with respect to the validation performance (Eggenesperger et al., 2021; Pineda Arango et al., 2021; Salinas et al., 2022; Pfisterer et al., 2022).⁵ This relationship between validation performance (i.e., the estimated generalization error derived from resampling) and true generalization performance (e.g., assessed through an outer holdout test set or additional resampling) of an optimal validation configuration found during HPO remains a largely unexplored area of research.

In general, little research has focused on the selection of resampling types, let alone the automated selection of resampling types (Guyon et al., 2010; Feurer et al., 2022). While we usually expect that a more intensive resampling will reduce the variance of the estimated generalization error and thereby improve the (rank) correlation between optimized validation and unbiased outer test performance within HPCs, this benefit is naturally offset by a higher computational expense. Overall, there is little research on which resampling method to use in practice for model selection, and we only know of a study for support vector machines (Wainer & Cawley, 2017), a simulation study for clinical prediction models (Dunias et al., 2024), a study on feature selection (Molinario et al., 2005) and a study on fast CV (Bergman et al., 2024). In addition, ML-Plan (Mohr et al., 2018) proposed a two-stage procedure. In a first stage (search), the tool uses planning on hierarchical task networks to find promising machine learning pipelines on 70% of the training data. In a second step (selection), it uses 100% of the training data and retrains the most promising candidates from the search step. Finally, it uses a combination of the internal generalization error estimation that was used during search and the 0.75 percentile of the generalization error estimation from the selection step to make a more unbiased selection of the final model. The paper found that this improves performance over using only regular cross-validation for search and selection. The general consensus, that is in agreement with our findings, is that CV or repeated CV generally leads to better generalization performance. In addition, while there are theoretical works that compare the accuracy of estimating the generalization error of holdout and CV (Blum et al., 1999), our goal is to correctly identify a single solution, which generalizes well, see the excellent survey by Arlot & Celisse (2010) for a discussion on this topic.

Bouthillier et al. (2021) studied the sources of variance in machine learning experiments, and find that the split into training and test data has the largest impact. Consequently, they suggest to reshuffle the data prior to splitting it into the training, which is then used for HPO, and the test set. We followed their suggestion when designing our experiments and draw a new test sample for every replication, see Section 4.1 and Appendix F. This dependence on the exact split was further already discussed in the context of how much the outcome of a statistical test on results of machine learning experiments depended on the exact train-test split (Bouckaert, 2004).

Finally, the first warning against comparing too many hypothesis using cross-validation was raised by Schaffer (1993), and in addition to the works discussed in Section 5 in the main paper, also picked up by Rao et al. (2008); Cawley & Talbot (2010). Moreover, the problem of finding a correct "upper objective" in a bilevel optimization problem has been noted (Guyon et al., 2010, 2015, 2019). Also, in the related field of algorithm configuration the problem has been identified (Eggenesperger et al., 2019).

B.1 Current Treatment of Resamplings in HPO Libraries and Software

In Table 3, we provide a brief summary of how resampling is handled in popular HPO libraries and software.⁶ For each library, we checked whether the core functionality, examples, or tutorials mention

⁵We admit that these benchmark libraries implement efficient benchmarking methods such as surrogate (Eggenesperger et al., 2018; Pfisterer et al., 2022) or tabular benchmarks (Pineda Arango et al., 2021). It would be possible to adapt them to return the test performance, however, changes in the HPO evaluation protocol, such as the one we propose, would not be feasible.

⁶This summary is not exhaustive but reflects the general consensus observed in widely-used software.

the possibility of reshuffling the resampling during HPO or if the resampling is considered fixed. If reshuffling is used in an example, mentioned, or if core functionality uses it, we mark it with a ✓. If it is unclear or inconsistent across examples and core functionality, we mark it with a ?. Otherwise, we use a ✗. Our conclusion is that the concept of reshuffling resampling generally receives little attention.

Table 3: Exemplary Treatment of Resamplings in HPO Libraries and Software

Software	Reshuffled?	Reference(s)
sklearn	✗	GridSearchCV ¹ / RandomizedSearchCV ²
HEBO	✗	sklearn_tuner ³
optuna	?	Inconsistency between examples ^{4,5,6}
bayesian-optimization	✗	sklearn Example ^{7,8}
ax	✗	CNN Example ⁹
spearmint	✗	No official HPO Examples
scikit-optimize	✗	BO for GBT Example ^{7,10}
SMAC3	✗	SVM Example ^{7,11}
dragonfly	✗	Tree Based Ensemble Example ¹²
aws sagemaker	✗	Blog Post ¹³
raytune	?	Inconsistency between examples ^{14,15}
hyperopt(-sklearn)	?	Cost Function Logic ¹⁶

✗: no reshuffling, ? : both reshuffling and no reshuffling or unclear, ✓: reshuffling

- ¹ https://github.com/scikit-learn/scikit-learn/blob/8721245511de2f225ff5f9aa5f5fadce663cd4a3/sklearn/moodel_selection/_search.py#L1263
- ² https://github.com/scikit-learn/scikit-learn/blob/8721245511de2f225ff5f9aa5f5fadce663cd4a3/sklearn/moodel_selection/_search.py#L1644
- ³ https://github.com/huawei-noah/HEBO/blob/b60f41aa862b4c5148e31ab4981890da6441f2b1/HEBO/hebo/sklearn_tuner.py#L73
- ⁴ https://github.com/optuna/optuna-integration/blob/15e6b0ec6d9a0d7f572ad387be8478c56257bef7/optuna_integration/sklearn/sklearn.py#L223 here sklearn's `cross_validate` is used which by default does not reshuffle the resampling https://github.com/scikit-learn/scikit-learn/blob/8721245511de2f225ff5f9aa5f5fadce663cd4a3/sklearn/moodel_selection/_validation.py#L186
- ⁵ https://github.com/optuna/optuna-examples/blob/dd56b9692e6d1f4fa839332ebdcd93fd48c16d8/pytorch/pytorch_sample.py#L79 here, data loaders for train and valid are instantiated within the objective of the trial but the data within the loaders is fixed
- ⁶ https://github.com/optuna/optuna-examples/blob/dd56b9692e6d1f4fa839332ebdcd93fd48c16d8/xgboost/xgboost_sample.py#L22 here, the train validation split is performed within the objective of the trial and no seed is set which results in reshuffling https://github.com/scikit-learn/scikit-learn/blob/8721245511de2f225ff5f9aa5f5fadce663cd4a3/sklearn/model_selection/_split.py#L259
- ⁷ functionality relies on sklearn's `cross_val_score` which by default does not reshuffle the resampling https://github.com/scikit-learn/scikit-learn/blob/8721245511de2f225ff5f9aa5f5fadce663cd4a3/sklearn/model_selection/_validation.py#L631
- ⁸ https://github.com/bayesian-optimization/BayesianOptimization/blob/c7e5c3926944fc6011ae7ace29f7b5ed0f9c983b/examples/sklearn_example.py#L32
- ⁹ https://github.com/facebook/ax/blob/ac44a6661f535d3046954f8f8d8701327f4a53e2/tutorials/tune_cnn_service.ipynb#L39 and https://github.com/facebook/ax/blob/ac44a6661f535d3046954f8f8d8701327f4a53e2/ax/util/tutorials/cnn_utils.py#L154
- ¹⁰ <https://github.com/scikit-optimize/scikit-optimize/blob/a2369dabc332d16d8ff173b12404b03fea472492/examples/hyperparameter-optimization.py#L82C21-L82C36>
- ¹¹ https://github.com/automl/SMAC3/blob/9aaa8e94a5b3a965773a87b903ee96c683cc42c/examples/1_basics/2_svm_cv.py#L63
- ¹² https://github.com/dragonfly/dragonfly/blob/3eef7d30bc2e56f2221a624bd8ec7f933f81e40/examples/tree_regression/skltree.py#L111
- ¹³ <https://aws.amazon.com/blogs/architecture/field-notes-build-a-cross-validation-machine-learning-model-pipeline-at-scale-with-amazon-sagemaker/>
- ¹⁴ <https://github.com/ray-project/ray/blob/3f5aa5c4642e6b12447d9de5dce22085512312f3/doc/source/tune/examples/tune-pytorch-cifar.ipynb#L120> here, data loaders for train and valid are instantiated within the objective but the data within the loaders are fixed
- ¹⁵ <https://github.com/ray-project/ray/blob/3f5aa5c4642e6b12447d9de5dce22085512312f3/doc/source/tune/examples/tune-xgboost.ipynb#L35> here, the train validation split is performed within the objective and no seed is set which results in reshuffling https://github.com/scikit-learn/scikit-learn/blob/8721245511de2f225ff5f9aa5f5fadce663cd4a3/sklearn/model_selection/_split.py#L259
- ¹⁶ https://github.com/hyperopt/hyperopt-sklearn/blob/4bc286479677a0bf2178dac4546ea268b3f3b77/hpsklearn/estimator/_cost_fn.py#L144 dependence on random seed which by default is not set and there is no discussion of reshuffling and behavior is somewhat unclear

C Proofs of the Main Results

C.1 Proof of Theorem 2.1

We impose *stability* assumptions on the learning algorithm similar to Bayle et al. (2020); Austern & Zhou (2020). Let $\mathbf{Z}, \mathbf{Z}_1, \dots, \mathbf{Z}_n, \mathbf{Z}'_1$, be *iid* random variables. Define $\mathcal{T} = \{\mathbf{Z}_i\}_{i=1}^n$, and \mathcal{T}' as \mathcal{T} but with \mathbf{Z}_n replaced by the independent copy \mathbf{Z}'_n . Define

$$\tilde{\ell}_n(\mathbf{z}, \boldsymbol{\lambda}) = \ell(\mathbf{z}, g_{\boldsymbol{\lambda}}(\mathcal{T})) - \mathbb{E}[\ell(\mathbf{Z}, g_{\boldsymbol{\lambda}}(\mathcal{T})) \mid \mathcal{T}],$$

assume that each $g_{\boldsymbol{\lambda}}(\mathcal{T})$ is invariant to the ordering in \mathcal{T} , ℓ is bounded, and

$$\max_{\boldsymbol{\lambda} \in \Lambda} \mathbb{E}\{[\tilde{\ell}(\mathbf{Z}, g_{\boldsymbol{\lambda}}(\mathcal{T})) - \tilde{\ell}(\mathbf{Z}, g_{\boldsymbol{\lambda}}(\mathcal{T}'))]^2\} = o(1/n). \quad (4)$$

This *loss stability* assumption is rather mild, see Bayle et al. (2020) for an extensive discussion. Further, define the risk $R(g) = \mathbb{E}[\ell(\mathbf{Z}, g)]$ and assume that for every $\boldsymbol{\lambda} \in \Lambda$, there is a prediction rule $g_{\boldsymbol{\lambda}}^*$ such that

$$\max_{\boldsymbol{\lambda} \in \Lambda} \mathbb{E}[|R(g_{\boldsymbol{\lambda}}(\mathcal{T})) - R(g_{\boldsymbol{\lambda}}^*)|] = o(1/\sqrt{n}). \quad (5)$$

This assumption requires $g_{\boldsymbol{\lambda}}(\mathcal{T})$ to converge to some fixed prediction rule sufficiently fast and serves as a reasonable working condition for our purposes. It is satisfied, for example, when ℓ is the square loss and $g_{\boldsymbol{\lambda}}$ is an empirical risk minimizer over a hypothesis class $\mathcal{G}_{\boldsymbol{\lambda}}$ with finite VC-dimension. For further examples, see, e.g., Bousquet & Zhivotovskiy (2021), van Erven et al. (2015), and references therein. The assumption could be relaxed, but this would lead to a more complicated limiting distribution but with the same essential interpretation.

Theorem C.1. *Under assumptions (4) and (5), it holds*

$$\sqrt{n}(\hat{\mu}(\boldsymbol{\lambda}_j) - \mu(\boldsymbol{\lambda}_j))_{j=1}^J \rightarrow_d \mathcal{N}(0, \Sigma),$$

where

$$\begin{aligned} \Sigma_{j,j'} &= \tau_{i,j,M} \lim_{n \rightarrow \infty} \text{Cov}[\bar{\ell}_n(\mathbf{Z}, \boldsymbol{\lambda}_j), \bar{\ell}_n(\mathbf{Z}, \boldsymbol{\lambda}_{j'})], \\ \tau_{j,j',M} &= \lim_{n \rightarrow \infty} \frac{1}{nM^2\alpha^2} \sum_{i=1}^n \sum_{m=1}^M \sum_{m'=1}^M \Pr(i \in \mathcal{I}_{m,j} \cap \mathcal{I}_{m',j'}). \end{aligned}$$

Proof. Define

$$\tilde{\mu}(\boldsymbol{\lambda}_j) = \frac{1}{M} \sum_{m=1}^M \mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j})) \mid \mathcal{T}_{m,j}].$$

By the triangle inequality (first and second step), Jensen's inequality (third step), and (5) (last step),

$$\begin{aligned} & \mathbb{E}[|\tilde{\mu}(\boldsymbol{\lambda}_j) - \mu(\boldsymbol{\lambda}_j)|] \\ & \leq \max_{1 \leq m \leq M} \mathbb{E}[|\mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j})) \mid \mathcal{T}_{m,j}] - \mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j}))]|] \\ & \leq \max_{1 \leq m \leq M} \mathbb{E}[|\mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j})) \mid \mathcal{T}_{m,j}] - \mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}^*(\mathcal{T}_{m,j}))]|] \\ & \quad + \max_{1 \leq m \leq M} \mathbb{E}[|\mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j}))] - \mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}^*(\mathcal{T}_{m,j}))]|] \\ & \leq 2 \max_{1 \leq m \leq M} \mathbb{E}[|\mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j})) \mid \mathcal{T}_{m,j}] - \mathbb{E}[L(\mathcal{V}_{m,j}, g_{\boldsymbol{\lambda}_j}^*(\mathcal{T}_{m,j}))]|] \\ & = 2 \max_{1 \leq m \leq M} \mathbb{E}[|R(g_{\boldsymbol{\lambda}_j}(\mathcal{T}_{m,j})) - R(g_{\boldsymbol{\lambda}_j}^*)|] \\ & = o(1/\sqrt{n}). \end{aligned}$$

Next, assumption (4) together with Theorem 2 and Proposition 3 of Bayle et al. (2020) yield

$$\sqrt{n}(\hat{\mu}(\boldsymbol{\lambda}_j) - \tilde{\mu}(\boldsymbol{\lambda}_j)) - \frac{1}{M} \sum_{m=1}^M \frac{1}{\alpha\sqrt{n}} \sum_{i \in \mathcal{I}_{m,j}} \bar{\ell}_n(\mathbf{Z}_i, \boldsymbol{\lambda}_j) \rightarrow_p 0.$$

Now rewrite

$$\frac{1}{M\alpha\sqrt{n}} \sum_{m=1}^M \sum_{i \in \mathcal{I}_{m,j}} \bar{\ell}_n(\mathbf{Z}_i, \boldsymbol{\lambda}_j) = \frac{1}{M\alpha\sqrt{n}} \sum_{i=1}^n \underbrace{\sum_{m=1}^M \mathbb{1}(i \in \mathcal{I}_{m,j})}_{:=\xi_{i,n}^{(j)}} \bar{\ell}_n(\mathbf{Z}_i, \boldsymbol{\lambda}_j).$$

The sequence $(\xi_{i,n}^{(j)})_{i=1}^n = (\xi_{i,n}^{(j)}, \dots, \xi_{i,n}^{(j)})_{i=1}^n$ is a triangular array of independent, centered, and bounded random vectors. Because $\mathbb{1}(\mathbf{Z}_i \in \mathcal{V}_{m,j})$ and \mathbf{Z}_i are independent, it holds

$$\text{Cov}(\xi_{i,n}^{(j)}, \xi_{i,n}^{(j')}) = \sum_{m=1}^M \sum_{m'=1}^M \mathbb{E}[\mathbb{1}(i \in \mathcal{I}_{m,j} \cap \mathcal{I}_{m',j'})] \mathbb{E}[\bar{\ell}_n(\mathbf{Z}_i, \boldsymbol{\lambda}_j) \bar{\ell}_n(\mathbf{Z}_i, \boldsymbol{\lambda}_{j'})],$$

so

$$\lim_{n \rightarrow \infty} \text{Cov} \left[\frac{1}{M\alpha\sqrt{n}} \sum_{i=1}^n \xi_{i,n}^{(j)}, \frac{1}{M\alpha\sqrt{n}} \sum_{i=1}^n \xi_{i,n}^{(j')} \right] = \lim_{n \rightarrow \infty} \frac{1}{nM^2\alpha^2} \sum_{i=1}^n \text{Cov} [\xi_{i,n}^{(j)}, \xi_{i,n}^{(j')}] = \Sigma_{j,j'}.$$

Now the result follows from Lindeberg's central limit theorem for triangular arrays (e.g., van der Vaart, 2000, Proposition 2.27). \square

C.2 Proof of Theorem 2.2

We want to bound the probability that $\mu(\hat{\boldsymbol{\lambda}}) - \mu(\boldsymbol{\lambda}^*)$ is large. For some $\delta > 0$, define the set of 'good' hyperparameters

$$\Lambda_\delta = \{\boldsymbol{\lambda}_j : \mu(\boldsymbol{\lambda}_j) - \mu(\boldsymbol{\lambda}^*) \leq \delta\}.$$

Now

$$\begin{aligned} \Pr(\mu(\hat{\boldsymbol{\lambda}}) - \mu(\boldsymbol{\lambda}^*) > \delta) &= \Pr(\hat{\boldsymbol{\lambda}} \notin \Lambda_\delta) \\ &= \Pr\left(\min_{\boldsymbol{\lambda} \notin \Lambda_\delta} \hat{\mu}(\boldsymbol{\lambda}) < \min_{\boldsymbol{\lambda} \in \Lambda_\delta} \hat{\mu}(\boldsymbol{\lambda})\right) \\ &\leq \Pr\left(\min_{\boldsymbol{\lambda} \notin \Lambda_\delta} \hat{\mu}(\boldsymbol{\lambda}) < \min_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \hat{\mu}(\boldsymbol{\lambda})\right) \\ &= \Pr\left(\min_{\boldsymbol{\lambda} \notin \Lambda_\delta} \mu(\boldsymbol{\lambda}) + \epsilon(\boldsymbol{\lambda}) < \min_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \mu(\boldsymbol{\lambda}) + \epsilon(\boldsymbol{\lambda})\right) \\ &\leq \Pr\left(\delta + \min_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda}) < \delta/2 + \min_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \epsilon(\boldsymbol{\lambda})\right) \\ &= \Pr\left(\min_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda}) - \min_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \epsilon(\boldsymbol{\lambda}) < -\delta/2\right) \\ &= \Pr\left(\max_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda}) - \max_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \epsilon(\boldsymbol{\lambda}) > \delta/2\right). \quad (\epsilon \stackrel{d}{=} -\epsilon) \end{aligned}$$

There is a tension between the two maxima. The more $\boldsymbol{\lambda}$'s there are in $\Lambda_{\delta/2}$ and the less they are correlated, the more likely it is to find one $\epsilon(\boldsymbol{\lambda})$ that is large. This makes the probability small. However, the less ϵ is correlated, the larger is $\max_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda})$, making the probability large. To formalize this, use the Gaussian concentration inequality (Talagrand, 2005, Lemma 2.1.3):

$$\begin{aligned} &\Pr\left(\max_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda}) - \max_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \epsilon(\boldsymbol{\lambda}) > \delta/2\right) \\ &\leq \Pr\left(2 \left| \max_{\boldsymbol{\lambda} \in \Lambda} \epsilon(\boldsymbol{\lambda}) - \mathbb{E} \left[\max_{\boldsymbol{\lambda} \in \Lambda} \epsilon(\boldsymbol{\lambda}) \right] \right| > \delta/2 - \mathbb{E} \left[\max_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \epsilon(\boldsymbol{\lambda}) \right] + \mathbb{E} \left[\max_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda}) \right]\right) \\ &\leq 2 \exp \left\{ - \frac{(\delta/2 - \mathbb{E} [\max_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \epsilon(\boldsymbol{\lambda})] + \mathbb{E} [\max_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda})])^2}{8\sigma^2} \right\}, \end{aligned}$$

provided $\delta/2 - \mathbb{E} [\max_{\boldsymbol{\lambda} \in \Lambda_{\delta/2}} \epsilon(\boldsymbol{\lambda})] + \mathbb{E} [\max_{\boldsymbol{\lambda} \notin \Lambda_\delta} \epsilon(\boldsymbol{\lambda})] \geq 0$. We bound the two maxima separately.

Lower Bound for Maximum over the Good Set

Recall the definition of m right before Theorem 2.2 and observe

$$\begin{aligned}\Lambda_{\delta/2} &= \{\boldsymbol{\lambda}: \mu(\boldsymbol{\lambda}) - \mu(\boldsymbol{\lambda}^*) \leq \delta/2\} \supset \{\boldsymbol{\lambda}: m\|\boldsymbol{\lambda} - \boldsymbol{\lambda}^*\|^2 \leq \delta/2\} = \{\boldsymbol{\lambda}: \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^*\| \leq (\delta/2m)^{1/2}\} \\ &= B(\boldsymbol{\lambda}^*, (\delta/2m)^{1/2}).\end{aligned}$$

Pack the ball $B(\boldsymbol{\lambda}^*, (\delta/2m)^{1/2})$ with smaller balls with radius η . We can always construct such a packing with at least $(\delta/2m\eta^2)^{d/2}$ elements. By assumption, each small ball contains at least one element of Λ . Pick one element from each small ball and collect them into the set $\Lambda'_{\delta/2}$. By construction, $|\Lambda'_{\delta/2}| \geq (\delta/2m\eta^2)^{d/2}$ and

$$\min_{\boldsymbol{\lambda} \neq \boldsymbol{\lambda}' \in \Lambda'_{\delta/2}} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\| \geq \eta.$$

Sudakov's minoration principle (e.g., Wainwright, 2019, Theorem 5.30) gives

$$\begin{aligned}\mathbb{E} \left[\max_{\boldsymbol{\lambda} \in \Lambda'_{\delta/2}} \epsilon(\boldsymbol{\lambda}) \right] &\geq \frac{1}{2} \sqrt{\log |\Lambda'_{\delta/2}|} \min_{\{\boldsymbol{\lambda} \neq \boldsymbol{\lambda}'\} \cap \Lambda'_{\delta/2}} \sqrt{\text{Var} [\epsilon(\boldsymbol{\lambda}) - \epsilon(\boldsymbol{\lambda}')] } \\ &\geq \frac{1}{2} \sqrt{\log |\Lambda'_{\delta/2}|} \min_{\|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\| \geq \eta} \sqrt{\text{Var} [\epsilon(\boldsymbol{\lambda}) - \epsilon(\boldsymbol{\lambda}')] }.\end{aligned}$$

In general,

$$\begin{aligned}\text{Var} [\epsilon(\boldsymbol{\lambda}) - \epsilon(\boldsymbol{\lambda}')] &= K(\boldsymbol{\lambda}, \boldsymbol{\lambda}) + K(\boldsymbol{\lambda}', \boldsymbol{\lambda}') - 2\tau^2 K(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \\ &= (1 - \tau^2)[K(\boldsymbol{\lambda}, \boldsymbol{\lambda}) + K(\boldsymbol{\lambda}', \boldsymbol{\lambda}')] + \tau^2[K(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - K(\boldsymbol{\lambda}, \boldsymbol{\lambda}')] + \tau^2[K(\boldsymbol{\lambda}', \boldsymbol{\lambda}') - K(\boldsymbol{\lambda}, \boldsymbol{\lambda}')] \\ &\geq 2\sigma^2(1 - \tau^2).\end{aligned}$$

Hence, we have

$$\min_{\|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\| \geq \eta} \text{Var} [\epsilon(\boldsymbol{\lambda}) - \epsilon(\boldsymbol{\lambda}')] \geq 2\sigma^2(1 - \tau^2),$$

which implies

$$\mathbb{E} \left[\max_{\boldsymbol{\lambda} \in \Lambda'_{\delta/2}} \epsilon(\boldsymbol{\lambda}) \right] \geq \frac{1}{2} \sigma \sqrt{d} \sqrt{1 - \tau^2} \sqrt{\log(\delta/2m\eta^2)} =: \sigma \sqrt{d} A(\tau, \delta)/2.$$

Upper Bound for Maximum over the Bad Set

Dudley's entropy bound (e.g., Giné & Nickl, 2016, Theorem 2.3.6) gives

$$\mathbb{E} \left[\max_{\boldsymbol{\lambda} \notin \Lambda_s} \epsilon(\boldsymbol{\lambda}) \right] \leq 12 \int_0^\infty \sqrt{\log N(s)} ds,$$

where $N(s)$ is the minimum number of points $\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_{N(s)}$ such that

$$\sup_{\boldsymbol{\lambda} \in \Lambda} \min_{1 \leq k \leq N(s)} \sqrt{\text{Var} [\epsilon(\boldsymbol{\lambda}) - \epsilon(\boldsymbol{\lambda}_k)]} \leq s.$$

Note that

$$\sup_{\boldsymbol{\lambda}, \boldsymbol{\lambda}' \in \Lambda} \sqrt{\text{Var} [\epsilon(\boldsymbol{\lambda}) - \epsilon(\boldsymbol{\lambda}')] } \leq 2\sigma,$$

so $N(s) = 1$ for all $s \geq 2\sigma$. For $s^2 \leq 4\sigma^2(1 - \tau^2)$, we can use the trivial bound $N(s) \leq J$. For $s^2 > 4\sigma^2(1 - \tau^2)$, cover Λ with ℓ_2 -balls of size $(s/2\sigma\tau\kappa)$. We can do this with less than $N(s) \leq (6\sigma\kappa/s)^d \vee 1$ such balls. Let $\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_N$ be the centers of these balls. In general, it holds

$$\begin{aligned}\text{Var} [\epsilon(\boldsymbol{\lambda}) - \epsilon(\boldsymbol{\lambda}')] &= K(\boldsymbol{\lambda}, \boldsymbol{\lambda}) + K(\boldsymbol{\lambda}', \boldsymbol{\lambda}') - 2\tau^2 K(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \\ &= (1 - \tau^2)[K(\boldsymbol{\lambda}, \boldsymbol{\lambda}) + K(\boldsymbol{\lambda}', \boldsymbol{\lambda}')] + \tau^2[K(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - K(\boldsymbol{\lambda}, \boldsymbol{\lambda}')] + \tau^2[K(\boldsymbol{\lambda}', \boldsymbol{\lambda}') - K(\boldsymbol{\lambda}, \boldsymbol{\lambda}')] \\ &\leq 2(1 - \tau^2)\sigma^2 + 2\tau^2\sigma^2\kappa^2\|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\|^2.\end{aligned}$$

For $s^2 > 4\sigma^2(1 - \tau^2)$, we thus have

$$\begin{aligned} \sup_{\lambda \in \Lambda} \min_{1 \leq k \leq N(s)} \text{Var} [\epsilon(\lambda) - \epsilon(\lambda_k)] &\leq \sup_{\|\lambda - \lambda'\|_2 \leq (s/2\tau\sigma\kappa)^2} \text{Var} [\epsilon(\lambda) - \epsilon(\lambda')] \\ &\leq 2(1 - \tau^2)\sigma^2 + 2\tau^2\sigma^2\kappa^2(s/2\tau\sigma\kappa)^2 \\ &\leq s^2, \end{aligned}$$

as desired. Now decompose the integral

$$\begin{aligned} \int_0^\infty \sqrt{\log N(s)} ds &= \int_0^{2\sigma\sqrt{1-\tau^2}} \sqrt{\log N(s)} ds + \int_{2\sigma\sqrt{1-\tau^2}}^{2\sigma} \sqrt{\log N(s)} ds \\ &\leq 2\sigma\sqrt{d}\sqrt{1-\tau^2}\sqrt{\log J} + \int_{2\sigma\sqrt{1-\tau^2}}^{2\sigma} \sqrt{\log N(s)} ds. \end{aligned}$$

For the second term, compute

$$\begin{aligned} \int_{2\sigma\sqrt{1-\tau^2}}^{2\sigma} \sqrt{\log N(s)} ds &\leq \sqrt{d} \int_{2\sigma\sqrt{1-\tau^2}}^{2\sigma} \sqrt{\log(6\sigma\kappa/s)_+} ds \\ &= \sigma\sqrt{d} \int_{2\sqrt{1-\tau^2}}^2 \sqrt{\log(6\kappa/s)_+} ds \\ &\leq \sigma\sqrt{d} \left(\int_0^2 \log(6\kappa/s)_+ ds \right)^{1/2} \left(2(1 - \sqrt{1 - \tau^2}) \right)^{1/2} \\ &= \sigma\sqrt{d}\sqrt{2 + 2\log(3\kappa)_+} \left(2(1 - \sqrt{1 - \tau^2}) \right)^{1/2} \\ &= 2\sigma\sqrt{d}\sqrt{1 + \log(3\kappa)_+} \frac{\tau}{(1 + \sqrt{1 - \tau^2})^{1/2}} \\ &\leq 2\sigma\sqrt{d}\tau\sqrt{1 + \log(3\kappa)_+}. \end{aligned}$$

We have shown that

$$\mathbb{E} \left[\max_{\lambda \notin \Lambda_s} \epsilon(\lambda) \right] \leq 24\sigma\sqrt{d} \left[\sqrt{1 - \tau^2}\sqrt{\log J} + \tau\sqrt{1 + \log(3\kappa)_+} \right] =: \sigma\sqrt{d}B(\tau)/4.$$

Integrating Probabilities

Summarizing the two previous steps, we have

$$\Pr \left(\mu(\widehat{\lambda}) - \mu(\lambda^*) > \delta \right) \leq 2 \exp \left\{ - \frac{\left(\delta - \sigma\sqrt{d}[B(\tau) - A(\tau, \delta)] \right)^2}{36\sigma^2} \right\},$$

provided $t \geq \sigma\sqrt{d}[B(\tau) - A(\tau, \delta)]$. Now for any $s \geq 0$ and $t \geq 2e^{s^2}m\eta^2$, it holds

$$A(\tau, s) \geq (\underline{\sigma}/\sigma)\sqrt{1 - \tau^2}s =: A(\tau)s.$$

In particular, if

$$t \geq 2e^{s^2}m\eta^2 + \sigma\sqrt{d}[B(\tau) - A(\tau)s] =: C,$$

we have

$$\Pr \left(\mu(\widehat{\lambda}) - \mu(\lambda^*) > \delta \right) \leq 4 \exp \left\{ - \frac{\left(\delta - \sigma\sqrt{d}[B(\tau) - A(\tau)s] \right)^2}{36\sigma^2} \right\}.$$

Integrating the probability gives

$$\begin{aligned}
\mathbb{E}[\mu(\widehat{\boldsymbol{\lambda}}) - \mu(\boldsymbol{\lambda}^*)] &= \int_0^\infty \Pr(\mu(\widehat{\boldsymbol{\lambda}}) - \mu(\boldsymbol{\lambda}^*) > \delta) d\delta \\
&= \int_0^C \Pr(\mu(\widehat{\boldsymbol{\lambda}}) - \mu(\boldsymbol{\lambda}^*) > \delta) d\delta + \int_C^\infty \Pr(\mu(\widehat{\boldsymbol{\lambda}}) - \mu(\boldsymbol{\lambda}^*) > \delta) d\delta \\
&\leq C + \int_C^\infty \exp\left\{-\frac{(\delta - \sigma\sqrt{d}[B(\tau) - A(\tau)s])^2}{36\sigma^2}\right\} d\delta \\
&\leq C + \sqrt{36}\sigma \\
&= 2e^{s^2}m\eta^2 + \sigma\sqrt{d}[B(\tau) - A(\tau)s] + 6\sigma.
\end{aligned}$$

Simplifying

The bound can be optimized with respect to s , but the solution involves the Lambert W -function, which has no analytical expression. Instead choose s for simplicity as

$$s = \sqrt{\log\left(\frac{\sigma}{2m\eta^2}\right)_+}$$

which gives

$$\mathbb{E}[\mu(\widehat{\boldsymbol{\lambda}}) - \mu(\boldsymbol{\lambda}^*)] \leq \sigma\sqrt{d} \left[8 + B(\tau) - A(\tau)\sqrt{\log\left(\frac{\sigma}{2m\eta^2}\right)} \right]. \quad \square$$

D Additional Results on the Density of Random HPC Grids

Lemma D.1. *Suppose that the J elements in Λ are drawn independently from a continuous density p with $c := \min_{\|\boldsymbol{\lambda}\| \leq 1} p(\boldsymbol{\lambda}) > 0$. Then with probability at least $1 - \delta$,*

$$\eta \lesssim \left(\sqrt{\log(1/\delta)/J}\right)^{1/d},$$

and with probability 1,

$$\eta \lesssim \left(\sqrt{\log(J)/J}\right)^{1/d},$$

for all J sufficiently large.

Proof. We want to bound the probability that there is a $\boldsymbol{\lambda}$ such that $|B(\boldsymbol{\lambda}, \eta) \cap \Lambda| = 0$. In what follows $\boldsymbol{\lambda}$ is silently understood to have norm bounded by 1. Let $\tilde{\boldsymbol{\lambda}}_1, \dots, \tilde{\boldsymbol{\lambda}}_N$ the centers of $\eta/2$ -balls covering $\{\|\boldsymbol{\lambda}\| \leq 1\}$, for which we may assume $N \leq (6/\eta)^d$. For $\tilde{\boldsymbol{\lambda}}_k$ the closest center to $\boldsymbol{\lambda}$, it holds

$$\|\boldsymbol{\lambda}' - \boldsymbol{\lambda}\| \leq \|\boldsymbol{\lambda}' - \tilde{\boldsymbol{\lambda}}_k\| + \|\tilde{\boldsymbol{\lambda}}_k - \boldsymbol{\lambda}\| \leq \|\boldsymbol{\lambda}' - \tilde{\boldsymbol{\lambda}}_k\| + \eta/2,$$

so $\|\boldsymbol{\lambda}' - \tilde{\boldsymbol{\lambda}}_k\| \leq \eta/2$ implies $\|\boldsymbol{\lambda}' - \boldsymbol{\lambda}\| \leq \eta$. We thus have

$$\begin{aligned}
\Pr(\exists \boldsymbol{\lambda}: |B(\boldsymbol{\lambda}, \eta) \cap \Lambda| = 0) &= \Pr\left(\inf_{\boldsymbol{\lambda}} \sum_{i=1}^J \mathbb{1}\{\|\boldsymbol{\lambda}_i - \boldsymbol{\lambda}\| \leq \eta\} \leq 0\right) \\
&\leq \Pr\left(\min_{1 \leq k \leq N} \sum_{i=1}^J \mathbb{1}\{\|\boldsymbol{\lambda}_i - \tilde{\boldsymbol{\lambda}}_k\| \leq \eta/2\} \leq 0\right).
\end{aligned}$$

Further

$$\begin{aligned}
& \Pr \left(\min_{1 \leq k \leq N} \sum_{i=1}^J \mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \leq 0 \right) \\
&= \Pr \left(\max_{1 \leq k \leq N} \sum_{i=1}^J -\mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \geq 0 \right) \\
&\leq \Pr \left(\max_{1 \leq k \leq N} \sum_{i=1}^J \mathbb{E} \left[\mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \right] - \mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \geq J \inf_{\lambda} \mathbb{E} \left[\mathbb{1}\{\|\lambda_i - \lambda\| \leq \eta/2\} \right] \right).
\end{aligned}$$

It holds

$$\begin{aligned}
\mathbb{E} \left[\mathbb{1}\{\|\lambda_i - \lambda\| \leq \eta/2\} \right] &= \Pr(\|\lambda_i - \lambda\| \leq \eta/2) = \int_{\|\lambda' - \lambda\| \leq \eta/2} p(\lambda') d\lambda' \geq c \text{vol}(B(0, \eta/2)) \\
&= cv_d(\eta/2)^d,
\end{aligned}$$

where $v_d = \text{vol}(B(0, 1))$. Now the union bound and Hoeffding's inequality give

$$\begin{aligned}
\Pr \left(\min_{1 \leq k \leq N} \sum_{i=1}^J \mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \leq 0 \right) &\leq N \exp \left(-\frac{Jc^2v_d^2(\eta/2)^{2d}}{2} \right) \\
&\leq (6/\eta)^d \exp \left(-\frac{Jc^2v_d^2(\eta/2)^{2d}}{2} \right).
\end{aligned}$$

Choosing

$$\eta = 2 \left(\sqrt{2 \log(3^d \sqrt{J} cv_d / \delta)} / \sqrt{J} cv_d \right)^{1/d}$$

gives

$$\Pr(\exists \lambda: |B(\lambda, \eta) \cap \Lambda| = 0) \leq \delta / \sqrt{2 \log(3^d \sqrt{J} cv_d)},$$

which is bounded by δ when $\sqrt{J} \geq e^{1/2} / 3^d cv_d$. Further, setting $\eta = 2(\sqrt{6 \log(J)} / \sqrt{J} cv_d)^{1/d}$ gives

$$\Pr \left(\min_{1 \leq k \leq N} \sum_{i=1}^J \mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \leq 0 \right) \lesssim J^{-5/2},$$

so that

$$\begin{aligned}
& \sum_{J=1}^{\infty} \Pr \left(\min_{1 \leq j \leq J} \min_{1 \leq k \leq N} \sum_{i=1}^j \mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \leq 0 \right) \\
&\leq \sum_{J=1}^{\infty} J \Pr \left(\min_{1 \leq k \leq N} \sum_{i=1}^J \mathbb{1}\{\|\lambda_i - \tilde{\lambda}_k\| \leq \eta/2\} \leq 0 \right) \\
&\lesssim \sum_{J=1}^{\infty} \frac{1}{J^{3/2}} < \infty.
\end{aligned}$$

Now the Borel-Cantelli lemma (e.g., Kallenberg, 1997, Theorem 4.18) implies that, with probability 1,

$$|B(\lambda, \eta) \cap \Lambda| \geq 1,$$

for all J sufficiently large. \square

E Selected Validation Schemes

E.1 Definition of Index Sets

Recall:

- (i) (holdout) Let $M = 1$ and $\mathcal{I}_{1,j} = \mathcal{I}_1$ for all $j = 1, \dots, J$, and some size- $\lceil \alpha n \rceil$ index set \mathcal{I}_1 .
- (ii) (reshuffled holdout) Let $M = 1$ and $\mathcal{I}_{1,1}, \dots, \mathcal{I}_{1,J}$ be independently drawn from the uniform distribution over all size- $\lceil \alpha n \rceil$ subsets from $\{1, \dots, n\}$.
- (iii) (M -fold CV) Let $\alpha = 1/M$ and $\mathcal{I}_1, \dots, \mathcal{I}_M$ be a disjoint partition of $\{1, \dots, n\}$, and $\mathcal{I}_{m,j} = \mathcal{I}_m$ for all $j = 1, \dots, J$.
- (iv) (reshuffled M -fold CV) Let $\alpha = 1/M$ and $(\mathcal{I}_{1,j}, \dots, \mathcal{I}_{M,j}), j = 1, \dots, J$, be independently drawn from the uniform distribution over disjoint partitions of $\{1, \dots, n\}$.
- (v) (M -fold holdout) Let $\mathcal{I}_m, m = 1, \dots, M$, be independently drawn from the uniform distribution over size- $\lceil \alpha n \rceil$ subsets of $\{1, \dots, n\}$ and set $\mathcal{I}_{m,j} = \mathcal{I}_m$ for all $m = 1, \dots, M, j = 1, \dots, J$.
- (vi) (reshuffled M -fold holdout) Let $\mathcal{I}_{m,j}, m = 1, \dots, M, j = 1, \dots, J$, be independently drawn from the uniform distribution over size- $\lceil \alpha n \rceil$ subsets of $\{1, \dots, n\}$.

E.2 Derivation of Reshuffling Parameters in Limiting Distribution

Recall

$$\tau_{i,j,M} = \frac{1}{nM^2\alpha^2} \sum_{s=1}^n \sum_{m=1}^M \sum_{m'=1}^M \Pr(s \in \mathcal{I}_{m,i} \cap \mathcal{I}_{m',j}).$$

For all schemes in the proposition, the probabilities are independent of the index s , so the average over $s = 1, \dots, n$ can be omitted. We now verify the constants σ, τ from Table 1.

- (i) It holds

$$\Pr(s \in \mathcal{I}_{1,i} \cap \mathcal{I}_{1,j}) = \Pr(s \in \mathcal{I}_1) = \alpha.$$

Hence,

$$\tau_{i,j,1} = 1/\alpha = 1/\alpha \times 1 = \sigma^2 \times \tau^2.$$

- (ii) (reshuffled holdout) This is a special case of part (vi) with $M = 1$.
- (iii) (M -fold CV) It holds

$$\Pr(s \in \mathcal{I}_{m,i} \cap \mathcal{I}_{m',j}) = \Pr(s \in \mathcal{I}_m \cap \mathcal{I}_{m'}) = \begin{cases} 1/M, & m = m', \\ 0, & m \neq m'. \end{cases}$$

Only M probabilities in the double sum are non-zero, whence

$$\tau_{i,j,M} = \frac{1}{M^2\alpha^2} \times M/M = 1/\alpha^2 M^2 = 1 \times 1 = \sigma^2 \times \tau^2,$$

where we used $\alpha = 1/M$.

- (iv) (reshuffled M -fold CV) It holds

$$\Pr(s \in \mathcal{I}_{m,i} \cap \mathcal{I}_{m',j}) = \begin{cases} 1/M, & m = m', i = j \\ 0, & m \neq m', i = j \\ 1/M^2, & m = m', i \neq j \\ 1/M^2, & m \neq m', i \neq j. \end{cases}$$

For $i = j$, only M probabilities in the double sum are non-zero. Also using $\alpha = 1/M$, we get

$$\tau_{i,j,M} = \frac{1}{M^2\alpha^2} \times M \times 1/M = 1 = \sigma^2.$$

For $i \neq j$,

$$\tau_{i,j,M} = \frac{1}{M^2\alpha^2} \times M^2 \times 1/M^2 = 1 \times 1 = \sigma^2 \times \tau^2.$$

(v) (M -fold holdout) It holds

$$\Pr(s \in \mathcal{I}_{m,i} \cap \mathcal{I}_{m',j}) = \Pr(s \in \mathcal{I}_m \cap \mathcal{I}_{m'}) = \begin{cases} \alpha, & m = m', \\ \alpha^2, & \text{else.} \end{cases}$$

This gives

$$\tau_{i,j,M} = \frac{1}{M^2\alpha^2} \times [M \times \alpha + (M-1)M \times \alpha^2] = [1/\alpha M + (M-1)/M] \times 1 = \sigma^2 \times \tau^2.$$

for all i, j .

(vi) (reshuffled M -fold holdout) It holds

$$\Pr(s \in \mathcal{I}_{m,i} \cap \mathcal{I}_{m',j}) = \begin{cases} \alpha, & m = m', i = j \\ \alpha^2, & \text{else.} \end{cases}$$

For $i = j$, this gives

$$\tau_{i,j,M} = \frac{1}{M^2\alpha^2} \times [M \times \alpha + (M-1)M \times \alpha^2] = 1/\alpha M + (M-1)/M.$$

For $i \neq j$,

$$\tau_{i,j,M} = \frac{1}{M^2\alpha^2} \times (M^2 \times \alpha^2) = 1.$$

This implies that (1) holds with $\sigma^2 = 1/M\alpha + (M-1)/M$, $\tau^2 = 1/(1/M\alpha + (M-1)/M)$.

Remark E.1. Although not technically covered by Theorem 2.1, performing independent bootstraps for each λ_j correspond to reshuffled n -fold holdout with $\alpha = 1/n$. Accordingly, $\sigma \approx \sqrt{2}$ and $\tau \approx \sqrt{1/2}$.

F Details Regarding Benchmark Experiments

F.1 Datasets

We list all datasets used in the benchmark experiments in Table 4.

Table 4: List of datasets used in benchmark experiments. All information can be found on [OpenML](#) (Vanschoren et al., 2014).

OpenML Dataset ID	Dataset Name	Size ($n \times p$)
23517	numerai28.6	96320 \times 21
1169	airlines	539383 \times 7
41147	albert	425240 \times 78
4135	Amazon_employee_access	32769 \times 9
1461	bank-marketing	45211 \times 16
1590	adult	48842 \times 14
41150	MiniBooNE	130064 \times 50
41162	kick	72983 \times 32
42733	Click_prediction_small	39948 \times 11
42742	porto-seguro	595212 \times 57

Note that datasets serve as data generating processes (DGPs; Hothorn et al., 2005). As we are mostly concerned with the actual generalization performance of the final best HPC found during HPO based on validation performance we rely on a comparably large held out test set that is not used during HPO. We therefore use 5000 data points sampled from a DGP as an outer test set. To further be able to measure the generalization performance robustly for varying data sizes available during HPO, we construct concrete tasks based on the DGPs by sampling subsets of (`train_valid`; n) size 500, 1000 and 5000 from the DGPs. This results in 30 tasks in total (10 DGPs \times 3 `train_valid` sizes). For more details and the concrete implementation of this procedure, see Appendix F.3. We also collected another 5000 data points as an external validation set, but did not use it. Therefore, we had to tighten the restriction to 10000 data points mentioned in the main paper to 15000 data points as the lower bound on data points. To allow for stronger variation over different replications, we decided to use 20000 as the final lower bound.

F.2 Learning Algorithms

Here we briefly present training pipeline details and search spaces of the learning algorithms used in our benchmark experiments.

The funnel-shaped MLP is based on sklearn’s MLP Classifier and is constructed in the following way: The hidden layer size for each layer is determined by `num_layers` and `max_units`. We start with `max_units` and half the number of units for every subsequent layer to create a funnel. `max_batch_size` is the largest power of 2 that is smaller than the number of training samples available. We use ReLU as activation function and train the network optimizing logloss as a loss function via SGD using a constant learning rate and Nesterov momentum for 100 epochs. Table 5 lists the search space (inspired from Zimmer et al. (2021)) used during HPO.

The Elastic Net is based on sklearn’s Logistic Regression Classifier. We train it for a maximum of 1000 iterations using the "saga" solver. Table 6 lists the search space used during HPO.

The XGBoost and CatBoost search spaces are listed in Table 7 and Table 8, both inspired from their search spaces used in McElfresh et al. (2023).

For both the Elastic Net and Funnel MLP, missing values are imputed in the preprocessing pipeline (mean imputation for numerical features and adding a new level for categorical features). Categorical features are target encoded in a cross-validated manner using a 5-fold CV. Features are then scaled to zero mean and unit variance via a standard scaler. For XGBoost, we impute missing values for categorical features (adding a new level) and target encode them in a cross-validated manner using a 5-fold CV. For CatBoost, no preprocessing is performed.

XGBoost and CatBoost models are trained for 2000 iterations and stop early if the validation loss (using the default internal loss function used during training, i.e., logloss) does not improve over a horizon of 20 iterations. For retraining the best configuration on the whole train and validation data, the number of boosting iterations is set to the number of iterations used to find the best validation performance prior to the stopping mechanism taking action.⁷

F.3 Exact Implementation

In the following, we outline the exact implementation of performing one HPO run for a given learning algorithm on a concrete task (dataset \times train_valid size) and a given resampling. We release all code to replicate benchmark results and reproduce our analyses via https://github.com/slds-1mu/paper_2024_resuffling. For a given replication (in total 10):

1. We sample (without replacement) `train_valid` size (500, 1000 or 5000 points) and `test` size (always 5000) points from the DGP (i.e. a concrete dataset in Table 4). These are shared for every learning algorithm (i.e. all learning algorithms are evaluated on the same data).
2. A given HPC is evaluated in the following way:
 - The resampling operates on the train validation⁸ set of size `train_valid`.
 - The learning algorithm is configured by the HPC.
 - The learning algorithm is trained on training splits and evaluated on validation splits according to the resampling strategy. In case reshuffling is turned on, the training and validation splits are recreated for every HPO. We compute the Accuracy, ROC AUC and logloss when using a random search and compute ROC AUC when using HEBO or SMAC3 and average performance over all folds for resamplings involving multiple folds.
 - For each HPC we then always re-train the model on all `train_valid` data being available and evaluate the model on the held-out `test` set to compute an outer estimate of generalization performance for each HPC (regardless of whether it is the incumbent for a given iteration or not).

⁷For CV and repeated holdout we take the average number of boosting iterations over the models trained on the different folds.

⁸With train validation we refer to all data being available during HPO which is then further split by a resampling into train and validation sets.

Table 5: Search Space for Funnel-Shaped MLP Classifier.

Parameter	Type	Range	Log
num_layers	Int.	1 to 5	No
max_units	Int.	64, 128, 256, 512	No
learning_rate	Num.	1×10^{-4} to 1×10^{-1}	Yes
batch_size	Int.	16, 32, ..., max_batch_size	No
momentum	Num.	0.1 to 0.99	No
alpha	Num.	1×10^{-6} to 1×10^{-1}	Yes

Table 6: Search Space for Elastic Net Classifier.

Parameter	Type	Range	Log
C	Num.	1×10^{-6} to 1×10^4	Yes
l1_ratio	Num.	0.0 to 1.0	No

Table 7: Search Space for XGBoost Classifier.

Parameter	Type	Range	Log
max_depth	Int.	2 to 12	Yes
alpha	Num.	1×10^{-8} to 1.0	Yes
lambda	Num.	1×10^{-8} to 1.0	Yes
eta	Num.	0.01 to 0.3	Yes

Table 8: Search Space for CatBoost Classifier.

Parameter	Type	Range	Log
learning_rate	Num.	0.01 to 0.3	Yes
depth	Int.	2 to 12	Yes
l2_leaf_reg	Num.	0.5 to 30	Yes

3. We evaluate 500 HPCs when using random search and 250 HPC when using HEBO or SMAC3 (SMAC4HPO facade).

As resamplings, we use holdout with a 80/20 train-validation split and 5 folds for CV, so that the holdout strategy is just one fold of the CV and the fraction of data points being used for training and respectively validation are the same across different resampling strategies. 5-fold holdout simply repeats the holdout procedure five times and 5x 5-fold CV repeats the 5-fold CV five times. Each of the four resamplings can be reshuffled or not (standard).

As mentioned above, the test set is only varied for each of the 10 replica (repetitions with different seeds), but consistent for different tasks (i.e. the different learning algorithms are evaluated on the same test set, similarly, also the different dataset subsets all share the same test set). This allows for fair comparisons of different resamplings on a concrete problem (i.e. a given dataset, train_valid size and learning algorithm). Additionally, for the random search, the 500 HPCs evaluated for a given learning algorithm are also fixed over different dataset and train_valid size combinations. This is done to allow for an isolation of the effect, the concrete resampling (and whether it is reshuffled or not) has on generalization performance, reducing noise arising due to different HPCs. Learning algorithms themselves are not explicitly seeded to allow for variation during model training over different replications. Resamplings and partitioning of data are always performed in a stratified manner with respect to the target variable.

For the random search, we only ran (standard and reshuffled) holdout and (standard and reshuffled) 5x 5-fold CV experiments (because we can simulate 5-fold CV and 5-fold holdout experiments based

on the results obtained from the 5x 5-fold CV (by only considering the first repeat or the first fold for each of the five repeats).⁹

For running HEBO or SMAC3, each resampling (standard and reshuffled for holdout, 5-fold holdout, 5-fold CV, 5x 5-fold CV) has to be actually run due to the adaptive nature of BO.

For the random search experiments, this results in 10 (DGPs) \times 3 (`train_valid` sizes) \times 4 (learning algorithms) \times 2 (holdout or 5x 5-fold CV) \times 2 (standard or reshuffled) \times 10 (replications) = 4800 HPO runs,¹⁰ each involving the evaluation of 500 HPCs and each evaluation of an HPC involving either 2 (for holdout; due to retraining on train validation data) or 26 (for 5x 5-fold CV; due to retraining on train validation data) model fits. In summary, the random search experiments involve the evaluation of 2.4 Million HPCs with in total 33.6 Million model fits.

Similarly, for the HEBO and SMAC3 experiments, this each results in 10 (DGPs) \times 3 (`train_valid` sizes) \times 4 (learning algorithms) \times 4 (holdout, 5-fold CV, 5x 5-fold CV or 5-fold holdout) \times 2 (standard or reshuffled) \times 10 (replications) = 9600 HPO runs¹¹, each involving the evaluation of 250 HPCs and each evaluation of an HPC involving either 2 (for holdout; due to retraining on train validation data), 6 (for 5-fold CV or 5-fold holdout; due to retraining on train validation data) or 26 (for 5x 5-fold CV; due to retraining on train validation data) model fits. In summary, the HEBO and SMAC3 experiments *each* involve the evaluation of 2.4 Million HPCs with in total 24 Million model fits.

F.4 Compute Resources

We estimate our total compute time for the random search, HEBO and SMAC3 experiments to be roughly 11.86 CPU years. Benchmark experiments were run on an internal HPC cluster equipped with a mix of Intel Xeon E5-2670, Intel Xeon E5-2683 and Intel Xeon Gold 6330 instances. Jobs were scheduled to use a single CPU core and were allowed to use up to 16GB RAM. Total emissions are estimated to be an equivalent of roughly 6508.67 kg CO₂.

G Additional Benchmark Results Visualizations

G.1 Main Experiments

In this section, we provide additional visualizations of the results of our benchmark experiments.

Figure 6 illustrates the trade-off between the final number of model fits required by different resamplings and the final average normalized test performance (AUC ROC) after running random search for a budget of 500 hyperparameter configurations. We can see that the reshuffled holdout on average comes close to the final test performance of the overall more expensive 5-fold CV.

Below, we give an overview of the different types of additional analyses and visualizations we provide. Normalized metrics, i.e., normalized validation or test performance refer to the measure being scaled to $[0, 1]$ based on the empirical observed minimum and maximum values obtained on the raw results level (ADTM; see Wistuba et al., 2018). More concretely, for each scenario consisting of a learning algorithm that is run on a given task (dataset \times `train_valid` size) given a certain performance metric, the performance values (validation or test) for all resamplings and optimizers are normalized on the replication level to $[0, 1]$ by subtracting the empirical best value and dividing by the range of performance values. Therefore a normalized performance value of 0 is best and 1 is worst. Note that we additionally provide further aggregated results on the learning algorithm level and raw results of validation and test performance via https://github.com/slds-lmu/paper_2024_resuffling.

- Random search
 - Normalized validation performance in Figure 7.

⁹We even could have simulated the vanilla holdout from the 5x 5-fold CV experiments by choosing an arbitrary fold and repeat but choose not to do so, to have some sanity checks regarding our implementation by being able to compare the "true" holdout with a the simulated holdout.

¹⁰Note that we do not have to take the 3 different metrics into account because random search allows us to simulate runs for different metric post hoc.

¹¹Note that HEBO and SMAC3 were only run for ROC AUC as the performance metric.

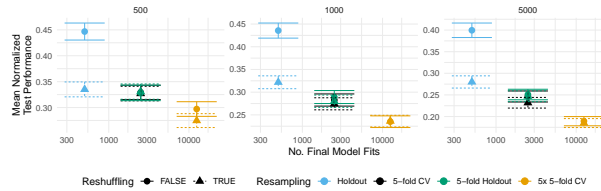


Figure 6: Trade-off between the final number of model fits required by different resamplings and the final average normalized test performance (AUC ROC) after running random search for a budget of 500 hyperparameter configurations. Averaged over different tasks, learning algorithms and replications separately for increasing n (train-validation sizes, columns). Shaded areas represent standard errors.

- Normalized test performance in Figure 8.
- Improvement in test performance over 5-fold CV in Figure 9.
- Rank w.r.t. test performance in Figure 10.
- HEBO and SMAC3 vs. random search holdout
 - Normalized validation performance in Figure 11.
 - Normalized test performance in Figure 12.
 - Improvement in test performance over standard holdout in Figure 13.
 - Rank w.r.t. test performance in Figure 14.
- HEBO and SMAC3 vs. random search 5-fold holdout
 - Normalized validation performance in Figure 15.
 - Normalized test performance in Figure 16.
 - Improvement in test performance over standard 5-fold holdout in Figure 17.
 - Rank w.r.t. test performance in Figure 18.
- HEBO and SMAC3 vs. random search 5-fold CV
 - Normalized validation performance in Figure 19.
 - Normalized test performance in Figure 20.
 - Improvement in test performance over 5-fold CV in Figure 21.
 - Rank w.r.t. test performance in Figure 22.
- HEBO and SMAC3 vs. random search 5x 5-fold CV
 - Normalized validation performance in Figure 23.
 - Normalized test performance in Figure 24.
 - Improvement in test performance over 5x 5-fold CV in Figure 25.
 - Rank w.r.t. test performance in Figure 26.

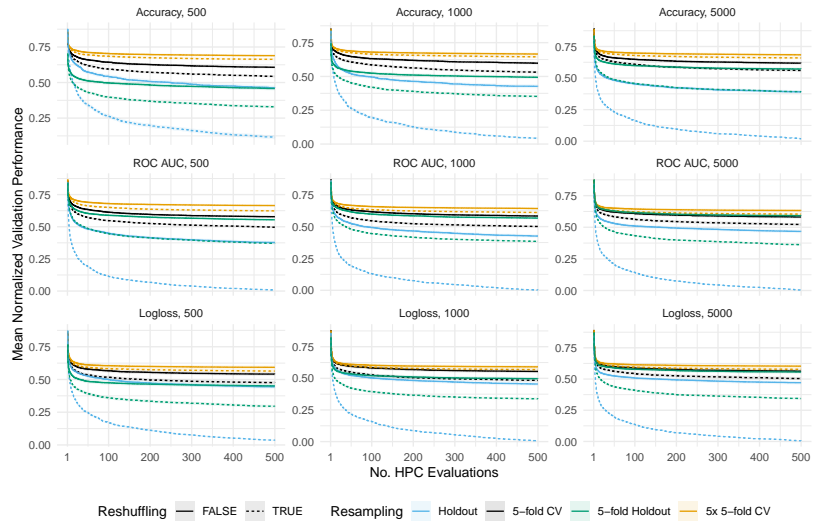


Figure 7: Random search. Average normalized performance over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

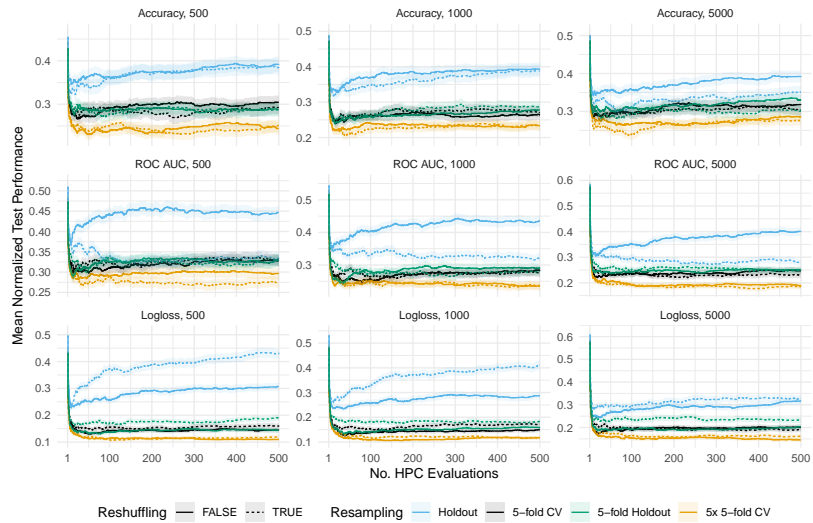


Figure 8: Random search. Average normalized test performance over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

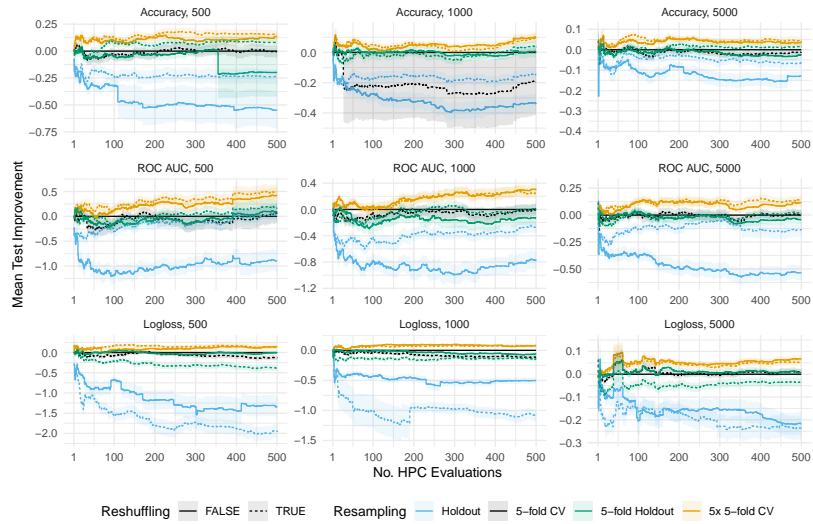


Figure 9: Random search. Average improvement (compared to standard 5-fold CV) with respect to test performance of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

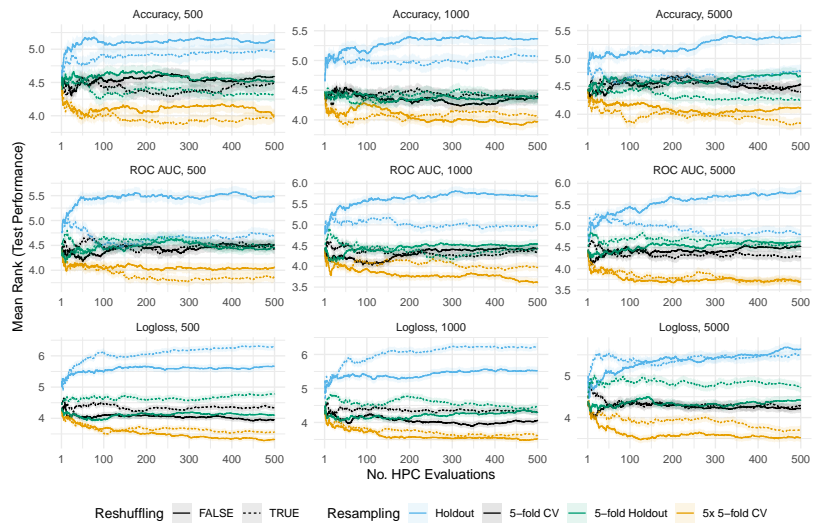


Figure 10: Random search. Average ranks (lower is better) with respect to test performance over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

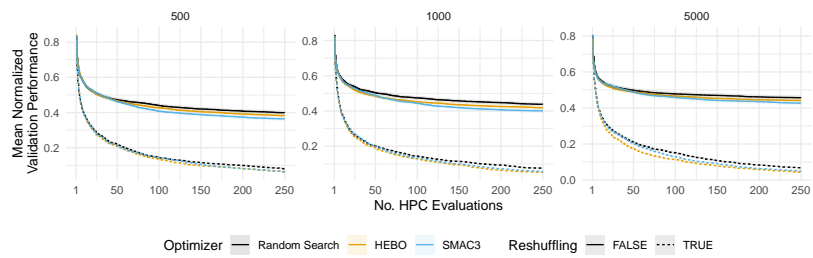


Figure 11: HEBO and SMAC3 vs. random search for holdout. Average normalized validation performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

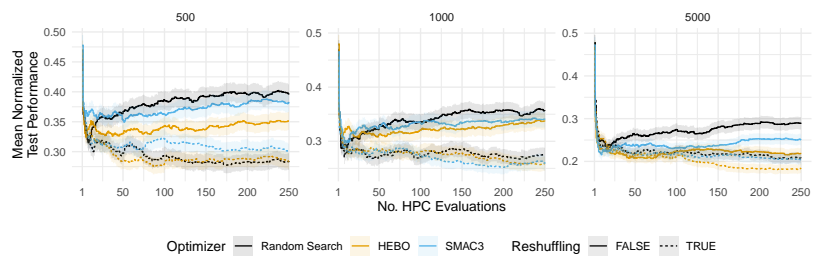


Figure 12: HEBO and SMAC3 vs. random search for holdout. Average normalized test performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

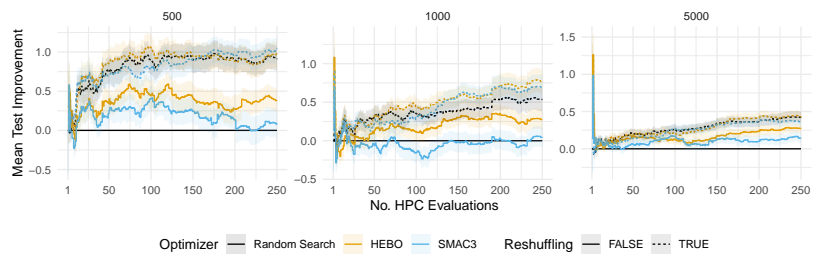


Figure 13: HEBO and SMAC3 vs. random search for holdout. Average improvement (compared to standard holdout) with respect to test performance (ROC AUC) of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

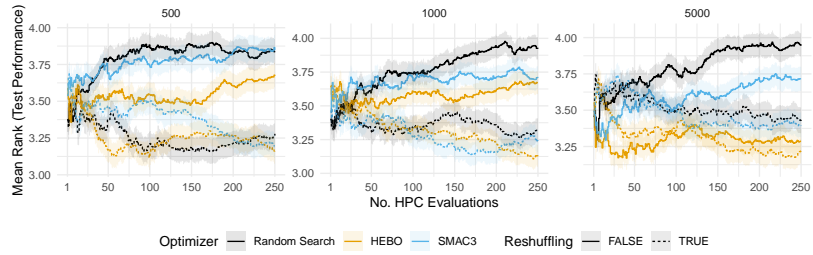


Figure 14: HEBO and SMAC3 vs. random search for holdout. Average ranks (lower is better) with respect to test performance (ROC AUC) of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

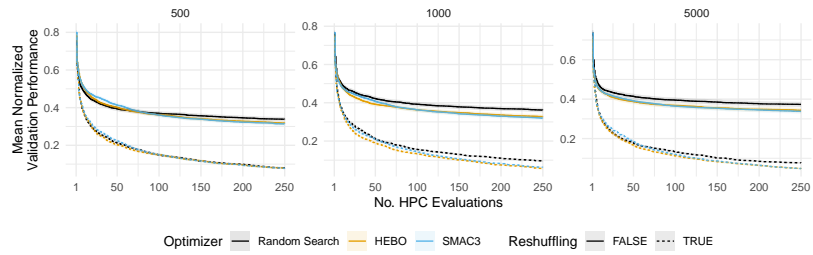


Figure 15: HEBO and SMAC3 vs. random search for 5-fold holdout. Average normalized validation performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

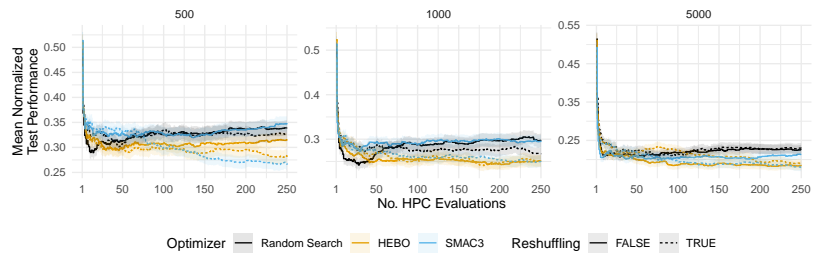


Figure 16: HEBO and SMAC3 vs. random search for 5-fold holdout. Average normalized test performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

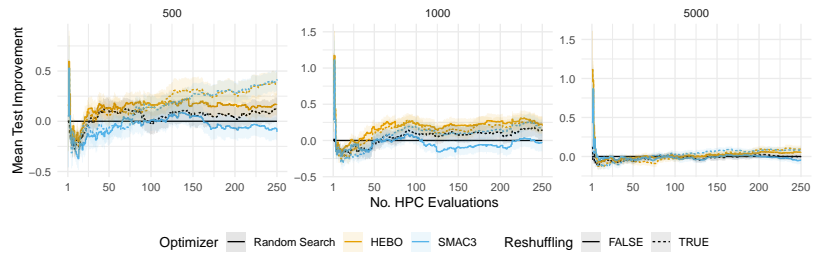


Figure 17: HEBO and SMAC3 vs. random search for 5-fold holdout. Average improvement (compared to standard 5-fold holdout) with respect to test performance (ROC AUC) of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

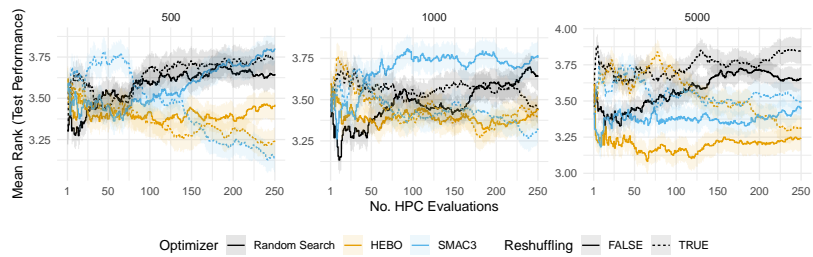


Figure 18: HEBO and SMAC3 vs. random search for 5-fold holdout. Average ranks (lower is better) with respect to test performance (ROC AUC) of the incumbent tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

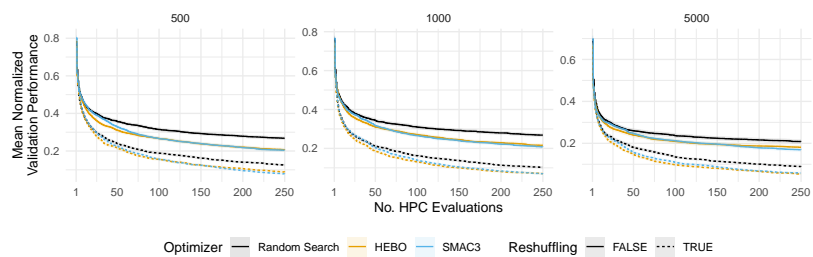


Figure 19: HEBO and SMAC3 vs. random search for 5-fold CV. Average normalized validation performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

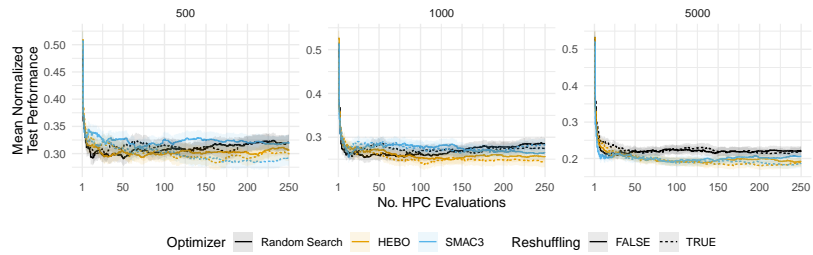


Figure 20: HEBO and SMAC3 vs. random search for 5-fold CV. Average normalized test performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

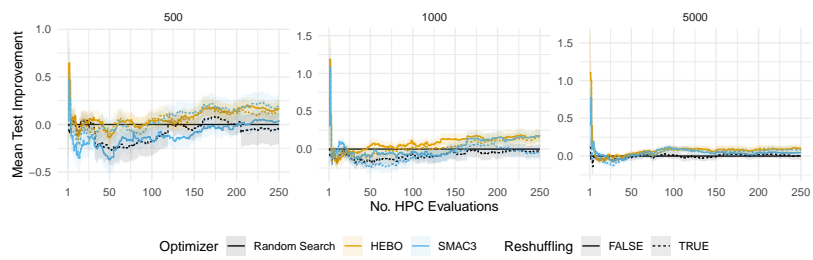


Figure 21: HEBO and SMAC3 vs. random search for 5-fold CV. Average improvement (compared to standard 5-fold CV) with respect to test performance (ROC AUC) of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

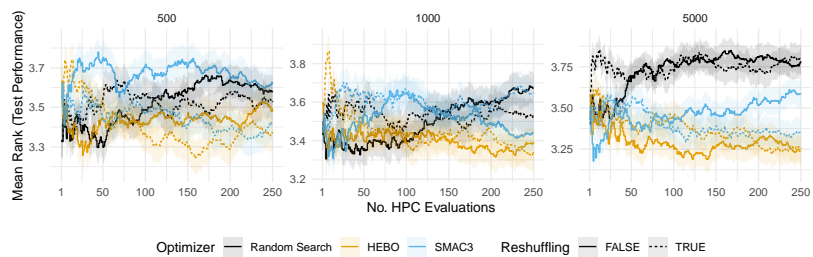


Figure 22: HEBO and SMAC3 vs. random search for 5-fold CV. Average ranks (lower is better) with respect to test performance (ROC AUC) of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

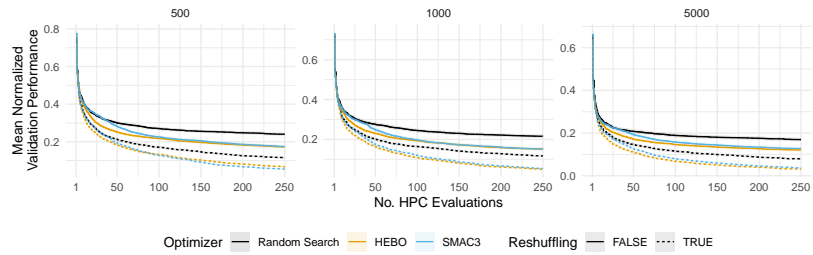


Figure 23: HEBO and SMAC3 vs. random search for 5x 5-fold CV. Average normalized validation performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

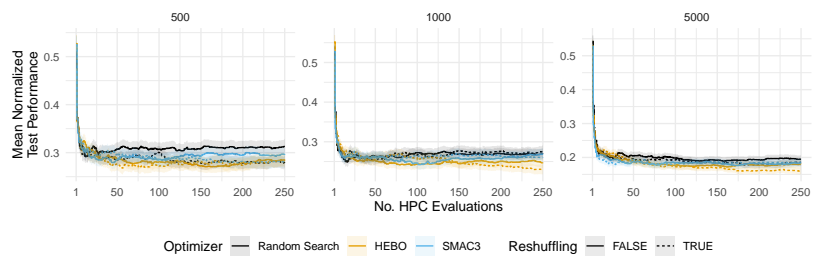


Figure 24: HEBO and SMAC3 vs. random search for 5x 5-fold CV. Average normalized test performance (ROC AUC) over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

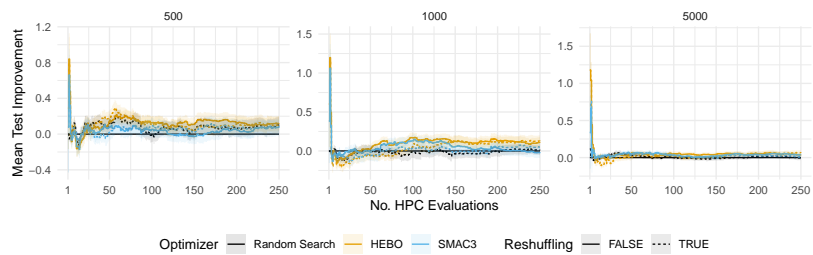


Figure 25: HEBO and SMAC3 vs. random search for 5x 5-fold CV. Average improvement (compared to standard 5x 5-fold CV) with respect to test performance (ROC AUC) of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

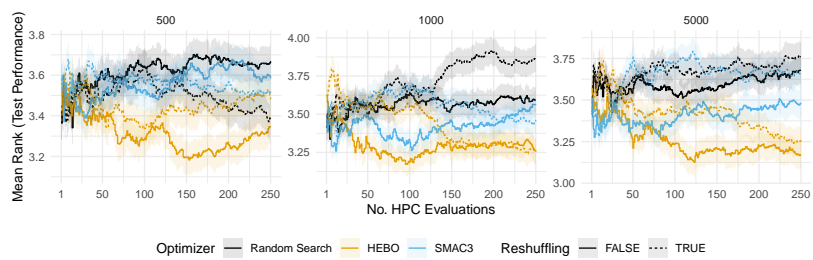


Figure 26: HEBO and SMAC3 vs. random search for 5x 5-fold CV. Average ranks (lower is better) with respect to test performance (ROC AUC) of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

G.2 Ablation on M-fold holdout

Based on the 5x 5-fold CV results we further simulated different M -fold holdout resamplings (standard and reshuffled) by taking M repeats from the first fold of the 5x 5-fold CV. This allows us to get an understanding of the effect more folds have on M -fold holdout, especially in the context of reshuffling.

Regarding normalized validation performance we observe that more folds generally result in a less optimistically biased validation performance (see Figure 27). Looking at normalized test performance (Figure 28) we observe the general trend that more folds result in better test performance – which is expected. Reshuffling generally results in better test performance compared to the standard resampling (with the exception of logloss where especially in the case of a single holdout, reshuffling can hurt generalization performance). This effect is smaller, the more folds are used, which is in line with our theoretical results presented in Table 1. Looking at improvement compared to standard 5-fold holdout with respect to test performance and ranks with respect to test performance, we observe that often reshuffled 2-fold holdout results that are highly competitive with standard 3, 4 or 5-fold holdout.

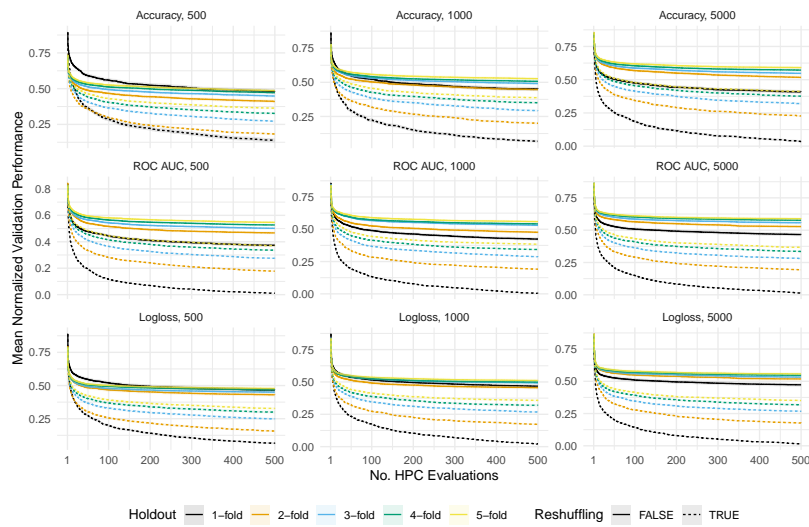


Figure 27: Random search. Average normalized validation performance over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

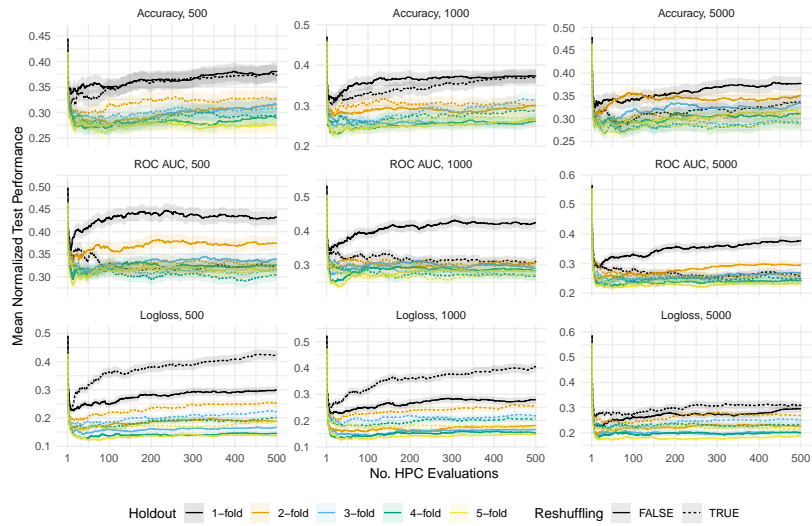


Figure 28: Random search. Average normalized test performance over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

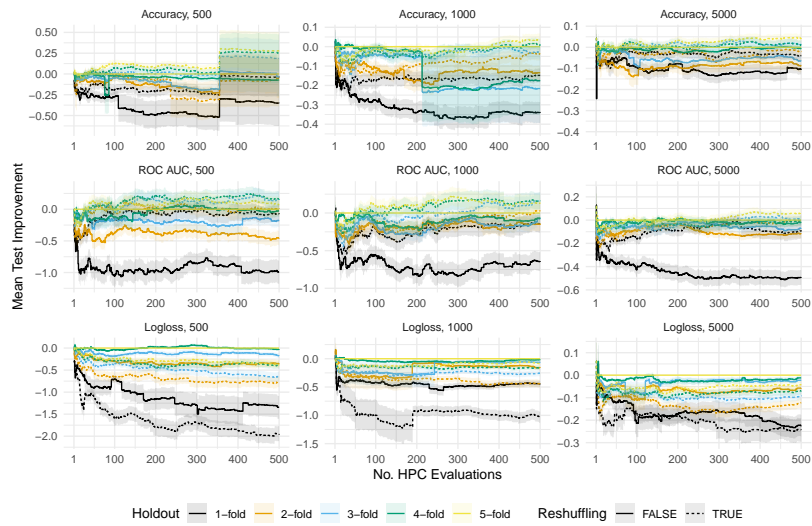


Figure 29: Random search. Average improvement (compared to standard 5-fold holdout) with respect to test performance of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

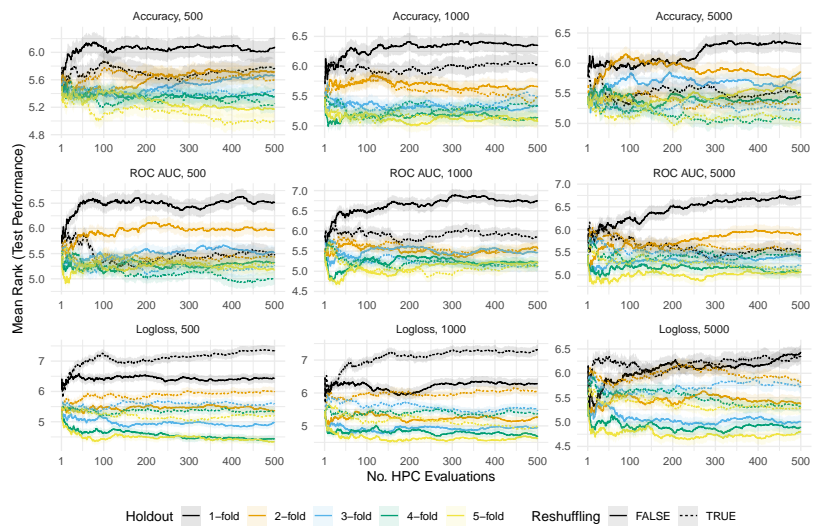


Figure 30: Random search. Average ranks (lower is better) with respect to test performance of the incumbent over tasks, learners and replications for different n (train-validation sizes, columns). Shaded areas represent standard errors.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: We outline our three main contributions in the introduction (Section 1). We do not discuss generalization in the introduction, but rather in the discussion in Section 5.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: The paper provides an analysis of reshuffling data in the context of estimating the generalization error for hyperparameter optimization. Our theoretical analysis explains why reshuffling works, and we experimentally verify the theoretical analysis. We discuss the limitations of our work in Section 5.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory Assumptions and Proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Full assumptions and proofs for our main results (Theorem 2.1 and Theorem 2.2) are given in Appendix C.1 and Appendix C.2, respectively. Derivations for the parameters in Table 1 are provided in Appendix E. The additional results for the grid density are stated and proven directly in Appendix D.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental Result Reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide thorough details on the experimental setup in Section 4.1 and Appendix F. Moreover, we provide code to reproduce our results under an open source license at https://github.com/slds-lmu/paper_2024_resuffling.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

- (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Regarding datasets, we rely on [OpenML.org](https://openml.org). We provide thorough details on the experimental setup in Section 4.1 and Appendix F. Moreover, we provide code to reproduce our results under an open source license at https://github.com/slds-lmu/paper_2024_resuffling.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental Setting/Details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We provide thorough details on the experimental setup in Section 4.1 and Appendix F. Moreover, we provide code to reproduce our results under an open source license at https://github.com/slds-lmu/paper_2024_resuffling.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment Statistical Significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report the standard error in every analysis.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments Compute Resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We provide details in Appendix F.4.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code Of Ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: Our work provides a study on reshuffling data when estimating the generalization error in hyperparameter tuning. Therefore, our work is applicable wherever standard machine learning is applicable, and we do not see any ethical concerns in our method.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader Impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: The paper conducts fundamental research that is not tied to particular applications, let alone deployment.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The paper conducts fundamental research that is not tied to particular applications, let alone deployment. The paper does not develop models that have a high risk for misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We used datasets from [OpenML.org](https://openml.org) and reference the dataset pages. Further information of the datasets, including their licenses, are available at [OpenML.org](https://openml.org).

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New Assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide code as a new asset and describe how we make our code available in Point 5 of the NeurIPS Paper Checklist.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and Research with Human Subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does neither involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does neither involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

3.2. HPO x ELA: Investigating Hyperparameter Optimization Landscapes by Means of Exploratory Landscape Analysis

Contributing article:

L. Schneider*, L. Schäpermeier*, R. P. Prager*, B. Bischl, H. Trautmann, and P. Kerschke. HPO × ELA: Investigating hyperparameter optimization landscapes by means of exploratory landscape analysis. In G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, editors, *Parallel Problem Solving from Nature - PPSN XVII*, pages 575–589, 2022. https://link.springer.com/chapter/10.1007/978-3-031-14714-2_40.

Copyright information:

This article is licensed under the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).

Author contributions:







Lennart Schneider, Lennart Schäpermeier, and Raphael Patrick Prager share the first authorship. Their overall contributions can be described as follows. The conceptualization was a collaborative effort involving all authors, with the main ideas contributed by Lennart Schneider, Lennart Schäpermeier, and Raphael Patrick Prager. The definition of research questions was jointly performed by Lennart Schneider, Lennart Schäpermeier, and Raphael Patrick Prager under the supervision of Bernd Bischl, Heike Trautmann, and Pascal Kerschke. The methodological framework, which involved contrasting optimizer performance on BBOB versus HPO problems as well as ELA feature space analysis and cluster analysis, was designed by Lennart Schneider, Lennart Schäpermeier, Raphael Patrick Prager, and Heike Trautmann, and was carried out by Lennart Schneider. The code for benchmark experiments was collaboratively developed by Lennart Schneider, Lennart Schäpermeier, and Raphael Patrick Prager. Data collection and benchmark experiments were performed by Lennart Schneider, who also conducted the formal analysis, analyzed all results, and created all visualizations presented in the manuscript. The initial draft of the manuscript was jointly authored by Lennart Schneider and Lennart Schäpermeier, with all authors contributing to the revision process.

Supplementary material available at:

- Code, data and online appendix: https://github.com/slds-lmu/hpo_ela
- arXiv version: <https://arxiv.org/abs/2208.00220>



HPO × ELA: Investigating Hyperparameter Optimization Landscapes by Means of Exploratory Landscape Analysis

Lennart Schneider¹ , Lennart Schäpermeier² , Raphael Patrick Prager³ ,
Bernd Bischl¹ , Heike Trautmann^{3,4} , and Pascal Kerschke² 

¹ Chair of Statistical Learning and Data Science, LMU Munich, Munich, Germany

{lennart.schneider,bernd.bischl}@stat.uni-muenchen.de

² Big Data Analytics in Transportation, TU Dresden, Dresden, Germany

{lennart.schaepermeier,pascal.kerschke}@tu-dresden.de

³ Data Science: Statistics and Optimization, University of Münster,
Münster, Germany

{raphael.prager,heike.trautmann}@wi.uni-muenster.de

⁴ Data Management and Biometrics Group, University of Twente,
Enschede, Netherlands

Abstract. Hyperparameter optimization (HPO) is a key component of machine learning models for achieving peak predictive performance. While numerous methods and algorithms for HPO have been proposed over the last years, little progress has been made in illuminating and examining the actual structure of these black-box optimization problems. Exploratory landscape analysis (ELA) subsumes a set of techniques that can be used to gain knowledge about properties of unknown optimization problems. In this paper, we evaluate the performance of five different black-box optimizers on 30 HPO problems, which consist of two-, three- and five-dimensional continuous search spaces of the XGBoost learner trained on 10 different data sets. This is contrasted with the performance of the same optimizers evaluated on 360 problem instances from the black-box optimization benchmark (BBOB). We then compute ELA features on the HPO and BBOB problems and examine similarities and differences. A cluster analysis of the HPO and BBOB problems in ELA feature space allows us to identify how the HPO problems compare to the BBOB problems on a structural meta-level. We identify a subset of BBOB problems that are close to the HPO problems in ELA feature space and show that optimizer performance is comparably similar on these two sets of benchmark problems. We highlight open challenges of ELA for HPO and discuss potential directions of future research and applications.

Keywords: Hyperparameter optimization · Exploratory landscape analysis · Machine learning · Black-box optimization · Benchmarking

L. Schneider, L. Schäpermeier and R. P. Prager—Equal contributions.

© The Author(s) 2022

G. Rudolph et al. (Eds.): PPSN 2022, LNCS 13398, pp. 575–589, 2022.

https://doi.org/10.1007/978-3-031-14714-2_40

1 Introduction

In machine learning (ML), hyperparameter optimization (HPO) constitutes one of the most frequently used tools for improving the predictive performance of a model [3]. The goal of classical single-objective HPO is to find a hyperparameter configuration that minimizes the estimated generalization error. Generally, neither a closed-form mathematical representation nor analytic gradient information is available, making HPO a *black-box* optimization problem and evolutionary algorithms (EAs) and model-based optimizers good candidate algorithms. As a consequence, no prior information about the optimization landscape – which could allow comparisons of HPO and other black-box problems, or provide guidance regarding the choice of optimizer – is available. This also extends to automated ML (AutoML) [14], which builds upon HPO.

In contrast, in the domain of *continuous* black-box optimization, a sophisticated toolbox for landscape analysis and the characterization of their properties has been developed over the years. In exploratory landscape analysis (ELA), optimization landscape features are calculated from small samples of evaluated points from the original black-box problem. It has been shown in numerous studies that ELA feature sets capture relevant landscape characteristics and that they can be used for automated algorithm selection, improving upon the state-of-the-art selector [5, 17]. Particularly well-studied are the functions from the black-box optimization benchmark (BBOB) [12].

Empirical studies [30, 31] in the closely related area of algorithm configuration hint that performance landscapes often are rather benign, i.e., unimodal and convex, although this only holds for an aggregation over larger instance sets and their analysis does not allow further characterization of individual problem landscapes. There exists some work to circumvent HPO altogether, by automatically configuring an algorithm for a given problem instance [1, 28]. However, these are limited to configuring optimization algorithms rather than ML models. In addition, they are often restricted in the number and type of variables they are able to configure. [26] apply fitness landscape analysis on AutoML landscapes, computing fitness distance correlations and neutrality ratios on various AutoML problems. They utilize these features only in an exploratory manner, characterizing the landscapes, without a link to optimizer performance, and cannot compare the analyzed landscapes to other black-box problems in a natural way. Similar work on fitness landscape analysis exists but focuses mostly on neural networks [6, 35]. Some preliminary work [9] on the hyperparameters of a $(1 + 1)$ -EA on a OneMax problem suggests that the ELA feature distribution of a HPO problem can be significantly different from other benchmark problems. Recently, [32] developed statistical tests for the deviation of loss landscapes from uni-modality and convexity and showed that loss landscapes of AutoML problems are highly structured and often uni-modal.

In this work, we characterize continuous HPO problems using ELA features, enabling comparisons between different black-box optimization problems and optimizers. Our main contributions are as follows:

1. We examine similarities and differences of HPO and BBOB problems by investigating the performance of different black-box optimizers.
2. We compute ELA features for all HPO and BBOB problems and demonstrate their usefulness in distinguishing between HPO and BBOB.
3. We demonstrate how HPO problems position themselves in ELA feature space on a meta-level by performing a cluster analysis on principle components derived from ELA features of HPO and BBOB problems and investigate performance differences of optimizers on HPO problems and BBOB problems that are close to the HPO problems in ELA feature space.
4. We discuss how ELA can be used for HPO in future work and highlight open challenges of ELA in the context of HPO.
5. We release code and data of all our benchmark experiments hoping to facilitate future research (which currently may be hindered due to the computationally expensive HPO black-box evaluations).

The remainder of this paper is structured as follows: Fundamentals for HPO and ELA are introduced in Sect. 2. The experimental setup is presented in Sect. 3, with the results regarding the algorithm performance and ELA feature space analysis in Sect. 4 and 5, respectively. Section 6 concludes this paper and offers future research directions.

2 Background

Hyperparameter Optimization. Hyperparameter optimization (HPO) methods aim to identify a well-performing hyperparameter configuration $\lambda \in \tilde{\Lambda}$ for an ML algorithm \mathcal{I}_λ [3]. An ML *learner* or *inducer* \mathcal{I} configured by hyperparameters $\lambda \in \Lambda$ maps a data set $\mathcal{D} \in \mathbb{D}$ to a model \hat{f} , i.e., $\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}$, $(\mathcal{D}, \lambda) \mapsto \hat{f}$. \mathcal{H} denotes the so-called hypothesis space, i.e., the function space to which a model belongs [3]. The considered search space $\tilde{\Lambda} \subset \Lambda$ is typically a subspace of the set of all possible hyperparameter configurations: $\tilde{\Lambda} = \tilde{\Lambda}_1 \times \tilde{\Lambda}_2 \times \dots \times \tilde{\Lambda}_d$, where $\tilde{\Lambda}_i$ is a bounded subset of the domain of the i -th hyperparameter Λ_i . This $\tilde{\Lambda}_i$ can be either real, integer, or category valued, and the search space can contain dependent hyperparameters, leading to a possibly hierarchical search space. The classical (single-objective) HPO problem is defined as:

$$\lambda^* \in \arg \min_{\lambda \in \tilde{\Lambda}} \widehat{\text{GE}}(\lambda), \quad (1)$$

i.e., the goal is to minimize the estimated generalization error. This typically involves a costly resampling procedure that can take a significant amount of time, see [3] for further details. $\widehat{\text{GE}}(\lambda)$ is a black-box function, as it generally has no closed-form mathematical representation, and analytic gradient information is generally not available. Therefore, the minimization of $\widehat{\text{GE}}(\lambda)$ forms an *expensive black-box* optimization problem. In general, $\widehat{\text{GE}}(\lambda)$ is only a stochastic estimate of the true unknown generalization error. Formally, $\widehat{\text{GE}}(\lambda)$ depends on

the concrete inducer, a resampling strategy (e.g., cross-validation) and a performance metric, for more details see [3]. In the following, we use the logloss as performance metric:

$$\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \left(- \sum_{k=1}^g \sigma_k(y^{(i)}) \log(\hat{\pi}_k(\mathbf{x}^{(i)})) \right). \quad (2)$$

Here, g is the total number of classes, $\sigma_k(y^{(i)})$ is 1 if y is class k , and 0 otherwise (multi-class one-hot encoding), and $\hat{\pi}_k(\mathbf{x}^{(i)})$ is the estimated probability for observation $\mathbf{x}^{(i)}$ belonging to class k .

Exploratory Landscape Analysis. The optimization landscapes of black-box functions, by design, carry no prior problem information, beyond the definition of their search parameters, which can be used for their characterization. In the continuous domain, ELA [23] addresses this problem by computing features on a small sample of evaluated points, which can be used for better understanding optimizer performance [24], algorithm selection [17] and even algorithm configuration [28].

The original ELA features consist, e.g., of meta model features (`ela.meta`) such as adjusted R^2 values for quadratic and linear models and y -distribution features (`ela.distr`) such as the skewness and kurtosis of the objective values. Over time, researchers continued to propose further feature sets, including nearest better clustering (`nbc`) [16] and dispersion (`disp`) [22] features to measure multi-modality, and information content (`ic`) features [25], which extract features from random walks across the problem landscape. The R package `flacco` [18] and Python package `pflacco` [27] implement a collection of the most widely used ELA feature sets.

ELA studies often focus on the noiseless BBOB functions, as they offer diverse, well-understood challenges (such as conditioning and multimodality) and a wide range of algorithm performance data is readily available. BBOB consists of 24 minimization problems, which are identified by their function ID (FID) and scalable with respect to their dimensionality, which ranges from 2 to 40. Furthermore, different instances, identified by instance IDs (IIDs), are defined for each function, creating slightly different optimization problems with the same fundamental characteristics by means of randomized transformations in the decision and objective space. All D -dimensional BBOB problems share a decision space of $[-5, 5]^D$, which is guaranteed to contain the (known) optimum.

3 Experimental Setup

We compare the following optimizers: `CMAES` (a simple CMA-ES with $\sigma_0 = 0.5$ and no restarts), `GENSA` (a generalized simulated annealing approach as described in [37]), `Grid` (a grid search performed by generating a uniform sized grid over the search space and evaluating configurations of the grid in random order),

Random (random search performed by sampling configurations uniformly at random), and **MBO** (Bayesian optimization using a Gaussian process as surrogate model and expected improvement as acquisition function [15], similarly configured as in [20]). All optimizers were given a budget of $50D$ function evaluations in total (where D is the dimensionality of the problem). All optimizer runs were replicated 10 times. We choose these optimizers for the following reasons: (1) they cover a wide range of optimizers that can be used for a black-box problem, (2) **Grid** and especially **Random** are frequently used for HPO and **Random** often can be considered a strong baseline [2].

As HPO problems, we tune XGBoost¹ [8] on ten different OpenML [36] data sets (classification tasks) chosen from the OpenML-CC18 benchmarking suite [4]. The specific data sets were chosen to cover a variety of the number of classes, instances, and features (cf. Table 1). To reduce noise as much as possible, performance (logloss) is estimated via 10-fold cross-validation with a fixed instantiating per data set. On each data set, we create 2, 3 and 5 dimensional XGBoost problems by tuning **nrounds**, **eta** ($2D$), **lambda** ($3D$), **gamma** and **alpha** ($5D$), resulting in 30 problems in total. We selected these hyperparameters because (1) they can be incorporated in a purely continuous search space which is generally required for the computation of ELA features, (2) they have been shown to be influential on performance [29] and (3) have a straightforward interpretation, i.e., **nrounds** controls the number of boosting iterations (typically increasing performance but also the tendency to overfit) while the other hyperparameters counteract overfitting and control various aspects of regularization. The full search space is described in Table 2. Note that **nrounds** is tuned on a logarithmic scale and therefore all parameters are treated as continuous during optimization. Missing values of numeric features were imputed using Histogram imputation (values are drawn uniformly at random between lower and upper histogram breakpoints with cells being sampled according to the relative frequency of points contained in a cell). Missing values of factor variables were imputed by adding a new factor level and factor variables were encoded using one-hot-encoding. While XGBoost is a practically relevant learner we do have to note that only considering a single learner is somewhat restrictive. We discuss this limitation in Sect. 6. In the following, individual HPO problems are abbreviated by `<name>_<d>`, i.e., `wilt_2` for the $2D$ wilt problem.

As BBOB problems we select FIDs 1–24 with IIDs 1–5 with a dimensionality of $\{2, 3, 5\}$, resulting in 360 problems in total. We abbreviate individual BBOB problems by `<fid>_<iid>_<dim>`, i.e., `24_1_5` for FID 24 with IID 1 in the $5D$ setting. Experiments have been conducted in R [33], where the individual implementation of an optimizer is referenced in the `mlr3` ecosystem [19]. The package `smoof` [7] provides the aforementioned BBOB problems. We release all data and code for running the benchmarks and analyzing results via the following GitHub repository: <https://github.com/slds-lmu/hpo.ela>. HPO benchmarks took around 2.2 CPU years on Intel Xeon E5-2670 instances, with optimizer overhead ranging from 10% (MBO for $5D$) to less than 1% (**Random** or **Grid**).

¹ Using a `gbtree` booster.

Table 1. OpenML data sets.

ID	Name	Number of		
		Cl.	Inst.	Feat.
40983	wilt	2	4839	5
469	anacatdata_dmft	6	797	4
41156	ada	2	4147	48
6332	cylinder-bands	2	540	37
23381	dresses-sales	2	500	12
1590	adult	2	48842	14
1461	Bank-marketing	2	45211	16
40975	car	4	1728	6
41146	sylvine	2	5124	20
40685	shuttle	7	58000	9

IDs correspond to OpenML data set IDs, which enable to query data set properties via <https://www.openml.org/d/<id>>.

Table 2. XGBoost search space.

Hyper-param.	Type	Range	Trafo
nrounds	int.	[3, 2000]	log
eta	cont.	[exp(-7), exp(0)]	log
lambda	cont.	[exp(-7), exp(7)]	log
gamma	cont.	[exp(-10), exp(2)]	log
alpha	cont.	[exp(-7), exp(7)]	log

“log” in the Trafo column indicates that this parameter is optimized on a (continuous) logarithmic scale, i.e., the range is given by $[\log(\text{lower}), \log(\text{upper})]$, and values are re-transformed via the exponential function prior to their evaluation. Parameters part of the full XGBoost search space that are not shown are set to their default.

4 Optimizer Performance

For each BBOB problem, we computed optimizer rankings based on the average final performance (best target value of an optimizer run averaged over replications). Figures 1a to 1c visualize the differences in rankings on the BBOB problems split for the dimensionality. Friedman tests indicated overall significant differences in rankings ($2D$: $\chi^2(4) = 154.55, p < 0.001$, $3D$: $\chi^2(4) = 219.16, p < 0.001$, $5D$: $\chi^2(4) = 258.69, p < 0.001$). We observe that MBO and CMAES perform well throughout all three dimensionalities, whereas GENSA only is significantly better than Grid or Random for dimensionalities 3 and 5. Moreover, Grid only falls behind Random for the $5D$ problems.

Figures 1d to 1f analogously visualize differences in rankings on the HPO problems split for the dimensionality. Friedman tests indicated overall significant differences in rankings ($2D$: $\chi^2(4) = 36.32, p < 0.001$, $3D$: $\chi^2(4) = 34.32, p < 0.001$, $5D$: $\chi^2(4) = 34.80, p < 0.001$). Again, MBO and CMAES perform well throughout all three dimensionalities. Notably, GENSA shows lacklustre performance regardless of the dimensionality, failing to outperform Grid or Random. Similarly as on the BBOB problems, Grid tends to fall behind Random for the higher-dimensional problems. We do want to note that critical difference plots for the HPO problems are somewhat underpowered when compared to the BBOB problems due to the difference in the number of benchmark problem which results in larger critical distances, as seen in the figures.

In Fig. 2, we visualize the anytime performance of optimizers by the mean normalized regret averaged over replications split for the dimensionality of problems. The normalized regret is defined for an optimizer trace on a benchmark problem as the distance of the current best solution to the overall best solution found across all optimizers and replications, scaled by the overall range of empirical solution values for this benchmark problem. We choose this metric due to the theoretical optimal solutions being unknown for HPO problems, and apply it to both BBOB and HPO problems to enable performance comparisons. We

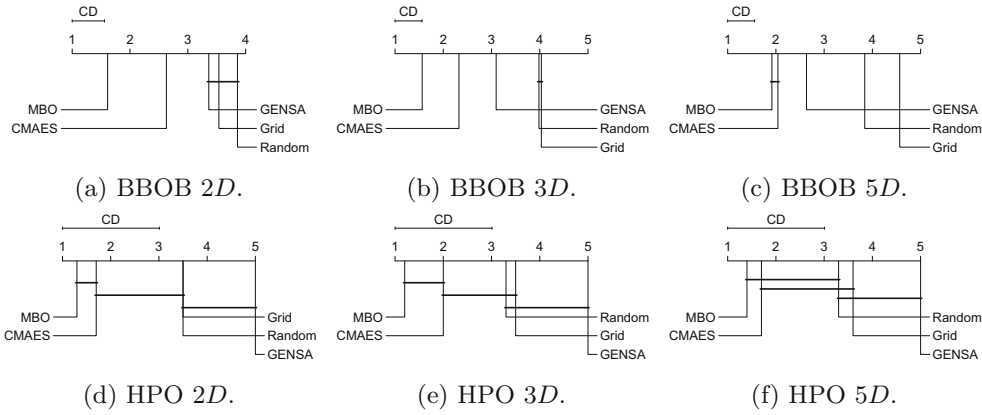


Fig. 1. Critical differences plots for mean ranks of optimizers on BBOB and HPO problems split with respect to the dimensionality.

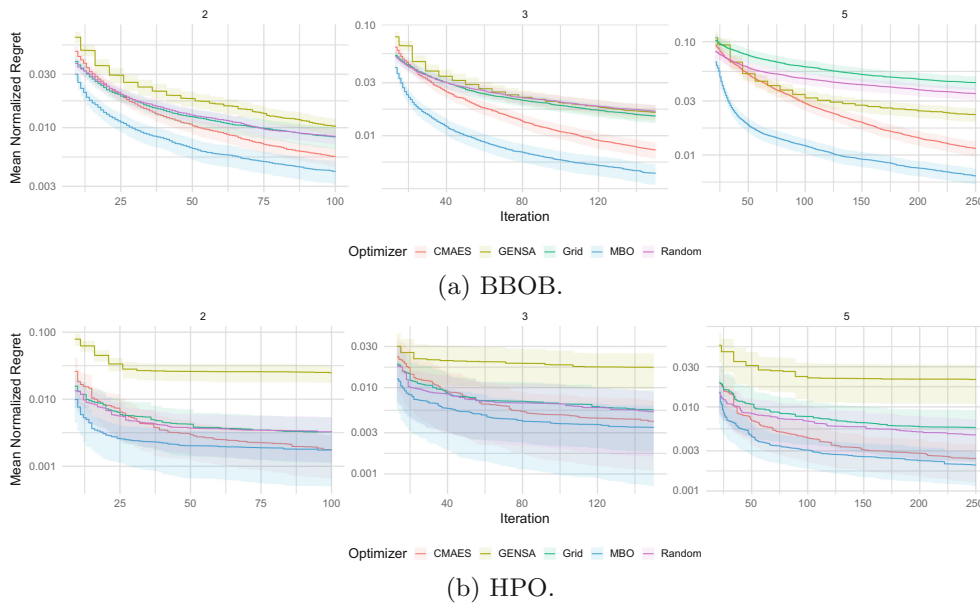


Fig. 2. Anytime mean normalized regret of optimizers on BBOB and HPO problems averaged over replications split for the dimensionality of problems. Ribbons represent standard errors. The x-axis starts after 8% of the optimization budget has been used (initial MBO design).

observe strong anytime performance of MBO and CMAES on both BBOB and HPO problems regardless their dimensionality. GENSA shows good performance on the 5D BBOB problems but shows poor anytime performance on HPO problems in general. Differences in anytime performance are less pronounced on the HPO problems, although we do want to note that the width of the standard error ribbons is strongly influenced by the number of benchmark problems.

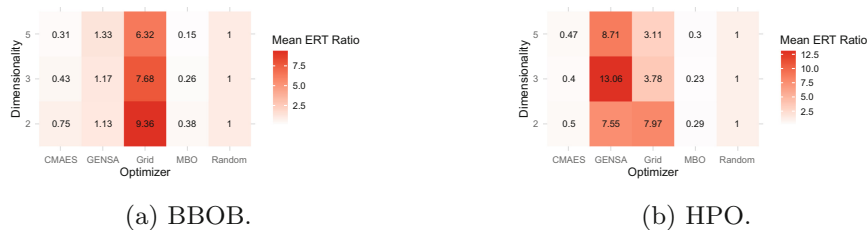


Fig. 3. Average ERT ratios (optimizers to **Random**) for HPO and BBOB problems.

As an additional performance evaluation, we calculated the Expected Running Time (ERT) [11]. In essence, for a given algorithm and problem, the ERT is defined as $ERT = \frac{1}{n} \sum_{i=1}^{10} FE_i$, where n is the number of repetitions which are able to reach a specific target, i refers to an individual repetition, and FE_i denotes the number of function evaluations used. We investigated the ERT of optimizers with the target given as the median of the best **Random** solutions (using $50D$ evaluations) over the ten replications per benchmark problem. We choose this (for BBOB unusual) target due to (1) the theoretical optimum of HPO problems being unknown and (2) **Random** being considered a strong baseline in HPO [2]. To bring all ERTs on the same scale, we computed the ERT ratios between optimizers and **Random** per benchmark problem which further allows us to aggregate these ratios over benchmark problems². We visualize these aggregated ERT ratios separately for the dimensionality of benchmark problems in Fig. 3. We observe that average ERT ratios of **MBO** and **CMAES** are comparably similar for BBOB and HPO problems although the tendency that these optimizers become even more efficient with increasing dimensionality is less pronounced on the HPO problems. **Grid** generally falls behind and **GENSA** shows lacklustre performance on HPO.

5 ELA Feature Space Analysis

For each HPO and BBOB problem, we use $50D$ points sampled by LHS (Min-Max) as an initial design for computing ELA features. We normalize the search space to the unit cube and standardize objective function values per benchmark problem ($(y - \hat{\mu})/\hat{\sigma}$) prior to calculating ELA features. This is done to counter potential artefacts that could be seen in ELA features solely due to different value ranges in decision and, in particular, in objective space. We calculate the feature sets `ela_meta`, `ic`, `ela_distr`, `nb` and `disp`, which were introduced in Sect. 2, using the `flacco` R package [18].

To answer the question whether ELA can be used to distinguish HPO from BBOB problems, we construct a binary classification task using ELA features to predict the label “HPO” vs. “BBOB”. We use a decision tree and estimate

² Following [17], optimizers that did not meet the target in any run were assigned an ERT of the worst ERT on a benchmark problem multiplied by a factor of 10.

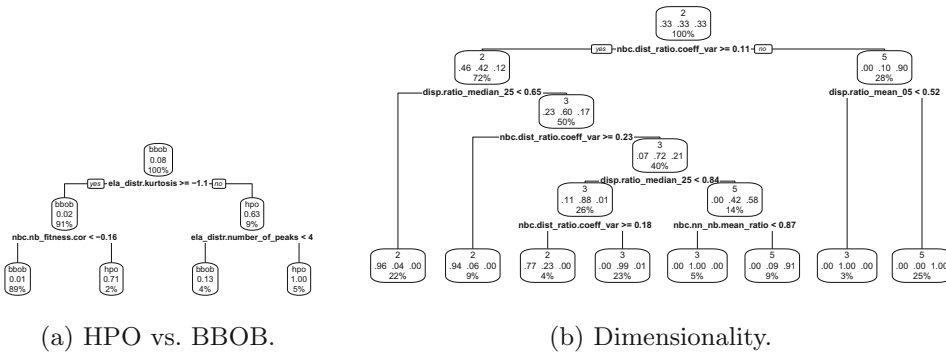


Fig. 4. Decision trees for classifying benchmark problems into HPO or BBOB problems (left) and classifying the dimensionality of BBOB problems (right).

the generalization error via 10 times repeated 10-fold cross-validation (stratified for the target). We obtain an estimated classification error of 3.54%. Figure 4a illustrates the decision tree obtained after training on all data. We observe that only few ELA features are needed to correctly classify problems: HPO problems tend to exhibit a lower `ela_distr.kurtosis` combined with more `ela_distr.number_of_peaks` or show a higher `nbc.nb_fitness.cor` than BBOB problems if the first split with respect to the kurtosis has not been affirmed. This finding is supported by visualizations of the 2D HPO problems, which we present in our [online appendix](#), i.e., most 2D HPO problems have large plateaus resulting in negative kurtosis.

To answer the question whether dimensionality is a different concept for HPO compared to BBOB problems³ we perform the following analysis: We construct a classification task using ELA features to predict the dimensionality of the problem but only use the BBOB subset for the training of a decision tree. We estimate the generalization error via 10 times repeated 10-fold cross-validation (stratified for the target) and obtain an estimated classification error of 7.39%. We then train the decision tree on all BBOB problems (illustrated in Fig. 4b) and determine the holdout performance on the HPO problems and obtain a classification error of 10%. Only few ELA features of the `disp` and `nbc` group are needed to predict the dimensionality of problems with high accuracy. Intuitively, this is sensible, due to `nbc` features involving the calculation of distance metrics (which themselves should be affected by the dimensionality) and both `nbc` and `disp` features being sensible to the multimodality of problems [16, 22] which should also be affected by the dimensionality. Based on the reasonable good hold-out performance of the classifier on the HPO problems, we conclude that “dimensionality” is a similar concept for BBOB and HPO problems.

³ For HPO problems, it is a priori often unclear whether a change in a parameter value also results in relevant objective function changes, i.e., the intrinsic dimensionality of a HPO problem may be lower than the number of hyperparameter suggests.

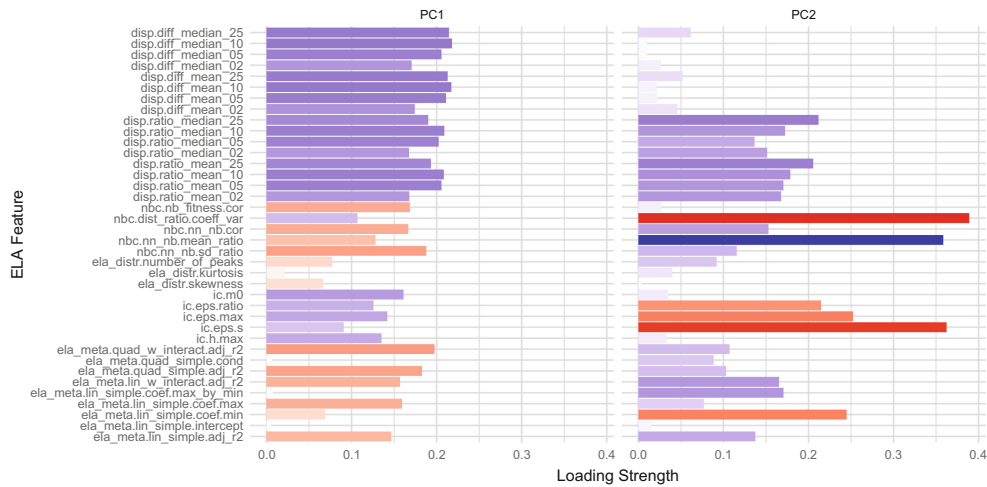


Fig. 5. Factor loadings of ELA features on the first two principle components. Blue indicates a positive loading, whereas red indicates a negative loading.

To gain insight on a meta-level, we performed a PCA on the scaled and centered ELA features of both the HPO and BBOB problems. To ease further interpretation, we select a two component solution that explains roughly 60% of the variance. Figure 5 summarizes factor loadings of ELA features on the first two principle components. Most `disp` features show a medium positive loading on PC1, whereas some `nbc` show medium negative loadings. `ela_meta` features, including R^2 measures of linear and quadratic models, also exhibit medium negative loadings on PC1. We therefore summarize PC1 as a latent dimension that mostly reflects multimodality of problems. Regarding PC2, three features stand out with strong loadings: `nbc.dist_ratio.coeff_var`, `nbc.nn_nb.mean_ratio` and `ic.eps.s`. Moreover, `disp_ratio_*` features generally have a medium negative loading. We observe that all features used by the decision tree in Fig. 4b also have comparably large loadings on PC2. Therefore, we summarize PC2 as an indicator of the dimensionality of problems.

We then performed k-means clustering on the two scaled and centered principal component scores. A silhouette analysis suggested the selection of three clusters. In Fig. 6, we visualize the assignment of HPO and BBOB problems to these three clusters. Labels represent IDs of BBOB and HPO problems. We observe that the dimensionality of problems is almost perfectly reflected in the PC2 alignment. Cluster 2 and 3 can be mostly distinguished along PC2 (cluster 3 contains low dimensional problems and cluster 2 contains higher dimensional problems) whereas cluster 1 contains problems with large PC1 values. HPO problems are exclusively assigned to cluster 2 or 3, exhibiting low variance with respect to their PC1 score, with the PC1 values indicating low multimodality.

As a final analysis we determined the nearest BBOB neighbors of the HPO problems (in ELA feature space based on the cluster analysis, i.e., minimizing the Euclidean distance over the first two principal component scores). For a

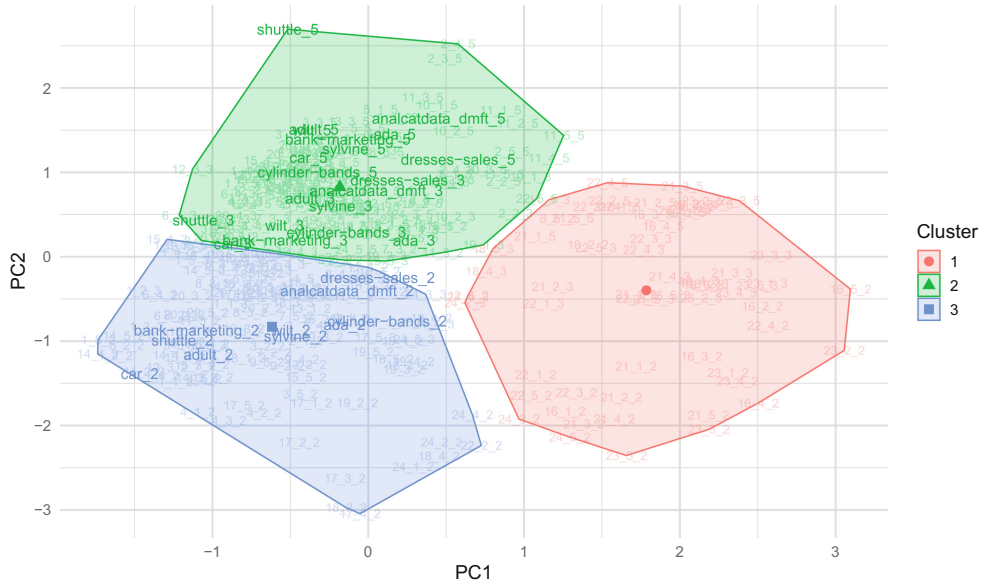


Fig. 6. Cluster analysis of BBOB and HPO problems on the first two principle component scores in ELA feature space.

complete list, see our [online appendix](#). We again computed optimizer rankings based on the average final performance of the optimizers (over the replications), but this time for all HPO problems (regardless their dimensionality) and the subset of BBOB problems that are closest to the HPO problems in ELA feature space (see Fig. 7). Friedman tests indicated overall significant differences in rankings for both HPO ($\chi^2(4) = 104.99, p < 0.001$) and nearest BBOB ($\chi^2(4) = 61.01, p < 0.001$) problems. We observe similar optimizer rankings, with MBO and CMAES outperforming Random or Grid, indicating that closeness in ELA feature space somewhat translates to optimizer performance. Nevertheless, we do have to note that GENSA exhibits poor performance on the HPO problems compared to the nearest BBOB problems. We hypothesize that this may be caused by the performance of GENSA being strongly influenced by its hyperparameter configuration itself and provide an initial investigation in our [online appendix](#).

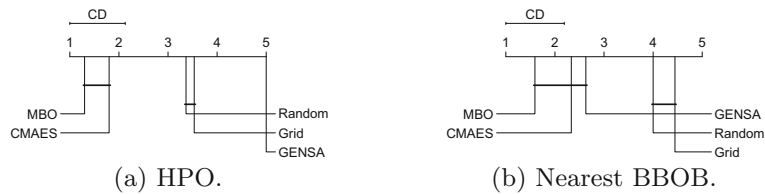


Fig. 7. Critical differences plots for mean ranks of optimizers on all HPO problems (left) and the subset of nearest BBOB problems.

6 Conclusion

In this paper, we characterized the landscapes of continuous hyperparameter optimization problems using ELA. We have shown that ELA features can be used to (1) accurately distinguish HPO from BBOB problems and (2) classify the dimensionality of problems. By performing a cluster analysis in ELA feature space, we have shown that our HPO problems mostly position themselves with BBOB problems of little multimodality, mirroring the results of [30,32]. Determining the nearest BBOB neighbor of HPO problems in ELA feature space allowed us to investigate performance differences of optimizers with respect to HPO problems and their nearest BBOB problems and we observed comparably similar performance. We believe that this work is an important first step in identifying BBOB problems that can be used in lieu of real HPO problems when, for example, configuring or developing novel HPO methods.

Our work still has several limitations. A major one is that traditional ELA is only applicable to continuous HPO problems, which constitute a minority of real-world problems. In many practical applications, search spaces include categorical and conditionally active hyperparameters – so-called hierarchical, mixed search spaces [34]. In such scenarios, measures such as the number of local optima, fitness-distance correlation or auto-correlation of fitness along a path of a random walk [10,13] can be used to gain insight into the fitness landscape. Another limitation is that our studied HPO problems all stem from tuning XGBoost, with little variety of comparably low dimensional search spaces, which limits the generalizability of our results.

In future work, we would like to extend our experiments to cover a broader range of HPO settings, in particular different learners and search spaces, but also data sets. We also want to reiterate that HPO is generally noisy and expensive. In our benchmark experiments, costly 10-fold cross-validation with a fixed instantiating per data set was employed to reduce noise to a minimal level. Future work should explore the effect of the variance of the estimated generalization error on the calculation and usage of ELA features which poses a serious challenge for ELA applied to HPO in practice. Besides, we used logloss as a performance metric which by definition is rather “smooth” compared to other metrics such as the classification accuracy (but the concrete choice of performance metric typically depends on the concrete application at hand). Moreover, ELA requires the evaluation of an initial design, which is very costly in the context of HPO. In general, HPO often can be performed with evaluations on multiple fidelity levels, i.e., by reducing the size of training data, and plenty of HPO methods make use of this resulting in significant speed-up [21]. Future work could explore the possibility of using low fidelity evaluations for the initial design required by ELA and how multiple fidelity levels of HPO affect ELA features.

We consider our work as pioneer work and hope to ignite the research interest in studying the landscape properties of HPO problems going beyond fitness measures. We envision that, by improved understanding of HPO landscapes and identifying relevant landscape properties, better optimizers may be designed, and eventually instance-specific algorithm selection and configuration for HPO may be enabled.

Acknowledgement. This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. H. Trautmann, R. P. Prager and P. Kerschke acknowledge support by the European Research Center for Information Systems (ERCIS). Further, L. Schäpermeier and P. Kerschke acknowledge support by the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI) Dresden/Leipzig. L. Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8).

References

1. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Per instance algorithm configuration of CMA-ES with limited budget. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 681–688 (2017)
2. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Proceedings of the 24th International Conference on Neural Information Processing Systems, pp. 2546–2554 (2011)
3. Bischl, B., et al.: Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. [arXiv:2107.05847](https://arxiv.org/abs/2107.05847) [cs, stat] (2021)
4. Bischl, B., et al.: OpenML benchmarking suites. In: Vanschoren, J., Yeung, S. (eds.) Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, vol. 1 (2021)
5. Bischl, B., Mersmann, O., Trautmann, H., Preuß, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, pp. 313–320 (2012)
6. Bosman, A.S., Engelbrecht, A.P., Helbig, M.: Progressive gradient walk for neural network fitness landscape analysis. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1473–1480 (2018)
7. Bossek, J.: smooF: Single- and multi-objective optimization test functions. R J. **9**(1), 103–113 (2017). <https://journal.r-project.org/archive/2017/RJ-2017-004/index.html>
8. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016)
9. Doerr, C., Dreó, J., Kerschke, P.: Making a case for (hyper-)parameter tuning as benchmark problems. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1755–1764 (2019)

10. Hains, D.R., Whitley, L.D., Howe, A.E.: Revisiting the big valley search space structure in the TSP. *J. Oper. Res. Soc.* **62**(2), 305–312 (2011). <https://doi.org/10.1057/jors.2010.116>
11. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2010: Experimental setup. Research Report RR-7215, Inria (2010). <https://hal.inria.fr/inria-00462481>
12. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical report RR-6829, Inria (2009). <https://hal.inria.fr/inria-00362633/document>
13. Hernando, L., Mendiburu, A., Lozano, J.A.: An evaluation of methods for estimating the number of local optima in combinatorial optimization problems. *Evol. Comput.* **21**(4), 625–658 (2013)
14. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning. TSS-CML, Springer, Cham (2019). <https://doi.org/10.1007/978-3-030-05318-5>
15. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. *J. Global Optim.* **13**(4), 455–492 (1998). <https://doi.org/10.1023/A:1008306431147>
16. Kerschke, P., Preuss, M., Wessing, S., Trautmann, H.: Detecting funnel structures by means of exploratory landscape analysis. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 265–272 (2015)
17. Kerschke, P., Trautmann, H.: Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evol. Comput.* **27**(1), 99–127 (2019)
18. Kerschke, P., Trautmann, H.: Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the R-package Flacco. In: Bauer, N., Ickstadt, K., Lübke, K., Szepannek, G., Trautmann, H., Vichi, M. (eds.) Applications in Statistical Computing. SCDAKO, pp. 93–123. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25147-5_7
19. Lang, M., et al.: mlr3: a modern object-oriented machine learning framework in R. *J. Open Source Softw.* **4**(44), 1903 (2019)
20. Le Riche, R., Picheny, V.: Revisiting Bayesian optimization in the light of the COCO benchmark. *Struct. Multidiscip. Optim.* **64**(5), 3063–3087 (2021). <https://doi.org/10.1007/s00158-021-02977-1>
21. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.* **18**(1), 6765–6816 (2017)
22. Lunacek, M., Whitley, D.: The dispersion metric and the CMA evolution strategy. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 477–484 (2006)
23. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, pp. 829–836 (2011)
24. Mersmann, O., Preuss, M., Trautmann, H., Bischl, B., Weihs, C.: Analyzing the BBOB results by means of benchmarking concepts. *Evol. Comput.* **23**(1), 161–185 (2015)
25. Muñoz, M.A., Kirley, M., Halgamuge, S.K.: Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Trans. Evol. Comput.* **19**(1), 74–87 (2014)
26. Pimenta, C.G., de Sá, A.G.C., Ochoa, G., Pappa, G.L.: Fitness landscape analysis of automated machine learning search spaces. In: Paquete, L., Zarges, C. (eds.) EvoCOP 2020. LNCS, vol. 12102, pp. 114–130. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43680-3_8

27. Prager, R.P.: pflacco: A Python Interface of the R Package Flacco, April 2022. <https://github.com/Reiyan/pflacco>
28. Prager, R.P., Trautmann, H., Wang, H., Bäck, T.H.W., Kerschke, P.: Per-instance configuration of the modularized CMA-ES by means of classifier chains and exploratory landscape analysis. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 996–1003. IEEE (2020)
29. Probst, P., Boulesteix, A.L., Bischl, B.: Tunability: Importance of hyperparameters of machine learning algorithms. *J. Mach. Learn. Res.* **20**(53), 1–32 (2019)
30. Pushak, Y., Hoos, H.: Algorithm configuration landscapes: In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) PPSN 2018, Part II. LNCS, vol. 11102, pp. 271–283. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99259-4_22
31. Pushak, Y., Hoos, H.H.: Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 245–253 (2020)
32. Pushak, Y., Hoos, H.H.: AutoML landscapes. *ACM Trans. Evol. Learn. Optim. (TELO)* (2022, in print). <https://www.cs.ubc.ca/labs/algorithms/Projects/ACLandscapes/PusHoo22a.pdf>
33. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2021). <https://www.R-project.org/>
34. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 847–855 (2013)
35. Traoré, K.R., Camero, A., Zhu, X.X.: Fitness landscape footprint: A framework to compare neural architecture search problems. [arXiv:2111.01584](https://arxiv.org/abs/2111.01584) [cs] (2021)
36. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning. *SIGKDD Explor.* **15**(2), 49–60 (2013)
37. Xiang, Y., Gubian, S., Suomela, B., Hoeng, J.: Generalized simulated annealing for global optimization: The GenSA package. *R J.* **5**(1), 13–28 (2013)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



4. Multi-Objective and Quality Diversity Hyperparameter Optimization

In this chapter, we present contributions to HPO in the presence of more than one objective, namely multi-objective and quality diversity HPO. Vanilla HPO focuses on optimizing a single objective, usually the estimated generalization error. In real-world applications, however, other objectives such as interpretability, fairness, robustness, and computational efficiency are equally relevant. Multi-objective HPO equips us with the ability to optimize multiple, usually conflicting objectives simultaneously.

One application area of multi-objective HPO is given by model performance versus model complexity or interpretability, especially in the context of tabular data. This trade-off between simpler, directly interpretable models such as linear models, additive models, or shallow decision trees and often better performing yet black-box models such as gradient boosted ensembles or deep neural networks has direct consequences regarding the choice of models of practitioners (Velikova and Daniels, 2004; Stiglic et al., 2020; Wang and Lin, 2021). However, quantifying the “interpretability” of ML models is not an easy task (Vellido et al., 2012; Bibal and Frénay, 2016). Nevertheless, to optimize a construct, one must first operationalize it and pioneer work tying interpretability to model complexity to then quantify it, is given by Molnar et al. (2020). Here, the prediction function of a model is decomposed into an intercept, main effects, and a remainder term containing interactions, similar in concept to a functional ANOVA (Hooker, 2007). Based on this decomposition, Molnar et al. (2020) propose the number of features used by a model, the interaction strength of features, and main effect complexity as measures of model complexity and in a broader sense interpretability. Based on such a quantification of interpretability, one can proceed to simultaneously optimize for predictive performance and ease of interpretability via multi-objective optimization. Still, this is rarely done and to the best of our knowledge only briefly explored in Molnar et al. (2020) going beyond investigations that categorize learning algorithms into interpretable or non-interpretable and investigate their performance (Freitas, 2019).

Another highly relevant group of objectives in HPO is given by measures tied to computational efficiency and resource usage. Especially in the field of NAS, latency or memory usage arise as objectives, e.g., when deploying models on edge devices. This has also coined the term hardware-aware NAS (Benmeziane et al., 2021). Often, NAS is formulated as a constrained optimization problem in this setting (Cai et al., 2019; Tan et al., 2019). Other works have formulated NAS as a multi-objective optimization problem with the goal of identifying the different trade-offs tied to the conflict of good performance and low resource usage (Elsken et al., 2019b; Lu et al., 2020). However, neither constrained NAS nor multi-objective NAS is specifically designed to simultaneously identify architectures that perform well while satisfying different hardware constraints. Constrained NAS yields only a single solution that meets a single constraint, whereas multi-objective NAS aims to approximate the entire Pareto front. Here, multi-objective optimization can lead to unnecessary evaluations for regions of the Pareto front that are practically irrelevant.

In the first contributing article *Multi-objective optimization of performance and interpretability of tabular supervised machine learning models* we present a model-agnostic framework for jointly optimizing predictive performance and interpretability of tabular supervised ML models via HPO. To operationalize interpretability, we follow the general ideas of Molnar et al. (2020), however, we propose the number of features used by a model, interaction sparsity measured by the number of interactions in a model, and the number of non-monotone features used by a model as measures for model interpretability. Interaction sparsity and sparsity of non-monotone features are intuitive measures and immediately reflect if features behave in simple, understandable ways. The paper is not only an illustration of how multi-objective HPO can be used to identify trade-offs in performance and interpretability but also has technical contributions. To allow for efficient optimization, we augment the search space of hyperparameters with feature selection, interaction, and monotonicity constraints. We then show that a reformulation as a grouping problem of features allows for optimization via an EA that operates on both the search space of hyperparameters as well as a group structure of features and their interaction and monotonicity constraints. This allows for finding models that achieve similar or better performance than state-of-the-art models solely optimized for performance while improving interpretability.

In the second contributing article *Tackling neural architecture search with quality diversity optimization* we approach the problem of obtaining multiple well-performing architectures for different hardware constraints in a single optimization run. We show that this problem can be formulated as a quality diversity optimization problem and introduce three novel quality diversity NAS algorithms building upon existing BO and bandit ideas in the field of HPO and combine them with algorithmic ideas of the field of quality diversity optimization. Our proposed methods outperform multi-objective NAS in terms of solution quality and computational efficiency on a variety of benchmarks and we further demonstrate their usefulness for the practical scenario of model compression. This work not only demonstrates that quality diversity can be used to identify multiple well-performing yet diverse architectures in a single optimization run but also bridges a gap between two research communities.

4.1. Multi-Objective Optimization of Performance and Interpretability of Tabular Supervised Machine Learning Models

Contributing article:

L. Schneider, B. Bischl, and J. Thomas. Multi-objective optimization of performance and interpretability of tabular supervised machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 538–547, 2023. <https://dl.acm.org/doi/abs/10.1145/3583131.3590380>.

Copyright information:

This article is licensed under the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).

Author contributions:

Lennart Schneider is the first author of this manuscript, contributing the following. Lennart Schneider, Bernd Bischl, and Janek Thomas conceptualized the project, developing the idea of multi-objective HPO of performance and interpretability, which involved creating new measures for interpretability based on past work of the research group and designing an efficient algorithm for solving the optimization problem. Under the supervision of Janek Thomas, Lennart Schneider devised the evolutionary strategy (EAGGA) suitable for HPO of performance and interpretability and formulated the problem as a feature grouping problem. The idea and development of using feature, interaction, and monotonicity detectors was conceptualized by Janek Thomas and developed by Lennart Schneider and Janek Thomas, with the supervision and input of Bernd Bischl. The benchmark design was jointly developed by Lennart Schneider and Janek Thomas. Lennart Schneider implemented all code used for the benchmarks, conducted the formal analysis, analyzed the results, and created all visualizations presented in the work. Lennart Schneider wrote the initial draft of the manuscript. All authors collaboratively contributed to the revision process.

Supplementary material available at:

- Code: https://github.com/slds-lmu/paper_2023_eagga
- R package: <https://github.com/sumny/eagga>
- arXiv version: <https://arxiv.org/abs/2307.08175>



Multi-Objective Optimization of Performance and Interpretability of Tabular Supervised Machine Learning Models

Lennart Schneider
LMU Munich & Munich Center for
Machine Learning (MCML)
Munich, Germany
lennart.schneider@stat.uni-
muenchen.de

Bernd Bischl
LMU Munich & Munich Center for
Machine Learning (MCML)
Munich, Germany
bernd.bischl@stat.uni-muenchen.de

Janek Thomas
LMU Munich & Munich Center for
Machine Learning (MCML)
Munich, Germany
janek.thomas@stat.uni-muenchen.de

ABSTRACT

We present a model-agnostic framework for jointly optimizing the predictive performance and interpretability of supervised machine learning models for tabular data. Interpretability is quantified via three measures: feature sparsity, interaction sparsity of features, and sparsity of non-monotone feature effects. By treating hyperparameter optimization of a machine learning algorithm as a multi-objective optimization problem, our framework allows for generating diverse models that trade off high performance and ease of interpretability in a single optimization run. Efficient optimization is achieved via augmentation of the search space of the learning algorithm by incorporating feature selection, interaction and monotonicity constraints into the hyperparameter search space. We demonstrate that the optimization problem effectively translates to finding the Pareto optimal set of groups of selected features that are allowed to interact in a model, along with finding their optimal monotonicity constraints and optimal hyperparameters of the learning algorithm itself. We then introduce a novel evolutionary algorithm that can operate efficiently on this augmented search space. In benchmark experiments, we show that our framework is capable of finding diverse models that are highly competitive or outperform state-of-the-art XGBoost or Explainable Boosting Machine models, both with respect to performance and interpretability.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning; Feature selection.**

KEYWORDS

supervised learning, performance, interpretability, tabular data, multi-objective, evolutionary computation, group structure

ACM Reference Format:

Lennart Schneider, Bernd Bischl, and Janek Thomas. 2023. Multi-Objective Optimization of Performance and Interpretability of Tabular Supervised Machine Learning Models. In *Genetic and Evolutionary Computation Conference (GECCO '23)*, July 15–19, 2023, Lisbon, Portugal. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3583131.3590380>



This work is licensed under a Creative Commons Attribution International 4.0 License.

GECCO '23, July 15–19, 2023, Lisbon, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0119-1/23/07.
<https://doi.org/10.1145/3583131.3590380>

1 INTRODUCTION

Tabular data are highly relevant for numerous application areas such as finance, bio-informatics, and medical diagnosis. State-of-the-art learning algorithms for tabular data include tree-based methods, e.g., gradient boosted trees (with larger depth) [20] such as XGBoost [8] and LightGBM [33], or random forests [6], which often still outperform deep neural networks [25], although the performance gap has recently shrunk considerably [23, 25, 31, 50]. To achieve peak predictive performance, AutoML tools such as AutoGluon-Tabular [15] or AutoSklearn [19] often make further use of ensembling and stacking multiple models. Moreover, careful hyperparameter optimization of learning algorithms is typically required to yield well performing models [47, 52].

While good predictive performance is generally of central importance, many applications desire or even require models to fulfill additional criteria, such as *interpretability* or *sparseness*. For example a model used for medical diagnosis that achieves high accuracy but lacks interpretability, such as black box models like gradient boosted trees or deep neural networks, may encounter difficulties in gaining trust and adoption. In contrast, a model that can provide insights into its reasoning, even if it has slightly lower performance, is more likely to be trusted and used in real-world scenarios. In the field of Interpretable Machine Learning [41], two different approaches for achieving *interpretability* of models have broadly emerged: (i) to only consider learning algorithms that induce “interpretable” models due to their simple intrinsic nature (e.g., logistic regression, decision trees, rule-based systems or generalized additive models) or (ii) to use post-hoc methods – which can either be model-agnostic, such as partial dependence plots (PDP) [20] or accumulated local effects (ALE) [1], or model-specific – to gain insight into the inner workings of a model.

When working with tabular data in real-world situations, finding the “right” model can be cumbersome and involves time-consuming manual trial and error. Often, various learning algorithms are tried to produce different models, which are then inspected to select a final model based on concrete user preferences at hand. While this process may be feasible if the goal is to “simply” find a good-performing model, it becomes inefficient if additional criteria such as feature *sparseness*, few *interactions* of features, or *monotonicity* of feature effects are also to be considered. In particular, monotonicity can be highly relevant in practice, as frequently only a model consistent with domain knowledge is acceptable to domain experts. For example, in credit loan approval, models are often required to be monotone with respect to the decision variables involved [53]. Our framework allows automatic generation of a set of models that

balance performance and *interpretability*. Formally, this requires two things: **(i)** a way to measure the interpretability of models on a global scale, and **(ii)** an efficient approach for solving the arising *multi-objective* optimization problem.

Our Contributions. We introduce a general, model-agnostic framework for jointly optimizing the predictive performance and interpretability of supervised machine learning models for tabular data. To achieve this, we propose a quantification of the *interpretability* of models on a global scale based on three measures: feature sparsity, interaction sparsity of features, and sparsity of non-monotone feature effects. We then formulate a multi-objective optimization problem of performance and interpretability over the hyperparameter search space of a learning algorithm, which is augmented by incorporating feature selection as well as interaction and monotonicity constraints into the hyperparameter search space. As a solution to the optimization problem, we present a novel hyperparameter optimization algorithm that can operate efficiently on this augmented search space, making use of the principles of evolutionary computation by treating feature selection as well as the specification of interaction and monotonicity constraints of features as a grouping problem.

2 RELATED WORK

When choosing a learning algorithm that induces interpretable models – e.g., logistic regression models, Elastic-Nets [58], or generalized additive models (GAMs) [28] – one typically loses predictive performance compared to black box models obtained via, e.g., tree based ensembles [10]. However, the downside of these black box models is that their interpretability is hindered by potentially plenty of interaction effects of features and non-linear or non-monotone feature effects. The Explainable Boosting Machine (EBM) [39, 40] positions itself between comparably poor-performing but intelligible models and well-performing but unintelligible models. EBM is a tree-based, cyclic gradient boosting GAM using automatic interaction detection based on FAST [40] to include a given number of second-order interactions in the model. EBM often yields good predictive performance [45] while being more intelligible than black box models. Nevertheless, EBM has some drawbacks: **(i)** EBM is comparably slow to train, as it relies on a large number of boosting steps with a small learning rate to cycle through all features¹, **(ii)** EBM naturally cannot induce a sparse model, as all features are included in a round robin fashion, and the contribution of each feature to a final prediction is therefore non-zero, **(iii)** as a result of the large number of boosting steps, EBM often fits highly non-linear and non-monotone shape functions (resulting in rather complex relationships of features and target), and, relatedly, **(iv)** EBM cannot handle monotonicity constraints during training – i.e., if it is known (or even required) that a feature should have a monotone increasing effect on the target variable, EBM can neither make use of this information nor guarantee such an effect.

A popular approach for constructing sparser models is given by feature selection, which is also related to the complexity and intelligibility of a model [2, 5, 26]. While feature selection can also be performed in the context of unsupervised learning [27], we focus on the supervised learning context. Here, the goal of feature

selection is to select only a subset of relevant features while still constructing a model with good predictive performance. There are two model-agnostic approaches to feature selection [26]: feature filters and feature wrappers. Feature filters use proxy measures that are cheap to compute to rank features by their potential explanatory power independent of the concrete learning algorithm being used. Popular examples include measures based on information theory, correlation, distance, or consistency [11]. In contrast to feature filters, feature wrappers directly optimize predictive performance over the space of feature subsets [35]. As every feature subset evaluation requires one or multiple model fits, making exhaustive search infeasible, a discrete black box optimization search strategy (such as a greedy search or an evolutionary algorithm [55]) is necessary. On the one hand, feature selection is often considered a single-objective optimization problem, and the feature selection step is only used to optimize performance [35]. On the other hand, feature selection can also be framed as a multi-objective optimization problem, maximizing predictive performance and feature sparsity simultaneously [2, 54]. Finally, recent work also explored the idea of identifying sets of features without predefined grouping [29].

Looking at measures for interpretability of models on a global scale, Molnar and colleagues [42] were among the first to explicitly propose model-agnostic measures of model complexity. They quantify model complexity by decomposing the prediction function of any model into a sum of components with increasing dimensionality, based on which they derive three measures: the number of features used by a model, the interaction strength of features, and the main effect complexity of features.

3 THEORETICAL BACKGROUND

Consider the supervised learning problem of inferring a model from labeled data \mathcal{D} with n observations where each observation $(\mathbf{x}^{(i)}, y^{(i)})$ consists of a p -dimensional feature vector $\mathbf{x}^{(i)}$. We assume that \mathcal{D} has been sampled i.i.d. from an underlying, unknown distribution, $\mathcal{D} \sim (\mathbb{P}_{\mathbf{x}y})^n$. A learning algorithm or *inducer* \mathcal{I} configured by hyperparameters $\lambda \in \Lambda$ maps a data set \mathcal{D} to a model \hat{f} , i.e., $\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}, (\mathcal{D}, \lambda) \mapsto \hat{f}_{\mathcal{D}, \lambda}$, where $\mathbb{D} := \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n$ is the set of all data sets, Λ is the search space of hyperparameters, and \mathcal{H} is the hypothesis space of models. In general, one is interested in constructing a model $\hat{f}_{\mathcal{D}, \lambda} = \mathcal{I}(\mathcal{D}, \lambda)$ that minimizes the *generalization error*², $\text{GE}(\hat{f}_{\mathcal{D}, \lambda}) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathbb{P}_{\mathbf{x}y}} [L(\hat{f}_{\mathcal{D}, \lambda}(\mathbf{x}), y)]$, where L is a loss function measuring discrepancy between the prediction and true label. However, the generalization error can only be estimated using in-sample data, $\widehat{\text{GE}}(\mathcal{I}_{\lambda}, \mathcal{D})$, through a resampling technique such as cross-validation. For more details, see, e.g., [3, 18].

3.1 Multi-Objective Hyperparameter Optimization

Let $c_1 : \Lambda \rightarrow \mathbb{R}, \dots, c_m : \Lambda \rightarrow \mathbb{R}, m \in \mathbb{N}$ denote m evaluation criteria of machine learning models. Note that evaluation criteria usually also depend on the data set and resampling technique at hand (which we omit here for clarity). Define $c : \Lambda \rightarrow$

¹Which we also observed in our benchmark experiments.

²With a slight abuse of notation, we will write \mathcal{I}_{λ} to denote that a certain hyperparameter configuration λ is fixed, i.e., $\mathcal{I}_{\lambda}(\mathcal{D}) = \mathcal{I}(\mathcal{D}, \lambda)$ with λ fixed.

\mathbb{R}^m to assign an m -dimensional cost vector to a hyperparameter configuration $\lambda \in \Lambda$. The general multi-objective hyperparameter optimization problem is then defined as $\min_{\lambda \in \Lambda} c(\lambda) = \min_{\lambda \in \Lambda} (c_1(\lambda), c_2(\lambda), \dots, c_m(\lambda))$. Generally, there is no single hyperparameter configuration that minimizes all criteria, as these criteria typically compete with one another. Therefore, focus is given to the concept of Pareto optimality and the set of Pareto optimal configurations: A hyperparameter configuration $\lambda \in \Lambda$ (Pareto-)dominates another configuration $\lambda' \in \Lambda$, written as $\lambda \prec \lambda'$, if and only if

$$\begin{aligned} \forall i \in \{1, \dots, m\} : c_i(\lambda) &\leq c_i(\lambda') \wedge \\ \exists j \in \{1, \dots, m\} : c_j(\lambda) &< c_j(\lambda'). \end{aligned}$$

The set of Pareto optimal solutions is therefore defined as $\mathcal{P} := \{\lambda \in \Lambda \mid \nexists \lambda' \in \Lambda \text{ s.t. } \lambda' \prec \lambda\}$. The image of \mathcal{P} under c , $c(\mathcal{P})$, is called the Pareto front. The goal of multi-objective optimization is to find a set of configurations $\hat{\mathcal{P}}$ so that $c(\hat{\mathcal{P}})$ approximates the true Pareto front well.

A popular quality indicator of multi-objective optimization is given by the dominated Hypervolume [57]. The Hypervolume of an approximation of the Pareto front $c(\hat{\mathcal{P}})$ is defined as the combined volume of the dominated hypercubes of all solution points with respect to a reference point $r \in \mathbb{R}^m$. For more details on multi-objective hyperparameter optimization in general as well as an overview of recent applications, we refer to [32, 43].

3.2 Quantifying Interpretability

We propose a quantification of interpretability that is conceptually similar to [42], but our measures and their operationalization differ. As measures for the interpretability of a model on a global scale, we propose to use feature sparsity, interaction sparsity of features, and sparsity of non-monotone features. All our measures are based on the prediction function $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}^g$ of a model³.

To define whether feature j is used by the model, we can determine whether the prediction function changes if the value of x_j changes, i.e., $\hat{f}(x_1, \dots, x'_j, \dots, x_p) \neq \hat{f}(x_1, \dots, x_j, \dots, x_p)$ whenever $x'_j \neq x_j$. The (relative) number of features used by a model, NF , can then be defined as

$$\begin{aligned} NF(\hat{f}) := |\{j \in \{1, \dots, p\} : \exists x_j, x'_j \in \mathcal{X}_j, x'_j \neq x_j \text{ s.t.} \\ \hat{f}(x_1, \dots, x'_j, \dots, x_p) \neq \hat{f}(x_1, \dots, x_j, \dots, x_p)\}|/p. \end{aligned} \quad (1)$$

Similarly, we want to define whether two features j and k interact. A prediction function \hat{f} of a model exhibits an interaction between two features j and k if the difference in the value of $\hat{f}(\mathbf{x})$ as a result of changing the value of x_j depends on the concrete value of x_k [21]. Consequently, given no interaction of features j and k , \hat{f} can be decomposed into $\hat{f}(\mathbf{x}) = f_{-j}(\mathbf{x}_{-j}) + f_{-k}(\mathbf{x}_{-k})$ where \mathbf{x}_{-j} and \mathbf{x}_{-k} are feature vectors excluding x_j and respectively x_k . The (relative) number of interactions in a model, NI , can then be defined as

$$\begin{aligned} NI(\hat{f}) := |\{(j, k), j, k \in \{1, \dots, p\}, k > j : \nexists f_{-j}, f_{-k} \text{ s.t.} \\ \hat{f}(\mathbf{x}) = f_{-j}(\mathbf{x}_{-j}) + f_{-k}(\mathbf{x}_{-k})\}|/((p(p-1))/2). \end{aligned} \quad (2)$$

³For regression, g is 1, while in classification the output usually represents the g decision scores or posterior probabilities of the g candidate classes. Without loss of generalization, we will assume $g = 1$ in the following.

If the hypothesis space of an inducer is restricted to only contain models including main effects and second-order interaction effects of features, NI is a direct measure of the violation of interaction sparsity of a model. However, if the hypothesis space contains models that include higher order interaction effects, NI falls short in penalizing such higher order interactions. To penalize the inclusion of many pairwise interactions and higher order interactions, we assume transitivity with respect to the interaction of features, i.e., if feature j and k and k and l interact, we also count an interaction of feature j and l .

Finally, we define feature j to have a monotone increasing effect if it holds that whenever $x_j \leq x'_j$, one has that $\hat{f}(x_1, \dots, x_j, \dots, x_p) \leq \hat{f}(x_1, \dots, x'_j, \dots, x_p)$. Analogously, we define feature j to have a monotone decreasing effect. The (relative) number of non-monotone features in a model, NNM , is then given by

$$\begin{aligned} NNM(\hat{f}) := |\{j \in \{1, \dots, p\} : (\exists x_j, x'_j \in \mathcal{X}_j, x_j \leq x'_j \text{ s.t.} \\ \hat{f}(x_1, \dots, x_j, \dots, x_p) > \hat{f}(x_1, \dots, x'_j, \dots, x_p)) \wedge \\ (\exists x_j, x'_j \in \mathcal{X}_j, x_j \leq x'_j \text{ s.t.} \\ \hat{f}(x_1, \dots, x_j, \dots, x_p) < \hat{f}(x_1, \dots, x'_j, \dots, x_p))\}|/p. \end{aligned} \quad (3)$$

Based on these formal definitions, NF , NI , and NNM can be operationalized in different ways. For example, NF can be estimated via a sampling procedure, as described in [42]. Similarly, NI could in principle be estimated based on the partial dependence function [21] or by calculating H-statistics [21] or Greenwell's interaction index [24] for all pairs of features. Depending on the concrete learning algorithm at hand, NF and NI can often also be determined in a straightforward manner by, e.g., looking at features used in splits in a decision tree. In the following, we will exactly determine NF and NI by directly inspecting the resulting model whenever possible. Finally, looking at monotonicity, estimating NNM is arguably difficult. In principle, one could try to test whether a feature has a monotone effect via verification-based testing [49] or adaptive random testing [9]. However, such procedures are always at risk of error, and as monotonicity is typically a hard⁴ requirement of a model [46, 53], we opt to determine NNM based on the configuration of the inducer. This requires the inducer to allow for the specification of monotonicity constraints of features, which is easily achievable for, e.g., tree-based methods or GAMs.

We want to note that a model that has low values with respect to NF , NI and NNM still can be complex and must not necessarily result in being intrinsically interpretable. Nevertheless, we believe that such a model is much more easier to interpret, e.g., based on a post-hoc ALE analysis, compared to a model with high values in NF , NI , or NNM . For instance, if a model uses only few features that have monotone increasing effects and do not interact with each other, the prediction function of the model can be easily summarized. For example, increasing the value of any individual feature would result in an increase in the predicted outcome, regardless of the values of other features. Such a simple and consistent relationship between features and the predicted outcome makes the model more *interpretable*. This direct connection between model

⁴In practice, a feature is typically expected to exhibit a monotone effect, or not, without any in-between or probabilistic formulation.

complexity and ease of interpretability is also the reason why we deem it appropriate to speak of multi-objective optimization of performance and interpretability.

3.3 Multi-Objective Optimization of Performance and Interpretability

We formulate the hyperparameter optimization problem of a learning algorithm as a multi-objective optimization problem with the goal of minimizing the estimated generalization error, NF , NI and NNM . To allow for efficient optimization, we extend the search space of the learning algorithm and include hyperparameters for the selection of features, interaction constraints, and monotonicity constraints of features to be part of the search space. Therefore, we require the learning algorithm to allow for the specification of feature selection as well as interaction and monotonicity constraints of features.

In the following, we denote by $\tilde{\Lambda}$ the extended search space. A hyperparameter configuration $\tilde{\lambda} \in \tilde{\Lambda}$ is given by the tuple $(\lambda, s, \mathbf{I}_s, \mathbf{m}_{\mathbf{I}_s})$. Here, $\lambda \in \Lambda$ is the usual hyperparameter configuration of a learning algorithm, s is a binary vector of length p , indicating selection of features, \mathbf{I}_s is a symmetric matrix of dimension $p \times p$ with $(\mathbf{I}_s)_{jk} = 1$ indicating that features j and k are allowed to interact in a model and 0 indicating otherwise, and $\mathbf{m}_{\mathbf{I}_s}$ is an integer vector of length p indicating monotonicity constraints of features (-1 for monotone decreasing, 1 for monotone increasing, and 0 for unconstrained⁵).

In principle, we could proceed to try solving the multi-objective optimization problem as given in Equation 4:

$$\min_{\tilde{\lambda} \in \tilde{\Lambda}} \left(\widehat{GE}(\mathcal{J}_{\tilde{\lambda}}, \mathcal{D}), NF(\hat{f}_{\mathcal{D}, \tilde{\lambda}}), NI(\hat{f}_{\mathcal{D}, \tilde{\lambda}}), NNM(\hat{f}_{\mathcal{D}, \tilde{\lambda}}) \right) \quad (4)$$

Although this formulation of the optimization problem is quite natural, it has several drawbacks: First, note that the extended search space has become complex, including a binary vector, a quadratic matrix, and an integer vector that scale linearly or quadratic in the number of features p . Second, note that \mathbf{I}_s depends on s , as only features that have been selected can be allowed to interact. Similarly, $\mathbf{m}_{\mathbf{I}_s}$ depends on both \mathbf{I} and s . For example, if feature j is required to have a monotone increasing effect but is also allowed to interact with another feature k , then the monotonicity of feature j may not be guaranteed if feature k does not also have a monotone increasing effect. This is because the interaction between feature j and k can potentially alter the overall effect of feature j , and without the monotonicity constraint on feature k , the monotonicity of feature j may be compromised. Therefore, in the general model-agnostic case, it is most straightforward to require both features j and k to have monotone increasing effects to ensure that the monotonicity of feature j is maintained in the presence of their potential interaction effect.

We will now derive a reformulation of the search space of the optimization problem stated in Equation 4 that is much easier to handle. To do so, recall the definition of an endorelation and the properties reflexive, symmetric, and transitive. Note that a reflexive, symmetric, and transitive endorelation – also called an equivalence

relation – imposes a group structure on a set, i.e., it partitions the set by means of its equivalence classes.

To arrive at an easier formulation of the search space of the optimization problem in Equation 4, we define interactions of features as an endorelation. Let $C = \{1, \dots, p\}$ denote the index set of features and $C_s \subseteq C$ the index set of features selected for inclusion in a model and define an endorelation R on C_s , $R \subseteq C_s \times C_s$. We say feature j and feature k are *allowed to interact* if the model in principle allows for the inclusion of an (interaction) effect of the two, and write jRk . It follows that R is naturally reflexive and symmetric – i.e., if feature j is allowed to interact with feature k , then the reverse also holds, as the interaction of features is non-directional. However, note that the interaction of features must in fact not be transitive – i.e., even if feature j and k and k and l interact in a model, it must not follow that feature j and l also interact. Nevertheless, from a modeling perspective, it is reasonable to *allow* for features j and l to also interact, partially also due to the potential presence of a three-way interaction, which (in the most general scenario) can only be included in a model if R is closed under transitivity (and the same argument can be made for higher-order interactions)⁶. It is therefore natural to always consider the transitive closure of R , resulting in an equivalence relation. This implies that the equivalence classes induced by R partition the index set of selected features and naturally call for working with a *group structure*. Regarding monotonicity constraints of features, we want to note that monotonicity constraints must simply be defined as attributes of the equivalence classes (for the same reason illustrated earlier: if features are allowed to interact, they should share the same monotonicity constraint).

We can now introduce the group structure space \mathcal{G} . Each group structure $G \in \mathcal{G}$ consists of a g -tuple of sets of feature indices with the first set, i.e., group, representing the features that were not selected ($C \setminus C_s$) and all remaining sets resembling the k equivalence classes under the equivalence relation $R \subseteq C_s \times C_s$ of features being *allowed to interact* with each equivalence class also being equipped with a monotonicity attribute. Any group structure can therefore be encoded as follows: $G = (G_1 = C \setminus C_s, G_2 = (E_1, M_{E_1}), \dots, G_g = (E_k, M_{E_k}))$. Here, $E_k \subseteq C_s$ is an index set containing the indices of features part of the k -th equivalence class under R , and $M_{E_k} \in \{-1, 0, 1\}$ is the monotonicity attribute of the k -th equivalence class. We can now reformulate Equation 4 and introduce the augmented search space $\tilde{\Lambda} = \Lambda \times \mathcal{G}$ by considering the group structure $G \in \mathcal{G}$ instead of s , \mathbf{I}_s , and $\mathbf{m}_{\mathbf{I}_s}$. The reformulated search space now consists of the Cartesian product of the search space of the learning algorithm, Λ , and the group structure space \mathcal{G} and each configuration, $\tilde{\lambda}$ of the search space is given by a tuple (λ, G) , which we argue is much easier to optimize. We visualize the components involved in the optimization problem in Figure 1.

4 METHOD

For optimizing the multi-objective optimization problem, we introduce an optimizer consisting of an evolutionary algorithm (EA) for the original search space of the learning algorithm Λ and a so-called grouping genetic algorithm (GGA) [16] for the group structure space \mathcal{G} . We therefore dub our optimizer *EAGGA*.

⁵We will later argue that it suffices to only consider $\{0, 1\}$ as monotonicity constraints.

⁶This is also directly related to the principle of marginality; see, e.g., [44].

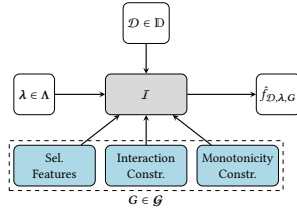


Figure 1: Overview of the components involved in the hyperparameter optimization problem. The inducer is required to allow for the specification of feature selection, as well as interaction and monotonicity constraints of features, which are derived based on the group structure $G \in \mathcal{G}$.

4.1 EAGGA

The combination of using an EA and GGA allows us to jointly operate on the augmented search space $\hat{\Lambda} = \Lambda \times \mathcal{G}$. *EAGGA*'s main routine is heavily inspired by NSGA-II [12]. NSGA-II is an evolutionary multi-objective algorithm making use of the concepts of non-dominated sorting and crowding distance to select individuals for survival close to the Pareto front that also cover a wide spread along the Pareto front. In each generation, NSGA-II iterates through reproduction, crossover, mutation, and survival steps that generate the population of the next generation. In *EAGGA*, we perform parent selection via a binary tournament selection and simply apply suitable crossover and mutation operators to hyperparameters of the original search space ($\lambda \in \Lambda$) and group structures ($G \in \mathcal{G}$) next to each other to produce offspring.

4.1.1 EA Operators. For the original hyperparameters of the learning algorithm ($\lambda \in \Lambda$), we use the Cartesian product of operators that operate in different ways on the different parameter types [37]. We use a global crossover probability of $p = 0.7$ and a global mutation probability of $p = 0.3$. All hyperparameters undergo uniform crossover ($p = 0.5$) for recombination. Numeric and integer hyperparameters undergo Gaussian mutation ($p = 0.2, \sigma = 0.1$; values min-max scaled to $[0, 1]$ prior to mutation and re-transformed afterwards; values rounded to the closest integer in the case of integer hyperparameters), while categorical hyperparameters undergo uniform mutation ($p = 0.2$). The choice of operators and probabilities of crossover and mutation were mostly inspired by [2].

4.1.2 GGA Operators. Group structures ($G \in \mathcal{G}$) undergo mutation and crossover operators inspired by the original work of Falkenauer [16, 17]. We again use a global crossover probability of $p = 0.7$ and a global mutation probability of $p = 0.3$. Recall that a group structure is encoded as $G = (G_1 = C \setminus C_s, G_2 = (E_1, M_{E_1}), \dots, G_g = (E_k, M_{E_k}))$ where $G_1 = C \setminus C_s$ is an index set of features not selected and each $E_k \subseteq C_s$ is an index set of features part of the k -th equivalence class under the equivalence relation R of features being *allowed to interact*, and $M_{E_k} \in \{-1, 0, 1\}$ is the monotonicity attribute of the k -th equivalence class. The basic idea of a GGA is to apply operators directly on the group structure. For crossover, we select two crossing sites, delimiting the crossing section, in each of the two parents (e.g., $G_1G_2|G_3|G_4$ and $H_1|H_2H_3|H_4H_5$; G used for

the first parent and H for the second parent). We then inject the contents (groups together with their monotonicity attributes) of the crossing section of the first parent at the first crossing site of the second parent (e.g., inserting G_3 into the second parent, resulting in $H_1G_3H_2H_3H_4H_5$). Finally, we remove all items (feature indices) from the old groups now occurring twice in the second parent. For example, assume $H_3 = (\{1, 2, 3\}, 0)$ and $G_3 = (\{3\}, 1)$, then after inserting G_3 into the second parent, H_3 is given by $(\{1, 2\}, 0)$. In the case of the first group, i.e., the index set of features not selected, being injected, we simply add these indices to the first group of the parent. To create the second offspring, we swap the roles of the parents. For more details on the GGA crossover, see [17]. For mutation, we simply assign each feature index a new group membership with probability $p = 0.2$ and sample a new monotonicity attribute for each group with probability $p = 0.2$. To allow for more precise handling of the group structure, we incorporate a feedback loop into *EAGGA*: After evaluating an offspring, we can determine the actual features and interactions (closed under transitivity) as included in the model⁷ and update the group structure G of each offspring. In Section 5.3 and the supplementary material, we present results of an ablation study investigating the effect of turning off either crossover or mutation of group structures or both, where we observed that in general both of them are needed for good performance.

4.2 Initializing the Group Structures

As hyperparameter optimization is costly, we strive to make *EAGGA* more sample-efficient. We use three detectors (feature, interaction, and monotonicity) to find better initial population group structures. An ablation study in Section 5.3 shows that these detectors substantially improve *EAGGA*'s (anytime) performance.

4.2.1 Feature Detector. The goal of a *feature detector* is to quantify the importance of features so that the probability of selecting an important feature j (i.e., $j \in C_s$) can be increased. Formally, a feature detector maps a data set \mathcal{D} to a p -dimensional vector of real valued scores with the j -th element corresponding to the score of the j -th feature. In *EAGGA*, we use feature filters. A feature filter measures feature importance using a fast proxy, such as the entropy-based information gain filter [36], which calculates the difference between the target variable's entropy and the joint entropy conditioned on the feature. Based on the filter score for each feature, we can then weight the probability of selecting a feature. To determine the number of selected features S of a member of the initial population, we sample a random integer between 1 and p from a truncated geometric distribution similarly as in [2]. The features that are actually selected are then determined by sampling from all binary vectors s of length p that sum to S with weighted probabilities according to the feature filter scores.

4.2.2 Interaction Detector. The idea of a (pairwise) *interaction detector* is to quantify the importance of interactions of features so that the probability of those features being in the same group (i.e., the same equivalence class under the equivalence relation R *allowed to interact*) can be increased. Formally, an interaction detector maps

⁷The group structure only imposes an upper constraint, meaning that the resulting model may use all or some of the selected features, and the same applies to interactions.

a data set \mathcal{D} to a symmetric, real valued $p \times p$ matrix with the element at the j -th row and k -th column corresponding to the score of the j -th and k -th feature⁸. Recall that in *EAGGA*, the first group G_1 of a group structure G is always given by the indices of features that are not selected. To initialize the remaining groups, we make use of the FAST algorithm [40]. FAST allows for efficient quantification of the importance of all pairwise interactions of features based on the residual sums of squares when extending a main effects model to include an interaction effect. To determine the number of included interactions I of a member of the initial population, we sample a random integer between 1 and $(p(1-p))/2$ from a truncated geometric distribution. The actual groups are then determined by considering the I most important pairwise interactions according to FAST, constructing an equivalence relation R allowed to interact, and deriving the equivalence classes under R .

4.2.3 Monotonicity Detector. Using a *monotonicity detector* is helpful due to two reasons: First, recall that the monotonicity attribute of a group can in principle either be -1 (monotone decreasing), 1 (monotone increasing), or 0 (unconstrained). This is somewhat redundant, as a monotone decreasing feature effect (without loss of generalization, we assume purely numeric features) can always be realized by enforcing a monotone increasing effect and swapping the sign of the feature itself. Therefore, by detecting whether a monotone feature effect should be increasing or decreasing we can encode monotonicity constraints more efficiently. Second, by quantifying the mismatch in model fit between enforcing monotonicity and no constraint, the monotonicity detector can bias the probability of the monotonicity attribute being unconstrained. Formally, a monotonicity detector maps a data set \mathcal{D} to a p -dimensional vector of real valued scores with the j -th element corresponding to the score of the j -th feature where the sign of the score indicates the direction of monotonicity and the magnitude of the score reflects the strength of the monotone relationship between the feature and the target variable. In *EAGGA*, we use the following monotonicity detector: For each feature, we fit a decision tree on sub-sampled data and obtain the predictions. We then calculate Spearman's ρ between the feature values and the target predictions. Finally, we repeat this process 10 times and calculate the average Spearman's ρ , which we scale⁹ to $[0.2, 0.8]$. For each group of features of a member of the initial population, we take the average over the individual scores and use this average as a probability to sample the monotonicity attribute of the group.

5 BENCHMARK EXPERIMENTS

To our best knowledge, *EAGGA* is the first model-agnostic approach to perform *efficient* multi-objective optimization of performance and interpretability of machine learning models by incorporating feature selection as well as interaction and monotonicity constraints into the hyperparameter search space. In our experiments, we combine *EAGGA* with XGBoost ($EAGGA_{XGBoost}$) or XGBoost with a maximum depth fixed to 2 ($EAGGA_{XGBoost_{md2}}$, resulting in second-order interactions being the most complex higher-order interactions that can be picked up by the model). We configure *EAGGA* to use a population size of $\mu = 100$ and an offspring size

of $\nu = 10$, with the comparably large population size being inspired by [2, 54]. One naïve approach to generate a benchmark baseline is to simply use a collection of competitors that all excel at different objectives which *EAGGA* tries to optimize jointly and compare $EAGGA_{XGBoost}$ to the union of the competitors. Another approach is to compare $EAGGA_{XGBoost}$ to standard multi-objective optimization of XGBoost (without augmentation of the search space). Code and supplementary material are released via https://github.com/slds-lmu/paper_2023_eagga.

5.1 EAGGA vs. A Collection of Competitors

We construct a collection of competitors by considering an EBM, Elastic-Net, (untuned) random forest, and XGBoost. An EBM offers good performance with few interactions, an Elastic-Net provides sparse, monotone solutions, while a random forest and XGBoost usually deliver strong results using many features, interactions, and non-monotone effects. We tune the hyperparameters of the EBM, Elastic-Net, and XGBoost via Bayesian Optimization¹⁰ and optimize for predictive performance. For the search spaces of the learning algorithms, see our supplementary material. All learning algorithms are given a budget of 8 hours of sequential runtime on a single CPU (note that this is a disadvantage for *EAGGA*, as each competitor is given the same computational budget and therefore the union of competitors uses substantially more compute budget than *EAGGA*). As a performance metric, we choose the area under the receiver operating characteristic curve (AUC)¹¹. Performance estimation is conducted via nested resampling: As an outer resampling, we use a holdout with a ratio of 2/3, i.e., test performance is evaluated on 1/3 of the data. Hyperparameter optimization is then performed using 5-fold cross-validation on the remaining 2/3 of the data. For $EAGGA_{XGBoost}$ and $EAGGA_{XGBoost_{md2}}$, the Pareto optimal configurations found during optimization are re-evaluated on the test-set. For the EBM, Elastic-Net, random forest, and XGBoost, we re-evaluate the single best-performing configuration (found during optimization) on the test-set. For XGBoost models, NF and NI are determined by actually checking the model and all splits in all trees, whereas NNM is determined based on the monotonicity constraints of features used in the model (only applicable when optimized via *EAGGA*; for the standard XGBoost, we assume NNM to be the same as NF as we consider monotonicity of features to be a hard requirement as explained in Section 3.2). For the EBM, NF is always 1, as EBM cycles through all available features in a round robin fashion, whereas NI is directly given by the value of the hyperparameter `interactions` and we assume NNM to be the same as NF , as EBM does not allow for the specification of monotonicity constraints and cannot guarantee monotone feature effects. For the Elastic-Net, NF is determined by looking at the relative number of non-zero coefficients, whereas NI and NNM are always 0 (no interaction effects are included in the standard Elastic-Net and feature effects are always monotone). Finally, for the random forest, NF and NI are again determined by actually checking the model and all splits in all trees, whereas NNM is again the same as NF (for the same reason as for the standard XGBoost).

¹⁰We employ a Bayesian Optimization variant similarly configured as SMAC [38], i.e., using a random forest as surrogate model and Expected Improvement [30] as acquisition function.

¹¹We minimize the negative AUC.

⁸Note that the diagonal is of no interest and can be set to, e.g., 0.

⁹This is done to allow for some non-determinism during sampling.

All methods are compared on twenty binary classification tasks taken from OpenML CC-18 [4] and the AutoML benchmark [22]. We perform 10 replications of each optimization run on each task with different random seeds to allow for statistical analysis. Criteria for selecting the tasks were fewer than 100000 observations, the number of features being fewer than 1000 as well as numeric features, i.e., we focus on small- to medium-sized tabular data sets. We only consider binary classification tasks, as the EBM until now does not support the inclusion of interaction effects of features in the case of multi-class classification. More details on the data sets can be found in our supplementary material.

As we are comparing a multi-objective optimization framework (*EAGGA*) to a collection of models, we perform the following analysis: For every run on each task, we calculate the dominated Hypervolume of the (test-set) Pareto front of *EAGGA*_{XGBoost} and *EAGGA*_{XGBoost_{md2}} with respect to the reference point $r = (0, 1, 1, 1)^T$ and compare this with the dominated Hypervolume obtained by considering the non-dominated set of the EBM, Elastic-Net, random forest, and XGBoost solutions (evaluated on the test-set). To allow for a fair comparison, we always include a featureless learner that simply predicts the majority class without relying on any features when calculating the dominated Hypervolume¹². Results are given in Figure 2. Note that the number in parentheses after a task name indicates the number of features of the task. Using *EAGGA* results in substantially larger dominated Hypervolume (Wilcoxon signed-ranks test [13] on the mean dominated Hypervolume over replications: $T = 0, p < 0.001$ for *EAGGA*_{XGBoost} vs. competitors and $T = 0, p < 0.001$ for *EAGGA*_{XGBoost_{md2}} vs. competitors).

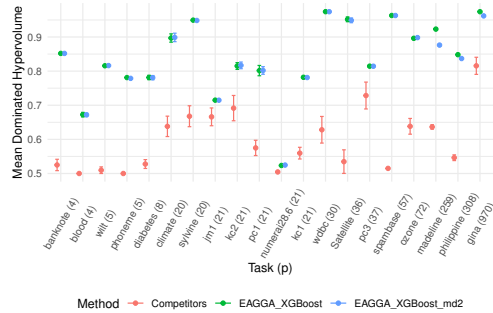


Figure 2: Mean dominated Hypervolume of *EAGGA*_{XGBoost}, *EAGGA*_{XGBoost_{md2}}, and the union of competitors averaged over 10 replications. Bars represent standard errors.

We further determine for each task the fraction of replications where each competitor yields a solution that is Pareto-dominated by the solutions of *EAGGA*_{XGBoost} or *EAGGA*_{XGBoost_{md2}}. Table 1a shows this fraction averaged over all tasks for *EAGGA*_{XGBoost} – i.e., on average, roughly 46% of the EBM solutions are Pareto-dominated

¹²As the resulting point $(-0.5, 0, 0, 0)^T$ will have a large contribution to the dominated Hypervolume, but only *EAGGA* might be able to consistently find a hyperparameter configuration resulting in such a model.

by the solutions found by *EAGGA*_{XGBoost}. Table 1b shows this fraction averaged over all tasks for *EAGGA*_{XGBoost_{md2}}. We also compute the counterpart – i.e., what is the fraction of replications where the whole Pareto set of *EAGGA*_{XGBoost} or *EAGGA*_{XGBoost_{md2}} is dominated by the Pareto set of the union of the competitors. This was never the case, neither for *EAGGA*_{XGBoost} nor *EAGGA*_{XGBoost_{md2}}. We want to note that in some runs, evaluating the initial design during optimization of the EBM took longer than the whole compute budget of 8 hours. In these cases, our fallback was to only evaluate the default configuration suggested by the EBM authors.

In our supplementary material, we also provide an illustrative example of the usage of *EAGGA* relying on the *ozone-level-8hr* task and analyze an exemplary Pareto front. Additionally we analyze the best performing models from each method in terms of AUC and interpretability. Results show that the best models found by *EAGGA* perform similarly to XGBoost models optimized for performance, but use less features, interactions, and non-monotone features, indicating improved interpretability.

Table 1: Mean fraction of runs over tasks and replications where competitors yield a solution that is dominated by *EAGGA*_{XGBoost} or *EAGGA*_{XGBoost_{md2}}.

(a) <i>EAGGA</i> _{XGBoost}			(b) <i>EAGGA</i> _{XGBoost_{md2}}		
Competitor	Mean	SE	Competitor	Mean	SE
EBM	0.46	0.04	EBM	0.36	0.03
Elastic-Net	0.30	0.03	Elastic-Net	0.28	0.03
Random Forest	0.81	0.03	Random Forest	0.74	0.03
XGBoost	0.40	0.03	XGBoost	0.31	0.03

SE = standard error.

5.2 *EAGGA* vs. Multi-Objective XGBoost

We also compare *EAGGA*_{XGBoost} to multi-objective optimization of XGBoost (without augmentation of the search space), which we will refer to as XGBoost_{MO}. As an optimizer, we employ ParEGO [34], a scalarization-based multi-objective Bayesian Optimization algorithm that we configure to use a random forest as surrogate model and Expected Improvement as acquisition function. The search space used within ParEGO is exactly the same as the search space used within *EAGGA* – with the exception that we do not augment the search space to include feature selection, interaction, and monotonicity constraints, as standard multi-objective optimizers such as ParEGO cannot naturally operate on such a search space. The question we want to answer is whether it is sufficient to work on the standard search space with a standard multi-objective optimizer to optimize XGBoost for predictive performance and interpretability. Benchmark tasks and the evaluation protocol are exactly the same as in Section 5.1 – i.e., for *EAGGA*_{XGBoost}, *EAGGA*_{XGBoost_{md2}}, and XGBoost_{MO}, the Pareto optimal configurations found during optimization are re-evaluated on the test-set. For each run on each task, we calculate the dominated Hypervolume of the (test-set) Pareto front of *EAGGA*_{XGBoost}, *EAGGA*_{XGBoost_{md2}}, and XGBoost_{MO}, which we visualize in Figure 3. Again, using *EAGGA* results in usually at least the same and often substantially larger dominated Hypervolume (Wilcoxon signed-ranks test on the mean dominated Hypervolume over replications: $T = 40, p = 0.0076$ for *EAGGA*_{XGBoost}

vs. $XGBoost_{MO}$ and $T = 50, p = 0.02$ for $EAGGA_{XGBoost_{md2}}$ vs. $XGBoost_{MO}$). Notably, the only tasks where $XGBoost_{MO}$ outperforms $EAGGA$ are tasks with few features. In our supplementary material, we also analyze the anytime dominated Hypervolume during optimization (i.e., calculated on the inner resampling).

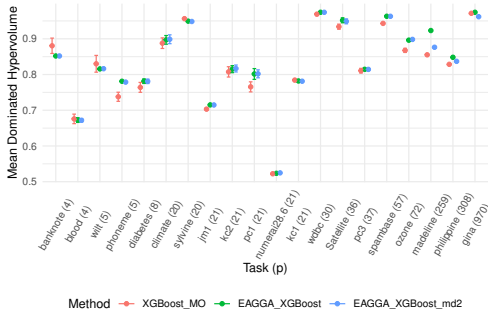


Figure 3: Mean dominated Hypervolume of $EAGGA_{XGBoost}$, $EAGGA_{XGBoost_{md2}}$, and $XGBoost_{MO}$ averaged over 10 replications. Bars represent standard errors.

5.3 An Ablation Study of $EAGGA$

We perform an ablation study of the components of $EAGGA$ with the goal to answer the following questions: (i) Does $EAGGA$ improve over a random search on the same search space? (ii) How important are crossover and respectively mutation of group structures? (iii) What is the benefit of using detectors to initialize the population?

To do so, we rerun all benchmark experiments with different flavors of $EAGGA$ and analyze the mean dominated Hypervolume during optimization, i.e., calculated on the inner resampling. We consider the following modifications or “flavors” of $EAGGA$: (i) Simply performing a random search on $\bar{\Lambda}$ after using $EAGGA$ ’s detectors to initialize the population (Random Search). (ii) Switching off either crossover or mutation of group structures ($G \in \mathcal{G}$) or both (No_Crossover, No_Mutation, No_Cross_Mut). (iii) Switching off the detectors of $EAGGA$ and initializing the population at random (No_Detectors).

We observe that (i) performing a random search performs comparably poorly, (ii) crossover and mutation of group structures are needed for good performance and (iii) using detectors can boost the performance although this is mainly due to using detectors strongly affecting the early performance of $EAGGA$. Conducting a Friedman test [13] on the final mean dominated Hypervolume during optimization indicates significant differences in ranks of optimizers ($\chi^2(6) = 52.99, p < 0.001$). Figure 4 visualizes the corresponding critical difference plot based on the follow up Nemenyi test. For completeness, we also include $XGBoost_{MO}$. For detailed results and discussion, please see our supplementary material.

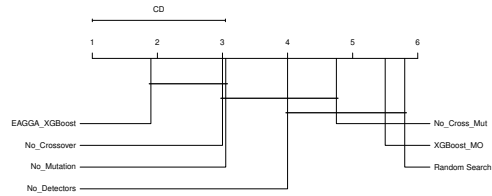


Figure 4: Critical difference plot of the ranks of optimizers based on the final mean dominated Hypervolume during optimization. Lower rank is better.

6 CONCLUSION

We have presented a general model-agnostic framework for jointly optimizing the predictive performance and interpretability of supervised machine learning models for tabular data. $EAGGA$ is a multi-objective optimizer making use of the principles of evolutionary computation to jointly optimize the hyperparameters of a learning algorithm as well as the group structure of features. $EAGGA$ allows for obtaining a set of diverse models in a single optimization run and can outperform state-of-the-art competitors both with respect to performance and interpretability.

In practice, users may have prior knowledge about which features to include, which features should interact or even a requirement for a certain feature to have a monotone effect. Although we studied $EAGGA$ in the context of no prior knowledge, it can be extended to incorporate such information by initializing the population accordingly and preventing crossover and mutation from creating offspring incongruent with the prior.

$EAGGA$ might be especially useful when using deep neural networks as learning algorithms, as Kadra and colleagues [31] demonstrated that strong regularization of neural networks can be a key component to achieving good performance on tabular data. Using $EAGGA$ in combination with neural networks would require the design of a network architecture that allows for the specification of interaction and monotonicity constraints of features. Notable work in this direction has been undertaken by [7, 14, 48, 51, 56].

Finally, it must be noted that $EAGGA$ cannot guarantee that the resulting group structure of a model is sensible, and the structure must be verified by domain experts (with respect to the selection of features, as well as their interaction and monotonicity constraints). Nevertheless, we believe that $EAGGA$ can be of significant interest for a wide variety of users.

ACKNOWLEDGMENTS

The authors of this work take full responsibilities for its content. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8). Lennart Schneider acknowledges funding from the LMU Mentoring Program of the Faculty of Mathematics, Informatics and Statistics.

REFERENCES

- [1] D. W. Apley and J. Zhu. 2020. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82, 4 (2020), 1059–1086.
- [2] M. Binder, J. Moosbauer, J. Thomas, and B. Bischl. 2020. Multi-Objective Hyperparameter Tuning and Feature Selection Using Filter Ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 471–479.
- [3] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, and M. Lindauer. 2021. Hyperparameter Optimization: Foundations, Algorithms, Best Practices, and Open Challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2021), e1484.
- [4] B. Bischl, G. Casalicchio, M. Feurer, P. Gijbbers, F. Hutter, M. Lang, R. Gomes Mantovani, J. N. van Rijn, and J. Vanschoren. 2021. OpenML Benchmarking Suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1.
- [5] B. Bischl, I. Vatulkin, and M. Preuss. 2010. Selecting Small Audio Feature Sets in Music Classification by Means of Asymmetric Mutation. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I* 314–323.
- [6] L. Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
- [7] C.-H. Chang, R. Caruana, and A. Goldenberg. 2022. Node-GAM: Neural Generalized Additive Model for Interpretable Deep Learning. *The Tenth International Conference on Learning Representations, ICLR* (2022).
- [8] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.
- [9] T. Y. Chen, H. Leung, and I. K. Mak. 2005. Adaptive Random Testing. In *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, M. J. Maher (Ed.), 320–329.
- [10] R. Couronné, P. Probst, and A.-L. Boulesteix. 2018. Random Forest versus Logistic Regression: A Large-Scale Benchmark Experiment. *BMC Bioinformatics* 19, 1 (2018), 1–14.
- [11] M. Dash and H. Liu. 1997. Feature Selection for Classification. *Intelligent Data Analysis* 1, 3 (1997), 131–156.
- [12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.
- [13] J. Demsar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.
- [14] A. Dubej, F. Radenovic, and D. Mahajan. 2022. Scalable Interpretability via Polynomials. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35.
- [15] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. In *7th ICML Workshop on Automated Machine Learning*.
- [16] E. Falkenauer. 1993. The Grouping Genetic Algorithms: Widening the Scope of the GA's. *Belgian Journal of Operations Research, Statistics, and Computer Science* 33, 1–2 (1993), 79–102.
- [17] E. Falkenauer. 1996. A Hybrid Grouping Genetic Algorithm for Bin Packing. *Journal of Heuristics* 2, 1 (1996), 5–30.
- [18] M. Feurer and F. Hutter. 2019. Hyperparameter Optimization. In *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren (Eds.). Springer International Publishing, Cham, 3–33.
- [19] M. Feurer, A. Klein, K. Eggenberger, J. Springenberg, M. Blum, and F. Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28.
- [20] J. H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.
- [21] J. H. Friedman and B. E. Popescu. 2008. Predictive Learning via Rule Ensembles. *The Annals of Applied Statistics* 2, 3 (2008), 916–954.
- [22] P. Gijbbers, M. L. P. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. 2022. AMLB: An AutoML Benchmark. *arXiv:2207.12560 [cs.LG]* (2022).
- [23] Y. Gorishniy, I. Rubachev, V. Khurlov, and A. Babenko. 2021. Revisiting Deep Learning Models for Tabular Data. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34.
- [24] B. M. Greenwell, B. C. Boehmke, and A. J. McCarthy. 2018. A Simple and Effective Model-Based Variable Importance Measure. *arXiv:1805.04755 [stat.ML]* (2018).
- [25] L. Grinsztajn, E. Oyallon, and G. Varoquaux. 2022. Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- [26] I. Guyon and A. Elisseeff. 2003. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research* 3 (2003), 1157–1182.
- [27] J. Handl and J. Knowles. 2006. Feature Subset Selection in Unsupervised Learning via Multiobjective Optimization. *International Journal of Computational Intelligence Research* 2, 3 (2006), 217–238.
- [28] T. Hastie and R. Tibshirani. 1986. Generalized Additive Models. *Statistical Science* 1, 3 (1986), 297–310.
- [29] F. Imrie, A. Nordcliffe, P. Liò, and M. van der Schaar. 2022. Composite Feature Selection using Deep Ensembles. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35.
- [30] D. R. Jones, M. Schonlau, and W. J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (1998), 455–492.
- [31] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka. 2021. Well-tuned Simple Nets Excel on Tabular Datasets. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34.
- [32] F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. 2022. Multi-Objective Hyperparameter Optimization - An Overview. *arXiv:2206.07438 [cs.LG]* (2022).
- [33] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30.
- [34] J. Knowles. 2006. ParEGO: A Hybrid Algorithm with On-Line Landscape Approximation for Expensive Multiobjective Optimization Problems. *IEEE Transactions on Evolutionary Computation* 10, 1 (2006), 50–66.
- [35] R. Kohavi and G. H. John. 1997. Wrappers for Feature Subset Selection. *Artificial Intelligence* 97, 1–2 (1997), 273–324.
- [36] C. Largeron, C. Moulin, and M. Géry. 2011. Entropy Based Feature Selection for Text Categorization. In *Proceedings of the 2011 ACM Symposium on Applied Computing*. 924–928.
- [37] R. Li, M. T. M. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. C. Reiber. 2013. Mixed Integer Evolution Strategies for Parameter Optimization. *Evolutionary Computation* 21, 1 (2013), 29–64.
- [38] M. Lindauer, K. Eggenberger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23 (2022), 54–1.
- [39] Y. Lou, R. Caruana, and J. Gehrke. 2012. Intelligible Models for Classification and Regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 150–158.
- [40] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker. 2013. Accurate Intelligible Models with Pairwise Interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 623–631.
- [41] C. Molnar. 2022. *Interpretable Machine Learning* (2 ed.). <https://christophm.github.io/interpretable-ml-book>
- [42] C. Molnar, G. Casalicchio, and B. Bischl. 2020. Quantifying Model Complexity via Functional Decomposition for Better post-hoc Interpretability. In *Machine Learning and Knowledge Discovery in Databases*, P. Celleri and K. Driessens (Eds.), 193–204.
- [43] A. Morales-Hernández, I. van Nieuwenhuysse, and S. Rojas Gonzalez. 2022. A Survey on Multi-Objective Hyperparameter Optimization Algorithms for Machine Learning. *Artificial Intelligence Review* (2022), 1–51.
- [44] J. A. Nelder. 1977. A Reformulation of Linear Models. *Journal of the Royal Statistical Society, Series A (General)* 140, 1 (1977), 48–77.
- [45] H. Nori, S. Jenkins, P. Koch, and R. Caruana. 2019. InterpretML: A Unified Framework for Machine Learning Interpretability. *arXiv:1909.09223 [cs.LG]* (2019).
- [46] R. Potharst and A. J. Feelders. 2002. Classification Trees for Problems with Monotonicity Constraints. *ACM SIGKDD Explorations Newsletter* 4, 1 (2002), 1–10.
- [47] P. Probst, A.-L. Boulesteix, and B. Bischl. 2019. Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *Journal of Machine Learning Research* 20, 53 (2019), 1–32.
- [48] F. Radenovic, A. Dubej, and D. Mahajan. 2022. Neural Basis Models for Interpretability. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35.
- [49] A. Sharma and H. Wehrheim. 2020. Testing Monotonicity of Machine Learning Models. *arXiv:2002.12278 [cs.LG]* (2020).
- [50] R. Shwartz-Ziv and A. Armon. 2022. Tabular Data: Deep Learning is Not All You Need. *Information Fusion* 81 (2022), 84–90.
- [51] M. Tsang, H. Liu, S. Purushotham, P. Murali, and Y. Liu. 2018. Neural Interaction Transparency (NIT): Disentangling Learned Interactions for Improved Interpretability. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31.
- [52] J. N. van Rijn and F. Hutter. 2018. Hyperparameter Importance Across Datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge*

- Discovery and Data Mining*, 2367–2376.
- [53] M. Velikova and H. A. M. Daniels. 2004. Decision Trees for Monotone Price Models. *Computational Management Science* 1 (2004), 231–244.
- [54] B. Xue, W. Fu, and M. Zhang. 2014. Multi-Objective Feature Selection in Classification: A Differential Evolution Approach. In *Simulated Evolution and Learning: 10th International Conference*. 516–528.
- [55] B. Xue, M. Zhang, W. N. Browne, and X. Yao. 2016. A Survey on Evolutionary Computation Approaches to Feature Selection. *IEEE Transactions on Evolutionary Computation* 20, 4 (2016), 606–626.
- [56] Z. Yang, A. Zhang, and A. Sudjianto. 2021. GAMI-Net: An Explainable Neural Network Based on Generalized Additive Models with Structured Interactions. *Pattern Recognition* 120 (2021), 108192.
- [57] E. Zitzler and L. Thiele. 1998. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. 292–304.
- [58] H. Zou and T. Hastie. 2005. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67, 2 (2005), 301–320.

4.2. Tackling Neural Architecture Search with Quality Diversity Optimization

Contributing article:

L. Schneider, F. Pfisterer, P. Kent, J. Branke, B. Bischl, and J. Thomas. Tackling neural architecture search with quality diversity optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, pages 9/1–30, 2022. <https://proceedings.mlr.press/v188/schneider22a.html>.

Copyright information:

This article is licensed under the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).

Author contributions:

Lennart Schneider is the first author of this manuscript, contributing the following. Lennart Schneider conceptualized the project, drawing inspiration from the BOP-Elites algorithm proposed and published by Paul Kent and Juergen Branke, and adapted it to the domain of NAS. Lennart Schneider developed and implemented all algorithmic modifications to BOP-Elites and extensions discussed in the manuscript and implemented the baselines and competitor algorithms. Lennart Schneider conducted all benchmarks, analyzed the results, and created the associated visualizations. Florian Pfisterer prototyped an initial code base for an application to model compression which Lennart Schneider integrated into the existing code base. Lennart Schneider authored the initial draft of the manuscript. Florian Pfisterer authored the section on defining niches within NAS and contributed to the review and iterative revision of the manuscript. Paul Kent and Juergen Branke made significant contributions to the general section on Quality Diversity Optimization by refining its content and ensuring its alignment with the manuscript objectives. Paul Kent provided feedback and valuable insights during the development of the project and the revision process of the manuscript. Juergen Branke, Bernd Bischl and Janek Thomas supervised the research project, contributing input during discussions and project iterations and in reviewing and revising the manuscript. All authors contributed collaboratively to the revision process.

Supplementary material available at:

- Code: https://github.com/slds-lmu/qdo_nas
- arXiv version: <https://arxiv.org/abs/2208.00204>

Tackling Neural Architecture Search With Quality Diversity Optimization

Lennart Schneider¹ Florian Pfisterer¹ Paul Kent² Juergen Branke³ Bernd Bischl¹
Janek Thomas¹

¹Department of Statistics, LMU Munich, Germany

²Mathematics of Real World Systems, University of Warwick, UK

³Warwick Business School, University of Warwick, UK

Abstract Neural architecture search (NAS) has been studied extensively and has grown to become a research field with substantial impact. While classical single-objective NAS searches for the architecture with the best performance, multi-objective NAS considers multiple objectives that should be optimized simultaneously, e.g., minimizing resource usage along the validation error. Although considerable progress has been made in the field of multi-objective NAS, we argue that there is some discrepancy between the actual optimization problem of practical interest and the optimization problem that multi-objective NAS tries to solve. We resolve this discrepancy by formulating the multi-objective NAS problem as a quality diversity optimization (QDO) problem and introduce three quality diversity NAS optimizers (two of them belonging to the group of multifidelity optimizers), which search for high-performing yet diverse architectures that are optimal for application-specific niches, e.g., hardware constraints. By comparing these optimizers to their multi-objective counterparts, we demonstrate that quality diversity NAS in general outperforms multi-objective NAS with respect to quality of solutions and efficiency. We further show how applications and future NAS research can thrive on QDO.

1 Introduction

The goal of neural architecture search (NAS) is to automate the manual process of designing optimal neural network architectures. Traditionally, NAS is formulated as a single-objective optimization problem with the goal of finding an architecture that has minimal validation error [13, 35, 45, 47, 46, 63]. Considerations for additional objectives such as efficiency have led to the formulation of constraint NAS methods that enforce efficiency thresholds [1] as well as multi-objective NAS methods [10, 12, 37, 53, 36] that yield a Pareto optimal set of architectures. However,

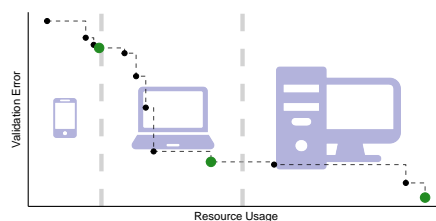


Figure 1: Optimizing neural network architectures for a discrete set of devices. We are interested in the best solution (green) within the constraints of the respective device (dashed vertical lines). Multi-objective optimization, in contrast, approximates the full Pareto front (black).

in most practical applications, we are not interested in the complete Pareto optimal set. Instead, we would like to obtain solutions for a discrete set of scenarios (e.g., end-user devices), which we henceforth refer to as *niches* in this paper. This is illustrated in Figure 1. A concrete example is finding neural architectures for microcontrollers [32] and other edge devices [38], e.g., in μ NAS [32] architectures for “mid-tier” IoT devices are searched. To evaluate the benefits for larger devices, the search would need to be restarted with adapted constraints, thus wasting computational resources. Formulating the search as a multi-objective problem would also waste resources; once an architecture satisfies the constraints of a device, we are not interested in additional trade-offs, and we select only based on the validation error.

We therefore argue that the multi-objective NAS problem *can* and usually *should* be formulated as a *quality diversity optimization* (QDO) problem, which directly corresponds to the actual optimization problem of interest. The main contributions of this paper are: We (1) formulate multi-objective NAS as a QDO problem; (2) show how to adapt black-box optimization algorithms for the QDO setting; (3) modify existing QDO algorithms for the NAS setting; (4) propose novel multifidelity QDO algorithms for NAS; and (5) illustrate that our approach can be used to extend a broad range of NAS methods from conventional to Once-for-All methods.

2 Theoretical Background and Related Work

Let \mathcal{A} denote a search space of architectures and Λ the search space of additional hyperparameters controlling the training of an architecture A . Furthermore, let $f_{\text{err}} : \mathcal{A} \times \Lambda \rightarrow \mathbb{R}$ denote the validation error obtained after training an architecture $A \in \mathcal{A}$ with a set of hyperparameters $\lambda \in \Lambda$ for a given number of epochs ($\lambda_{\text{epoch}} \in \lambda$). Typically, we consider $\lambda \in \Lambda$ to be fixed, except for λ_{epoch} in multifidelity methods, and we therefore omit λ in the following. The goal of single-objective NAS is to find the architecture with the lowest validation error, $A^* := \arg \min_{A \in \mathcal{A}} f_{\text{err}}(A)$.

NAS methods can be categorized along three dimensions: search space, search strategy, and performance estimation strategy [13]. For chain-structured neural networks (simply a connected sequence of layers), cell-based search spaces have gained popularity [46, 45]. In cell-based search spaces, different kinds of cells – typically, a normal cell preserving dimensionality of the input and a reduction cell reducing spatial dimension – are stacked in a predefined arrangement to form a final architecture. Regarding search strategy, popular methods utilize Bayesian optimization (BO) [3, 8, 39, 24, 57], evolutionary methods [41, 34, 47, 46, 12], reinforcement learning [63, 64], or gradient-based algorithms [35, 45]. For performance estimation, popular approaches leverage lower fidelity estimates [31, 14, 64] or make use of learning curve extrapolation [8, 27].

Multi-Objective Neural Architecture Search Contrary to the single-objective NAS formulation, multi-objective NAS does not solely aim for minimizing the validation error but simultaneously optimizes multiple objectives. These objectives typically take resource consumption – such as memory requirements, energy usage or latency – into account [10, 12, 37, 53, 36]. Denote by f_1, \dots, f_k the $k \geq 2$ objectives of interest, where typically $f_1 = f_{\text{err}}$ and denote by $\mathbf{f}(A)$ the vector of objective function values obtained for architecture $A \in \mathcal{A}$, $\mathbf{f}(A) = (f_1(A), \dots, f_k(A))'$. The optimization problem of multi-objective NAS is then formulated as $\min_{A \in \mathcal{A}} \mathbf{f}(A)$. There is no architecture that minimizes all objectives at the same time since these are typically in competition with each other. Rather, there are multiple Pareto optimal architectures reflecting different trade-offs in objectives approximating the true (unknown) Pareto front. An architecture A is said to dominate another architecture A' iff $\forall i \in \{1, \dots, k\} : f_i(A) \leq f_i(A') \wedge \exists j \in \{1, \dots, k\} : f_j(A) < f_j(A')$.

Constrained and Hardware-Aware Neural Architecture Search In contrast, *Constrained NAS* [62, 15, 55] solves the problem of finding an architecture that optimizes one objective (e.g., validation error) with constraints on secondary objectives (e.g., model size). Constraints can be naturally given by the target hardware that a model should be deployed on. *Hardware-Aware NAS* in turn searches for an architecture that trades off primary objectives [60] against secondary, hardware-specific

metrics. In Once-for-All [5], a large supernet is trained which can be efficiently searched for subnets that, e.g., meet latency constraints of target devices. For a recent survey, we refer to [1].

Quality Diversity Optimization The goal of a QDO algorithm is to find a set of high-performing, yet behaviorally diverse, solutions. Similarly to multi-objective optimization, there is no single best solution. However, whereas multi-objective optimization aims for the simultaneous minimization of multiple objectives, QDO minimizes a single-objective function with respect to diversity defined on one or more *feature functions*. A feature function measures a quality of interest and a combination of feature values points to a niche, i.e., a region in *feature space*. QDO could be considered a *set* of constrained optimisation problems over the same input domain where the niche boundaries are constraints in feature space. The key difference is that constrained optimisation seeks a single optimal configuration given some constraints, while QDO attempts to identify the optimal configuration for each of a set of constrained regions simultaneously. In this sense, QDO could be framed as a so-called multi-task optimization problem [43] where each task is to find the best solution belonging to a particular niche.

QDO algorithms maintain an archive of niche-optimal observations, i.e., a best-performing observed solution for each niche. Observations with similar feature values compete to be selected for the archive, and the solution set gradually improves during the optimization process. Once the optimization budget has been spent, QDO algorithms typically return this archive as their solution. QDO is motivated by applications where a group of diverse solutions is beneficial, such as the training of robot movement where a repertoire of behaviours must be learned [7], developing game playing agents with diverse strategies [44], and in automatic design where QDO can be used by human designers to search a large dimensional search space for diverse solutions before the optimization is finished by hand. Work on automatic design tasks have been varied and include air-foil design [18], computer game level design [16], and architectural design [9]. Recently, QDO algorithms were used for illuminating the interpretability and resource usage of machine learning models while minimizing their generalization error [52].

In the earliest examples, Novelty Search (NS; [29]) asks whether diversity alone can produce a good set of solutions. Despite not actively pursuing objective performance, NS performed surprisingly well in some settings and was followed by Novelty Search with Local Competition [30], the first true quality diversity (QD) algorithm. MAP-Elites [42], a standard evolutionary QDO algorithm, partitions the feature space a-priori into niches and attempts to identify the optimal solution in each of these niches. QDO has seen much work in recent years and a variant based on BO, BOP-Elites, was proposed recently [25]. BOP-Elites models the objective and feature functions with surrogate models and implements an acquisition function over a structured archive to achieve high sample efficiency even in the case of black-box features.

3 Formulating Neural Architecture Search as a Quality Diversity Optimization Problem

In the example in Figure 1, a quality diversity NAS (subsequently abbreviated as qdNAS) problem is given by the validation error and three behavioral niches (corresponding to different devices) that are defined via resource usage measured by a single feature function. Let $f_1 : \mathcal{A} \rightarrow \mathbb{R}, A \mapsto f_1(A)$ denote the objective function of interest (in our context, f_{err}). Denote by $f_i : \mathcal{A} \rightarrow \mathbb{R}, A \mapsto f_i(A), i \in \{2, \dots, k\}, k \geq 2$ the feature function(s) of interest (e.g., memory usage). Behavioral niches $N_j \subseteq \mathcal{A}, j \in \{1, \dots, c\}, c \geq 1$ are sets of architectures characterized via niche-specific boundaries $\mathbf{b}_{ij} = [l_{ij}, u_{ij}] \subseteq \mathbb{R}$ on the images of the feature functions. An architecture A belongs to niche N_j if its values with respect to the feature functions lie between the respective boundaries, i.e.:

$$A \in N_j \iff \forall i \in \{2, \dots, k\} : f_i(A) \in \mathbf{b}_{ij}.$$

The goal of a QDO algorithm is then to find for each behavioral niche N_j the architecture that minimizes the objective function f_i :

$$A_j^* := \arg \min_{A \in N_j} f_i(A).$$

In other words, the goal is to obtain a set of architectures $\mathcal{S} := \{A_1^*, \dots, A_c^*\}$ that are diverse with respect to the feature functions and yet high-performing.

A Remark about Niches In the classical QDO literature, niches are typically constructed to be pairwise disjoint, i.e., a configuration can only belong to a single niche (or none) [42, 25]. However, depending on the concrete application, relaxing this constraint and allowing for *overlap* can be beneficial. For example, in our context, an architecture that fits on a mid-tier device should also be considered for deployment on a higher-tier device, i.e., in Figure 1, the boundaries indicated by vertical dashed lines resemble the respective upper bound of a niche whereas the lower bound is unconstrained. This results in niches being nested within each other, i.e., $N_1 \subseteq N_2 \subseteq \dots \subseteq N_c \subseteq \mathcal{A}$, with N_1 being the most restrictive niche, followed by N_2 . In Supplement A, we further discuss different ways of constructing niches in the context of NAS.

3.1 Quality Diversity Optimizers for Neural Architecture Search

As the majority of NAS optimizers are iterative, we first demonstrate how any iterative optimizer can in principle be turned into a QD optimizer. Based on this correspondence, we introduce three novel QD optimizers for NAS: BOP-Elites*, qdHB and BOP-ElitesHB. Let $f_i : \mathcal{A} \rightarrow \mathbb{R}, A \mapsto f_i(x)$ denote the objective function that should be minimized. In each iteration, an iterative optimizer proposes a new configuration (e.g., architecture) for evaluation, evaluates this configuration, potentially updates the incumbent (best configuration evaluated so far) if better performance has been observed, and updates its archive. For generic pseudo code, see Supplement B.

Moving to a QDO problem, there are now feature functions $f_i : \mathcal{A} \rightarrow \mathbb{R}, A \mapsto f_i(A)$, $i \in \{2, \dots, k\}, k \geq 2$, and niches $N_j, j \in \{1, \dots, c\}, c \geq 1$, defined via their niche-specific boundaries $\mathbf{b}_{ij} = [l_{ij}, u_{ij}] \subseteq \mathbb{R}$ on the images of the feature functions. Any iterative single-objective optimizer must then keep track of the best incumbent per niche (often referred to as an *elite* in the QDO literature) and essentially becomes a QD optimizer (see Algorithm 1). The challenge

Algorithm 1: Generic pseudo code for an iterative quality diversity optimizer.

Input : $f_1, f_i, i \in \{2, \dots, k\}, k \geq 2, N_j, j \in \{1, \dots, c\}, c \geq 1, \mathcal{D}_{\text{design}}, n_{\text{total}}$
Result : $S = \{A_1^*, \dots, A_c^*\}$

- 1 $\mathcal{D} \leftarrow \mathcal{D}_{\text{design}}$
- 2 **for** $j \leftarrow 1$ **to** c **do**
- 3 | $A_j^* \leftarrow \arg \min_{A \in \mathcal{D}_{|N_j}} f_1(A)$ # initial incumbent of niche N_j based on archive
- 4 **end**
- 5 **for** $n \leftarrow 1$ **to** n_{total} **do**
- 6 | Propose a new candidate A^* # subroutine
- 7 | Evaluate $y \leftarrow f_1(A^*), \forall i \in \{2, \dots, k\} : z_i \leftarrow f_i(A^*)$
- 8 | **if** $A^* \in N_j \wedge y < f_1(A_j^*)$ **then**
- 9 | | $A_j^* \leftarrow A^*$ # update incumbent of niche N_j
- 10 | **end**
- 11 | $\mathcal{D} \leftarrow \mathcal{D} \cup \{(A^*, y, z_2, \dots, z_k)\}$
- 12 **end**

in designing an efficient and well-performing QD optimizer now mostly lies in proposing a new candidate for evaluation that considers improvement over all niches.

Bayesian Optimization A recently proposed model-based QD optimizer, BOP-Elites [25], extends BO [54, 17] to QDO. BOP-Elites relies on Gaussian process surrogate models for the objective function and all feature functions. New candidates for evaluation are selected by a novel acquisition function – the expected joint improvement of elites (EJIE), which measures the expected improvement to the ensemble problem of identifying the best solution in every niche:

$$\alpha_{\text{EJIE}}(A) := \sum_{j=1}^c \mathcal{P}(A \in N_j | \mathcal{D}) \mathbb{E}_y [\mathbb{I}_{|N_j}(A)]. \quad (1)$$

Here, $\mathcal{P}(A \in N_j | \mathcal{D})$ is the posterior probability of A belonging to niche N_j , and $\mathbb{E}_y [\mathbb{I}_{|N_j}(A)]$ is the expected improvement (EI; [23]) with respect to niche N_j :

$$\mathbb{E}_y [\mathbb{I}_{|N_j}(A)] = \mathbb{E}_y \left[\max \left(f_{\min_{N_j}} - y, 0 \right) \right],$$

where $f_{\min_{N_j}}$ is the best observed objective function value in niche N_j so far, and y is the surrogate model prediction for A . A new candidate is then proposed by maximizing the EJIE, i.e., Line 6 in Algorithm 1 looks like the following: $A^* \leftarrow \arg \max_{A \in \mathcal{A}} \alpha_{\text{EJIE}}(A)$.

BOP-Elites* In order to adapt BOP-Elites for NAS, we introduce several modifications. First, we employ truncated (one-hot) path encoding [56, 57]. In path encoding, architectures are transformed into a set of binary features indicating presence for each path of the directed acyclic graph from the input to the output. By then truncating the least-likely paths, the encoding scales linearly in the size of the cell [57] allowing for an efficient representation of architectures. Second, we substitute the Gaussian process surrogate models used in BOP-Elites with random forests [4] allowing us to model non-continuous hyperparameter spaces. Random forests have been successfully used as surrogates in BO [21, 33], often performing on a par with ensembles of neural networks [57] in the context of NAS [51, 59]. Third, we introduce a local mutation scheme similarly to the one used by the BANANAS algorithm [57] for optimizing the infill criterion EJIE: Since our aim is to find high quality solutions across all niches, we maintain an archive of the incumbent architecture in each niche and perform local mutations on each incumbent. We refer to our adjusted version as BOP-Elites* in the remainder of the paper to emphasize the difference from the original algorithm. For the initial design, we sample architectures based on adjacency matrix encoding [56].

Multifidelity Optimizers For NAS, performance estimation is the computationally most expensive component [13], and almost all NAS optimizers can be made more efficient by allowing access to cheaper, lower fidelity estimates [13, 31, 14, 64]. By evaluating most architectures at lower fidelity and only promoting promising architectures to higher fidelity, many more architectures can be explored given the same total computational budget. The fidelity parameter is typically the number of epochs over which an architecture is trained.

qdHB One of the most prominent multifidelity optimizers is Hyperband (HB; [31]), a multi-armed bandit strategy that uses repeated Successive Halving (SH; [22]) as a subroutine to identify the best configuration (e.g., architecture) among a set of randomly sampled ones. Given an initial and maximum fidelity, a scaling parameter η , and a set of configurations of size n , SH evaluates all configurations on the initial smallest fidelity, then sorts the configurations by performance and only keeps the best $1/\eta$ configurations. These configurations are then trained with fidelity increased by a factor of η . This process is repeated until the maximum fidelity for a single configuration is reached. HB repeatedly runs SH with different sized sets of initial configurations called brackets. Only two inputs are required: R , the maximum fidelity and η , the scaling parameter that controls the proportion of configurations discarded in each round of SH. Based on these inputs, the number s_{\max} and size n_i of different brackets is determined. To adapt HB to the QD setting, we must track the incumbent architecture in each niche and promote configurations based on their performance within the respective niche (see Supplement B): To achieve this, we choose the top $\lfloor n_i/\eta \rfloor$ configurations

to be promoted uniformly over the c niches (done in the `topk_qdo` function), i.e., we iteratively select one of the niches uniformly at random and choose the best configuration observed so far that has yet not been selected for promotion until $\lfloor n_i/\eta \rfloor$ configurations have been selected. Note that during this procedure, it may happen that not enough configurations belonging to a specific niche have been observed yet. In this case, we choose any configuration uniformly at random over the set of all configurations that have yet to be promoted. With those modifications, we propose qdHB, as a multifidelity QD optimizer.

BOP-ElitesHB While HB typically shows strong anytime performance [31], it only samples configurations at random and is typically outperformed by BO methods with respect to final performance if optimizer runtime is sufficiently large [14]. BOHB [14] combines the strengths of HB and BO in a single optimizer, resulting in strong anytime performance and fast convergence. This approach employs a fidelity schedule similar to HB to determine how many configurations to evaluate at which fidelity but replaces the random selection of configurations in each HB iteration by a model-based proposal. In BOHB, a Tree Parzen Estimator [2] is used to model densities $l(A) = p(y < \alpha|A, \mathcal{D})$ and $g(A) = p(y > \alpha|A, \mathcal{D})$, and candidates are proposed that maximize the ratio $l(A)/g(A)$, which is equivalent to maximizing EI [2]. Based on BOP-Elites* and qdHB, we can now derive the QD Bayesian optimization Hyperband optimizer (BOP-ElitesHB): Instead of selecting configurations at random at the beginning of each qdHB iteration, we propose candidates that maximize the EJIE criterion. This sampling procedure is described in Supplement B.

4 Main Benchmark Experiments and Results

We are interested in answering the following research questions: **(RQ1)** *Does qdNAS outperform multi-objective NAS if the optimization goal is to find high-performing architectures in pre-defined niches?* **(RQ2)** *Do multifidelity qdNAS optimizers improve over full-fidelity qdNAS optimizers?* To answer these questions, we benchmark our three qdNAS optimizers – BOP-Elites*, qdHB, and BOP-ElitesHB – on the well-known NAS-Bench-101 [61] and NAS-Bench-201 [11] and compare them to three multi-objective optimizers adapted for NAS: ParEGO*, moHB*, and ParEGOHB as well as a simple Random Search¹.

Experimental Setup It is important to compare optimizers using analogous implementation details. We therefore use truncated path encoding and random forest surrogates throughout our experiments for all model-based optimizers. Furthermore, we use local mutations as described in [57] in order to optimize acquisition functions in BOP-Elites*, BOP-ElitesHB, ParEGO*, and ParEGOHB. To control for differences in implementation, we re-implement all optimizers and take great care in matching the original implementations.

We provide full details regarding implementation in Supplement B and only briefly introduce conceptual differences: ParEGO* is a multi-objective optimizer based on ParEGO [28] and only deviates from BOP-Elites* in that it considers a differently scalarized objective in each iteration, which is optimized using local mutations similar to the acquisition function optimization of BOP-Elites*. moHB* is an extension of HB to the multi-objective setting (promoting configurations based on non-dominated sorting with hypervolume contribution for tie breaking, for similar approaches see, e.g., [48, 49, 50, 19]). ParEGOHB is a model-based extension of moHB* that relies on the ParEGO scalarization [28] and on the same acquisition function optimization as ParEGO*.

All optimizers were evaluated on NAS-Bench-101 (Cifar-10, validation error as the first objective and the number of trainable parameters as the feature function/second objective) and NAS-Bench-201 (Cifar-10, Cifar-100, ImageNet16-120, validation error as the first objective and latency as the feature function/second objective). For multifidelity, we train architectures for 4, 12, 36, 108 epochs on NAS-Bench-101 and for 2, 7, 22, 67, 200 epochs on NAS-Bench-201 (reflecting $\eta = 3$ in the HB variants). As the optimization budget, we consider 200 full architecture evaluations (resulting in

¹using adjacency matrix encoding [56]

a total budget of 21600 epochs for NAS-Bench-101 and 40000 epochs for NAS-Bench-201). For each of these four settings, we construct three different scenarios by considering different niches of interest with respect to the feature function, resulting in a total of 12 benchmark problems. In the *small/medium/large* settings, two, five and ten niches are considered, respectively. Niches are constructed to be overlapping, and boundaries are defined based on percentiles of the feature function. For the *small* setting, the boundary is given by the 50% percentile ($q_{50\%}$), effectively resulting in two niches with boundaries $[0, q_{50\%})$ and $[q_{50\%}, \infty)$. For the *medium* and *large* settings, percentiles indicating progressively larger niches were used, ranging from: 1% to 30% and 70% respectively. More details on the niches can be found in Supplement C. All runs were replicated 100 times.

Results As an anytime performance measure, we are interested in the validation error obtained for each niche, which we aggregate in a single performance measure as $\sum_{j=1}^c f_{\text{err}}(A_j^*)$, i.e., we consider the sum of validation errors over the best-performing architecture per niche. If a niche is empty, we assign a validation error of 100 as a penalty (this is common practice in QDO, i.e., if no solution has been found for a niche, this niche is assigned the worst possible objective function value [25]). For the final performance, we also consider the analogous test error. Figure 2 shows the anytime performance of optimizers. We observe that model-based optimizers (BOP-Elites* and ParEGO*) in general strongly outperform Random Search, and BO HB optimizers (BOP-ElitesHB and ParEGOHB) generally outperform their full-fidelity counterparts, although this effect diminishes with increasing optimization budget. In general, HB variants that do not rely on a surrogate model (qdHB and moHB*) show poor performance compared to the model-based optimizers. Moreover, especially in the *small* number of niches setting, QD strongly outperforms multi-objective optimization. Mean ranks of optimizers with respect to final validation and test performance are given in Table 1. For completeness, we also report critical differences plots of these ranks in Supplement C.

We also conducted two four-way ANOVAs on the average performance after half and all of the optimization budget is used, with the factors problem (benchmark problem), multifidelity (whether an optimizer uses multifidelity), QDO (whether an optimizer is a QD optimizer) and model-based (whether the optimizer relies on a surrogate model)². For half the budget used, results indicate significant main effects of the factors multifidelity ($F(1) = 19.13, p = 0.0001$), QDO ($F(1) = 11.08, p = 0.0017$) and model-based ($F(1) = 21.13, p < 0.0001$). For all of the budget used, the significance of multifidelity diminishes, whereas the main effects of QDO ($F(1) = 18.31, p = 0.0001$) and model-based ($F(43.44), p < 0.0001$) are still significant. We can conclude that QDO in general outperforms competitors when the goal is to find high-performing architectures in pre-defined niches. Multi-fidelity optimizers improve over full-fidelity optimizers but this effect diminishes with increasing budget. Detailed results are reported in Supplement C.

Regarding efficiency, we analyzed the expected running time (ERT) of the QD optimizers given the average performance of the respective multi-objective optimizers after half of the optimization budget: For each benchmark problem, we computed the mean validation performance of each multi-objective optimizer after having spent half of its optimization budget and investigated the ERT of the analogous³ QD optimizer. For each benchmark problem, we then computed the ratio of ERTs between multi-objective and QD optimizers and averaged them over the benchmark problems. For BOP-ElitesHB, we observe an average ERT ratio of 2.41, i.e., in expectation, BOP-ElitesHB is a factor of 2.41 faster than ParEGOHB in reaching the average performance of ParEGOHB (after half the optimization budget). For qdHB and BOP-Elites*, the average ERT ratios are 1.14 and 1.44. We conclude that all QD optimizers are more efficient than their multi-objective counterparts. More details can be found in Supplement C.

²For this analysis, we excluded qdHB and moHB* due to their lackluster performance.

³BOP-ElitesHB for ParEGOHB, qdHB for moHB*, and BOP-Elites* for ParEGO*

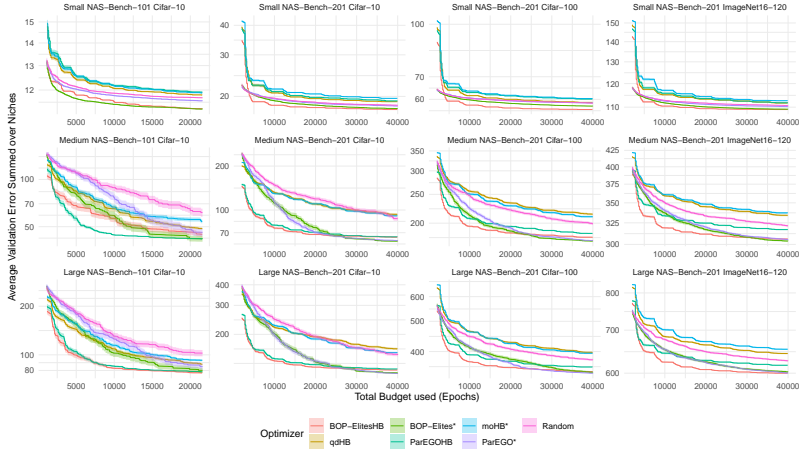


Figure 2: Anytime performance of optimizers. Ribbons represent standard errors over 100 replications. x-axis starts after 10 full-fidelity evaluations.

Table 1: Ranks of optimizers with respect to final performance, averaged over benchmark problems.

Mean Rank (SE)	BOP-ElitesHB	qdHB	BOP-Elites*	ParEGOHB	moHB*	ParEGO*	Random
Validation	2.08 (0.29)	5.92 (0.26)	1.83 (0.21)	4.25 (0.48)	6.42 (0.15)	2.58 (0.31)	4.92 (0.34)
Test	1.42 (0.19)	5.00 (0.17)	2.08 (0.23)	4.33 (0.50)	6.50 (0.19)	3.25 (0.35)	5.42 (0.42)

5 Additional Experiments and Applications

In this section, we illustrate how qdNAS can be used beyond the scenarios investigated so far and present results of additional experiments ranging from a comparison of qdNAS to multi-objective NAS on the MobileNetV3 search space to an example on how to incorporate QDO in existing frameworks such as Once-for-All [5] or how to use qdNAS for model compression.

Benchmarks on the MobileNetV3 Search Space We further investigated how qdNAS compares to multi-objective NAS on a search space that is frequently used in practice [20]. We consider CNNs divided into a sequence of units with feature map size gradually being reduced and channel numbers being increased. Each unit consists of a sequence of layers where only the first layer has stride 2 if the feature map size decreases and all other layers in the units have stride 1. Units can use an arbitrary number of layers (elastic depth chosen from $\{2, 3, 4\}$) and for each layer, an arbitrary number of channels (elastic width chosen from $\{3, 4, 6\}$) and kernel sizes (elastic kernel size chosen from $\{3, 5, 7\}$) can be used. Additionally, the input image size can be varied (elastic resolution ranging from 128 to 224 with a stride 4). For more details on the search space, see [5]. To allow for reasonable runtimes we use accuracy predictors (based on architectures trained and evaluated on ImageNet as described in [5]) and resource usage look-up tables of the Once-for-All module [5, 6] and construct a surrogate benchmark. As an objective function we select the validation error and as a feature function/second objective the latency (in ms) when deployed on a Samsung Note 10 (batch size of 1), or the number of FLOPS (M) used by the model. So far, we have only investigated qdNAS in the context of $k = 2$, i.e., considering one objective and one feature function. Here, we additionally consider a setting of $k = 3$, by incorporating both latency and the size of the model (in MB) as feature functions/second and third objective. We compare BOP-Elites* to ParEGO* and a

Random Search due to the accuracy predictors not supporting evaluations at multiple fidelities. We again construct three scenarios by considering different niches of interest with respect to the feature functions taking inspiration from latency and FLOPS constraints as used in [5] (details are given in Table 4 in Supplement C). Optimizers are given a total budget of 100 architecture evaluations. Figure 3 shows the anytime performance of optimizers with respect to the validation error summed over niches (averaged over 100 replications). BOP-Elites* strongly outperforms the competitors on all benchmark problems. More details are provided in Supplement D.

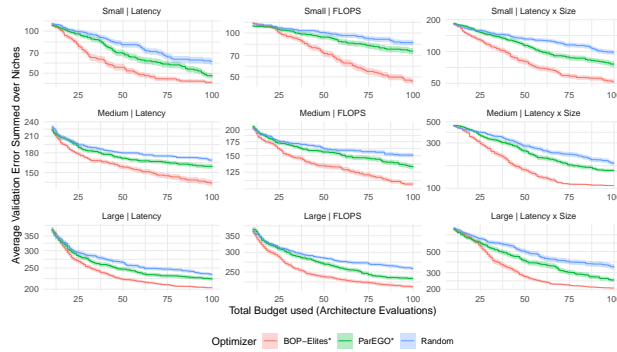


Figure 3: MobileNetV3 search space. Anytime performance of optimizers. Ribbons represent standard errors over 100 replications. x-axis starts after 10 evaluations.

Making Once-for-All Even More Efficient In Once-for-All [5], an already trained supernet is searched via regularized evolution [46] for a well performing subnet that meets hardware requirements of a target device relying on an accuracy predictor and resource usage look-up tables. This is sensible if only a single solution is required, however, if various subnets meeting different constraints on the same device are desired, repeated regularized evolution is not efficient. Moreover, look-up tables do not generalize to new target devices in which case using as few as possible architecture evaluations suddenly becomes relevant again. We notice that the search for multiple architectures within Once-for-All can again be formulated as a QDO problem and therefore compare MAP-Elites [42] to regularized evolution when performing a joint search for architectures meeting different latency constraints on a Samsung Note 10. Results are given in Table 2 with MAP-Elites consistently outperforming regularized evolution, making this novel variant of Once-for-All even more efficient. More details are provided in Supplement E.

Table 2: MAP-Elites vs. regularized evolution within Once-for-All.

Method	Best Validation Error for Different Latency Constraints						
	[0, 15)	[0, 18)	[0, 21)	[0, 24)	[0, 27)	[0, 30)	[0, 33)
Reg. Evo.	21.57 (0.01)	20.34 (0.02)	19.29 (0.01)	18.48 (0.02)	17.81 (0.02)	17.40 (0.02)	17.06 (0.02)
MAP-Elites	21.60 (0.01)	20.28 (0.01)	19.21 (0.01)	18.39 (0.01)	17.70 (0.01)	17.25 (0.01)	16.90 (0.01)

Average over 100 replications based on the accuracy predictor of Once-for-All [5, 6]. Standard errors in parentheses. Reg. Evo. = regularized evolution.

Applying qdNAS to Model Compression We are interested in deploying a MobileNetV2 across different devices that are mainly constrained by memory. For each device, we can therefore only consider models up to a fixed amount of parameters, similarly as depicted in Figure 1. Given that we have a pretrained model that achieves high performance, we want to *compress* this model exploiting redundancies in model parameters. In our application, we use the Stanford Dogs dataset [26] and

rely on the neural network intelligence (NNI; [40]) toolkit for model compression. Pruning consists of several (iterative) steps as well as re-training of the pruned architectures. Choices for the pruner itself, pruner hyperparameters, and hyperparameters controlling retraining are available and must be carefully selected to obtain optimal models (see Supplement F). We consider the number of model parameters as a proxy measure for memory requirement, yielding three overlapping niches for different devices. The pre-trained MobileNetV2 achieves a validation error of 20.25 using around 2.34 million model parameters. We define niches with boundaries corresponding to compression rates (number of parameters after pruning) of 40% to 50%, 40% to 60%, and 40% to 70%. As the QD optimizer, we use BOP-ElitesHB and specify the number of fine-tuning epochs a pruner can use as a fidelity parameter, since fine-tuning after pruning is costly but also strongly influences final performance. After evaluating only 69 configurations (a single BOP-ElitesHB run with $\eta = 3$), we obtain high-performing pruner configurations for each niche, resulting in the performance vs. memory requirement (number of parameters) trade-offs shown in Table 3.

Table 3: Results of using BOP-ElitesHB for model compression of MobileNetV2 on Stanford Dogs.

Niche	Validation Error	# Params (in millions; rounded)	% (# Params _{Baseline})
Niche 1 [0.94, 1.17)	31.20	1.13	48.10%
Niche 2 [0.94, 1.41)	29.07	1.29	54.99%
Niche 3 [0.94, 1.64)	27.76	1.62	68.97%
Baseline	20.25	2.34	100.00%

6 Conclusion

We demonstrated how multi-objective NAS can be formulated as a QDO problem that, contrary to multi-objective NAS, directly corresponds to the actual optimization problem of interest, i.e., finding high-performing architectures in different niches. We have shown how any iterative black box optimization algorithm can be adapted to the QDO setting and proposed three QDO algorithms for NAS, with two of which making use of multifidelity evaluations. In benchmark experiments, we have shown that qdNAS outperforms multi-objective NAS while simultaneously being more efficient. We furthermore illustrated how qdNAS can be used for model compression and how future NAS research can thrive on QDO. QDO is orthogonal to the NAS strategy of an algorithm and can be similarly used to extend, e.g., one-shot NAS methods.

Limitations The framework we describe relies on pre-defined niches, e.g., memory requirements of different devices. If niches are mis-specified or cannot be specified a priori, multi-objective NAS may outperform qdNAS. However, an initial study (see Supplement H) how qdNAS performs in a true multi-objective setting, which would correspond to unknown niches, shows little to no performance degradation depending on the choice of niches. Moreover, we only investigated the performance of qdNAS in the deterministic setting. Additionally, our multifidelity optimizers require niche membership to be unaffected by the multifidelity parameter. Finally, we mainly focused on model-based NAS algorithms that we have extended to the QDO setting.

Broader Impact Our work extends previous research on NAS and therefore inherits its implications on society and individuals such as potential discrimination in resulting models. Moreover, evaluating a large number of architectures is computationally costly and can introduce serious environmental issues. We have shown that qdNAS allows for finding better solutions, while simultaneously being more efficient than multi-objective NAS. As performance estimation is extremely costly in NAS, we believe that this is an important contribution towards reducing resource usage and the CO₂ footprint of NAS.

7 Reproducibility Checklist

1. For all authors...
 - (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
 - (b) Did you describe the limitations of your work? [Yes] See Section 6.
 - (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
 - (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
 - (a) Did you state the full set of assumptions of all theoretical results? [N/A]
 - (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
 - (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., requirements.txt with explicit version), an instructive README with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] The full code for experiments, application, figures and tables can be obtained from the following GitHub repository: https://github.com/slds-lmu/qdo_nas.
 - (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] Raw results are provided via the same GitHub repository.
 - (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] Scripts to generate figures and tables based on raw results are provided via the same GitHub repository.
 - (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]
 - (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] For our benchmark experiments we used NAS-Bench-101 and NAS-Bench-201. Regarding the Additional Experiments and Applications Section, all details are reported in Supplement D, Supplement E and Supplement F.
 - (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] As described in Section 4.
 - (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] See Supplement C, Supplement G and Supplement H.
 - (h) Did you use the same evaluation protocol for the methods being compared? [Yes] As described in Section 4.
 - (i) Did you compare performance over time? [Yes] Anytime performance was assessed with respect to the number of epochs as described in Section 4 or the number of architecture evaluations as described in Section 5.

- (j) Did you perform multiple runs of your experiments and report random seeds? [Yes] All runs of main benchmark experiments were replicated 100 times. Random seeds can be obtained via https://github.com/slids-lmu/qdo_nas.
 - (k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All results include error bars accompanying mean estimates.
 - (l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] We used the tabular NAS-Bench-101 and NAS-Bench-201 benchmarks.
 - (m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] As described in Supplement I.
 - (n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [N/A] No tuning of hyperparameters was performed.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
- (a) If your work uses existing assets, did you cite the creators? [Yes] NAS-Bench-101, NAS-Bench-201, Naszilla, the Once-for-All module, NNI, and the Stanford Dogs dataset are cited appropriately.
 - (b) Did you mention the license of the assets? [Yes] Done in Supplement I.
 - (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide all our code via https://github.com/slids-lmu/qdo_nas.
 - (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All assets used are either released under the Apache-2.0 License or MIT License.
 - (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
- (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
 - (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
 - (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Acknowledgements. The authors of this work take full responsibilities for its content. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8). This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. Paul Kent acknowledges support from EPSRC under grant EP/L015374. This work has been carried out by making use of AI infrastructure hosted and operated by the Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften and funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre (www.lrz.de).

References

- [1] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang. Hardware-aware neural architecture search: Survey and taxonomy. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4322–4329, 2021.
- [2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011.
- [3] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, pages 115–123, 2013.
- [4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [5] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-All: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.
- [6] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-All: Train one network and specialize it for efficient deployment. <https://github.com/mit-han-lab/once-for-all>, 2020.
- [7] A. Cully and J.-B. Mouret. Evolving a behavioral repertoire for a walking robot. *Evolutionary Computation*, 24(1):59–88, 2016.
- [8] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 3460–3468, 2015.
- [9] S. Doncieux, N. Bredeche, L. L. Goff, B. Girard, A. Coninx, O. Sigaud, M. Khamassi, N. Díaz-Rodríguez, D. Filliat, T. Hospedales, A. Eiben, and R. Duro. Dream architecture: A developmental approach to open-ended learning in robotics. *arXiv:2005.06223 [cs.AI]*, 2020.
- [10] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun. DPP-Net: Device-aware progressive search for pareto-optimal neural architectures. In *European Conference on Computer Vision*, 2018.
- [11] X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.
- [12] T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via Lamarckian evolution. In *Proceedings of the International Conference on Learning Representations*, 2019.
- [13] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.
- [14] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.
- [15] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough. Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, 32, 2019.

- [16] M. C. Fontaine, R. Liu, J. Togelius, A. K. Hoover, and S. Nikolaidis. Illuminating mario scenes in the latent space of a generative adversarial network. In *AAAI Conference on Artificial Intelligence*, 2021.
- [17] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv:1807.02811 [stat.ML]*, 2018.
- [18] A. Gaier, A. Asteroth, and J.-B. Mouret. Aerodynamic design exploration through surrogate-assisted illumination. In *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.
- [19] J. Guerrero-Viu, S. Hauns, S. Izquierdo, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M. Lindauer, and F. Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. In *8th ICML Workshop on Automated Machine Learning*, 2021.
- [20] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.
- [21] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523, 2011.
- [22] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.
- [23] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.
- [24] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.
- [25] P. Kent and J. Branke. BOP-Elites, a Bayesian optimisation algorithm for quality-diversity search. *arXiv:2005.04320 [math.OA]*, 2020.
- [26] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition*, 2011.
- [27] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations*, 2017.
- [28] J. Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.
- [29] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.
- [30] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 211–218, 2011.

- [31] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *International Conference on Learning Representations*, 2017.
- [32] E. Liberis, Ł. Dudziak, and N. D. Lane. μ nas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 70–79, 2021.
- [33] M. Lindauer, K. Eggenberger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhopf, R. Sass, and F. Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.
- [34] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the International Conference on Learning Representations*, 2017.
- [35] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.
- [36] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti. NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *European Conference on Computer Vision*, pages 35–51, 2020.
- [37] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf. NSGA-Net: A multi-objective genetic algorithm for neural architecture search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427, 2019.
- [38] B. Lyu, H. Yuan, L. Lu, and Y. Zhang. Resource-constrained neural architecture search on edge devices. *IEEE Transactions on Network Science and Engineering*, 9(1):134–142, 2021.
- [39] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In *ICML Workshop on Automatic Machine Learning*, 2016.
- [40] Microsoft. Neural Network Intelligence. <https://github.com/microsoft/nni>, 2021.
- [41] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.
- [42] J.-B. Mouret and Jeff. Clune. Illuminating search spaces by mapping elites. *arXiv:1504.04909 [cs.AI]*, 2015.
- [43] M. Pearce and J. Branke. Continuous multi-task Bayesian optimisation with correlation. *European Journal of Operational Research*, 270(3):1074–1085, 2018.
- [44] D. Perez-Liebana, C. Guerrero-Romero, A. Dockhorn, D. Jeurissen, and L. Xu. Generating diverse and competitive play-styles for strategy games. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021.
- [45] H. Pham, M. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4095–4104, 2018.
- [46] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019.

- [47] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2902–2911, 2017.
- [48] D. Salinas, V. Perrone, O. Cruchant, and C. Archambeau. A multi-objective perspective on jointly tuning hardware and hyperparameters. In *2nd Workshop on Neural Architecture Search at ICLR 2021*, 2021.
- [49] R. Schmucker, M. Donini, V. Perrone, M. B. Zafar, and C. Archambeau. Multi-objective multi-fidelity hyperparameter optimization with application to fairness. In *NeurIPS Workshop on Meta-Learning*, 2020.
- [50] R. Schmucker, M. Donini, M. B. Zafar, D. Salinas, and C. Archambeau. Multi-objective asynchronous successive halving. *arXiv:2106.12639 [stat.ML]*, 2021.
- [51] L. Schneider, F. Pfisterer, M. Binder, and B. Bischl. Mutation is all you need. In *8th ICML Workshop on Automated Machine Learning*, 2021.
- [52] L. Schneider, F. Pfisterer, J. Thomas, and B. Bischl. A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models. *arXiv:2204.14061 [cs.LG]*, 2022.
- [53] C. Schorn, T. Elsken, S. Vogel, A. Runge, A. Guntoro, and G. Ascheid. Automated design of error-resilient and hardware-efficient deep neural networks. *Neural Computing and Applications*, 32:18327–18345, 2020.
- [54] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, 2012.
- [55] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu. Single-Path NAS: Designing hardware-efficient ConvNets in less than 4 hours. In U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 481–497, 2020.
- [56] C. White, W. Neiswanger, S. Nolen, and Y. Savani. A study on encodings for neural architecture search. In *Advances in Neural Information Processing Systems*, 2020.
- [57] C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.
- [58] C. White, S. Nolen, and Y. Savani. Exploring the loss landscape in neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 654–664, 2021.
- [59] C. White, A. Zela, B. Ru, Y. Liu, and F. Hutter. How powerful are performance predictors in neural architecture search? In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, 2021.
- [60] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

- [61] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7105–7114, 2019.
- [62] Y. Zhou, S. Ebrahimi, S. Ö. Arık, H. Yu, H. Liu, and G. Diamos. Resource-efficient neural architect. *arXiv:1806.07912 [cs.NE]*, 2018.
- [63] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2017.
- [64] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

A Niches in NAS

In the classical QDO literature, niches are assumed to be pairwise disjoint. This implies that each architecture $A \in \mathcal{A}$ yields feature function values $f_i(A), i \geq 2$ that map to a single niche (or none). In practice, this does not necessarily have to be the case though, as an architecture can belong to multiple niches. For example, when considering memory or latency constraints, a model with lower latency or lower memory requirements can always be used in settings that allow for accommodating slower or larger models. This is illustrated in Figure 4. Note that we index niches in the disjoint scenario in Figure 4 with two indices, to highlight that some niches share the same boundaries on a given feature function (e.g., $N_{1,1}$ and $N_{2,1}$ share the same latency boundaries and only differ with respect to the memory boundaries). In this paper, we mainly investigated the scenario of nested niches. The setting for QDO in the NAS context as described in Figure 1 in the main paper is given by the search of models for deployment on multiple different end-user devices. Similarly, qdNAS can also be applied in the context of searching for models for deployment on a single end-user device, meeting different constraints, e.g., as illustrated in Section 5 (Benchmarks on the MobileNetV3 Search Space) in the main paper. Typically, relevant boundaries of feature functions that form niches naturally arise given the target device(s) and concrete application at hand.

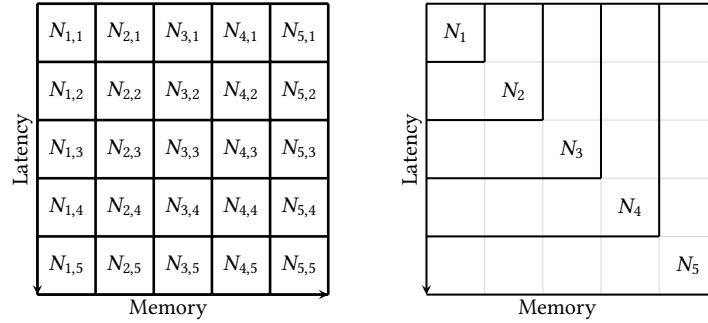


Figure 4: Disjoint (left) and nested (right) niches.

B Optimizers

In this section, we provide additional information on optimizers used throughout this paper. Algorithm 2 illustrates a generic iterative single-objective optimizer in pseudo code.

qdHB Algorithm 3 presents qdHB in pseudo code. qdHB requires only R (maximum fidelity) and η (scaling parameter) as input parameters and proceeds to determine the maximum number of brackets s_{\max} and the approximate total resources B which each bracket is assigned. In each bracket s , the number of configurations n and the fidelity r at which they should be evaluated is calculated and these parameters are used within the SH subroutine. The central step within the SH subroutine is the selection of the $\lfloor n_i/\eta \rfloor$ configurations that should be promoted to the next stage. Here, the $\text{top}_{k,\text{qdo}}$ function (highlighted in grey) works as follows: We iteratively select one of the niches uniformly at random and choose the best configuration within this niche observed so far that has yet not been selected for promotion. This procedure is repeated until $\lfloor n_i/\eta \rfloor$ configurations have been selected in total. If not enough configurations belonging to a specific niche have been observed so far, we choose any configuration uniformly at random over the set of all configurations that have yet to be promoted. Note that feature functions and thereupon derived niche membership are assumed to be unaffected by the multifidelity parameter. Niche membership is determined by the

Algorithm 2: Generic pseudo code for an iterative single-objective optimizer.

Input : $\hat{f}_1, \mathcal{D}_{\text{design}}, n_{\text{total}}$
Result : A^*

```

1  $\mathcal{D} \leftarrow \mathcal{D}_{\text{design}}$ 
2  $A^* \leftarrow \arg \min_{A \in \mathcal{D}} f_1(A)$  # initial incumbent based on archive
3 for  $n \leftarrow 1$  to  $n_{\text{total}}$  do
4   Propose a new candidate  $A^*$ 
5   Evaluate  $y \leftarrow f_1(A^*)$ 
6   if  $y < f_1(A^*)$  then
7      $A^* \leftarrow A^*$  # update incumbent
8   end
9    $\mathcal{D} \leftarrow \mathcal{D} \cup \{A^*, y\}$ 
10 end

```

get_niche_membership function which simply checks for each niche whether feature values of an architecture are within the respective niche boundaries. Moreover, we assume that all evaluations are written into an archive similarly as in Algorithm 1 in the main paper which allows us to return the best configuration per niche as the final result. Note that in practice, evaluating all stages of brackets with the same budget instead of iterating over brackets (like in the original HB implementation) can be more efficient. We use this scheduling variant throughout our benchmark experiments and application study. More details regarding our implementation can be obtained via https://github.com/slds-lmu/qdo_nas.

Algorithm 3: Quality Diversity Hyperband (qdHB).

Input : R, η # maximum fidelity and scaling parameter
Result : Best configuration per niche

```

1  $s_{\text{max}} = \lfloor \log_{\eta}(R) \rfloor, B = (s_{\text{max}} + 1)R$ 
2 for  $s \in \{s_{\text{max}}, s_{\text{max}} - 1, \dots, 0\}$  do
3    $n = \lfloor \frac{B}{R} \frac{\eta^s}{(s+1)} \rfloor, r = R\eta^{-s}$ 
4   # begin SH with  $(n, r)$  inner loop
5    $\mathbf{A} = \text{sample\_configuration}(n)$ 
6    $\mathbf{Z} = \{(f_i(A), \dots, f_k(A)) : A \in \mathbf{A}, i \in \{2, \dots, k\}\}$  # evaluate feature functions
7    $\mathbf{N} = \text{get\_niche\_membership}(\mathbf{A}, \mathbf{Z})$ 
8   for  $i \in \{0, \dots, s\}$  do
9      $n_i = \lfloor n\eta^{-i} \rfloor$ 
10     $r_i = r\eta^i$ 
11     $\mathbf{Y} = \{f_1(A, r_i) : A \in \mathbf{A}\}$  # evaluate objective function
12     $\mathbf{A} = \text{top}_{k\_qdo}(\mathbf{A}, \mathbf{Y}, \mathbf{N}, \lfloor n_i/\eta \rfloor)$ 
13  end
14 end

```

BOP-ElitesHB In Algorithm 4 we describe the sampling procedure (for a single configuration) used in BOP-ElitesHB in pseudo code. In contrast to the original BOHB algorithm, we use random forest as surrogate models, similarly as done in SMAC-HB [33]. Throughout our benchmark experiments and application study we set $\rho = 0$. Furthermore, we employ a variant that directly proposes batches of size n . This can be done by simply sorting all candidate architectures obtained via local mutation of the incumbent architectures of each niche within the acquisition function

optimization step by their EJI values and selecting the top n candidate architectures. Note that surrogate models are fitted on all available data contained in the current archive (this includes the multifidelity parameter) and predictions are obtained with respect to the fidelity parameter set to the current fidelity level. More details regarding our implementation can be obtained via https://github.com/slds-lmu/qdo_nas.

Algorithm 4: Sampling procedure in BOP-ElitesHB.

Input : ρ # fraction of configurations sampled at random
Result: Next configuration to evaluate

```

1 if rand() <  $\rho$  then
2   | return sample_configuration(1)
3 else
4   |  $A^* \leftarrow \arg \max_{A \in \mathcal{A}} \alpha_{\text{EJI}}(A)$  # Equation (1)
5   | return  $A^*$ 
6 end

```

ParEGO* ParEGO [28] is a multi-objective model-based optimizer that at each iteration scalarizes the objective functions differently using the augmented Tchebycheff function. First, the k objectives are normalized and at each iteration a weight vector λ is drawn uniformly at random from the following set of $\binom{s+k-1}{k-1}$ different weight vectors⁴:

$$\left\{ \lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \mid \sum_{i=1}^k \lambda_i = 1 \wedge \lambda_i = \frac{l}{s}, l \in \{0, \dots, s\} \right\}.$$

The scalarization is then obtained via $f_\lambda(A) = \max_{i=1}^k (\lambda_i \cdot f_i(A)) + \gamma \sum_{i=1}^k \lambda_i \cdot f_i(A)$, where γ is a small positive value (in our benchmark experiments we use 0.05). In ParEGO* we use the same truncated path encoding as in BOP-Elites* as well as a random forest surrogate modeling the scalarized objective function. For optimizing the EI, we use a local mutation scheme similarly to the one utilized by BANANAS [57], adapted for the multi-objective setting (conceptually similar to the one proposed by [19]): For each Pareto optimal architecture in the current archive, we obtain candidate architectures via local mutation and out of all these candidates we select the architecture with the largest EI for evaluation. More details regarding our implementation can be obtained via https://github.com/slds-lmu/qdo_nas.

moHB* moHB* [28] is an extension of HB to the multi-objective setting. The optimizer follows the basic HB routine except for the selection mechanism of configurations that should be promoted to the next stage: Configurations are promoted based on non-dominated sorting with hypervolume contribution for tie breaking. For similar approaches, see [48, 49, 50, 19]. In our benchmark experiments we again use a scheduling variant that evaluates all stages of brackets with the same budget instead of iterating over brackets. More details regarding our implementation can be obtained via https://github.com/slds-lmu/qdo_nas.

ParEGOHB ParEGOHB combines BO with moHB* by using the same scalarization as ParEGO*. Instead of selecting configurations at random at the beginning of each moHB* iteration, ParEGOHB proposes candidates that maximize the EI with respect to the scalarized objective. In our benchmark experiments we again set $\rho = 0$ (fraction of configurations sampled uniformly at random) and employ a variant that directly proposes batches of size n . Note that surrogate models are fitted on all available data contained in the current archive (this includes the multifidelity parameter) and predictions are obtained with respect to the fidelity parameter set to the current fidelity level. More details regarding our implementation can be obtained via https://github.com/slds-lmu/qdo_nas.

⁴note that s simply determines the number of different weight vectors

Table 4: Niches and their boundaries used throughout all benchmark experiments.

Benchmark	Dataset	Niches	Niche Boundaries										
			Niche 1	Niche 2	Niche 3	Niche 4	Niche 5	Niche 6	Niche 7	Niche 8	Niche 9	Niche 10	
NAS-Bench-101 # Params	Cifar-10	Small	[0, 5356682)	[0, ∞)	-	-	-	-	-	-	-	-	-
		Medium	[0, 650520)	[0, 1227914)	[0, 1664778)	[0, 3468426)	[0, ∞)	-	-	-	-	-	-
		Large	[0, 650520)	[0, 824848)	[0, 1227914)	[0, 1664778)	[0, 2538506)	[0, 3468426)	[0, 3989898)	[0, 5356682)	[0, 8118666)	[0, ∞)	-
NAS-Bench-201 Latency	Cifar-10	Small	[0, 0.01500044871408)	[0, ∞)	-	-	-	-	-	-	-	-	-
		Medium	[0, 0.00856115)	[0, 0.01030767)	[0, 0.01143533)	[0, 0.01363741)	[0, ∞)	-	-	-	-	-	-
		Large	[0, 0.00856115)	[0, 0.00893427)	[0, 0.01030767)	[0, 0.01143533)	[0, 0.01250159)	[0, 0.01363741)	[0, 0.01429903)	[0, 0.01500044)	[0, 0.01660615)	[0, ∞)	-
NAS-Bench-201 Latency	Cifar-100	Small	[0, 0.0159673188862048)	[0, ∞)	-	-	-	-	-	-	-	-	-
		Medium	[0, 0.00919228)	[0, 0.01138714)	[0, 0.01232998)	[0, 0.01475572)	[0, ∞)	-	-	-	-	-	-
		Large	[0, 0.00919228)	[0, 0.00957457)	[0, 0.01138714)	[0, 0.01232998)	[0, 0.01327515)	[0, 0.01475572)	[0, 0.01534633)	[0, 0.01596732)	[0, 0.01768237)	[0, ∞)	-
MobileNetV3 Latency	ImageNet	Small	[0, 0.014301609992981)	[0, ∞)	-	-	-	-	-	-	-	-	-
		Medium	[0, 0.00767465)	[0, 0.0094483)	[0, 0.01054566)	[0, 0.01271056)	[0, ∞)	-	-	-	-	-	-
		Large	[0, 0.00767465)	[0, 0.00825192)	[0, 0.0094483)	[0, 0.01054566)	[0, 0.01173623)	[0, 0.01271056)	[0, 0.01352221)	[0, 0.01430161)	[0, 0.01595311)	[0, ∞)	-
MobileNetV3 FLOPS	ImageNet	Small	[0, 17.5)	[0, 30)	-	-	-	-	-	-	-	-	-
		Medium	[0, 17)	[0, 20)	[0, 25)	[0, 30)	[0, 35)	-	-	-	-	-	-
		Large	[0, 17)	[0, 19)	[0, 21)	[0, 23)	[0, 25)	[0, 27)	[0, 29)	[0, 31)	[0, 33)	[0, 35)	-
MobileNetV3 Latency × Size	ImageNet	Small	[0, 150)	[0, 400)	-	-	-	-	-	-	-	-	-
		Medium	[0, 150) × [0, 20)	[0, 200) × [0, 20)	[0, 250) × [0, 20)	[0, 300) × [0, 20)	[0, 400) × [0, 20)	-	-	-	-	-	-
		Large	[0, 150) × [0, 20)	[0, 175) × [0, 20)	[0, 200) × [0, 20)	[0, 225) × [0, 20)	[0, 250) × [0, 20)	[0, 275) × [0, 20)	[0, 300) × [0, 20)	[0, 325) × [0, 20)	[0, 350) × [0, 20)	[0, 400) × [0, 20)	-

C Additional Benchmark Details and Results

In this section, we provide additional details and analyses with respect to our main benchmark experiments. Table 4 summarizes all niches and their boundaries used throughout our benchmarks (including the additional ones on the MobileNetV3 search space).

The following results extend the results reported for the main benchmark experiments. Critical differences plots ($\alpha = 0.05$) of optimizer ranks (with respect to final performance) are given in Figure 5. Friedman tests ($\alpha = 0.05$) that were conducted beforehand indicated significant differences in ranks for both the validation ($\chi^2(6) = 53.46, p < 0.001$) and test performance ($\chi^2(6) = 52.14, p < 0.001$). However, note that critical difference plots based on the Nemenyi test are underpowered if only few optimizers are compared on few benchmark problems.

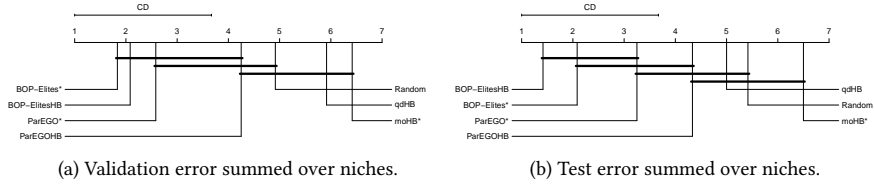


Figure 5: Critical differences plots of the ranks of optimizers.

Figure 6 and Figure 7 show the average best validation and test performance for each niche for each optimizer on each benchmark problem.

Table 5 summarizes results of a four way ANOVA on the average performance (validation error summed over niches) of BOP-ElitesHB, BOP-Elites*, ParEGOHB, ParEGO* and Random after having used half of the total optimization budget. Prior to conducting the ANOVA, we checked the ANOVA assumptions (normal distribution of residuals and homogeneity of variances) and found no violation of assumptions. The factors are given as follows: Problem indicates the benchmark problem (e.g., NAS-Bench-101 on Cifar-10 with small number of niches), multifidelity denotes if the optimizer uses multifidelity (TRUE for BOP-ElitesHB and ParEGOHB), QDO denotes whether the optimizer is a QD optimizer (TRUE for BOP-ElitesHB and BOP-Elites*) and model-based denotes whether the optimizer relies on a surrogate model (TRUE for BOP-ElitesHB, BOP-Elites*, ParEGOHB and ParEGO*). All main effects are significant at an α level of 0.05. We also computed confidence intervals based on Tukey’s Honest Significant Difference method for the estimated differences between factor levels: Multifidelity $-12.45[-18.19 - 6.72]$, QDO $-9.34[-15.08, -3.61]$, model-based $-13.55[-20.57, -6.52]$. Note that the negative sign indicates a decrease in the average validation error summed over niches.

Table 5: Results of a four way ANOVA on the average performance (validation error summed over niches) after having used half of the total optimization budget. Type II sums of squares.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Problem	11	1810569.79	164597.25	1410.16	0.0000
Multifidelity	1	2233.44	2233.44	19.13	0.0001
QDO	1	1293.41	1293.41	11.08	0.0017
Model-Based	1	2466.45	2466.45	21.13	0.0000
Residuals	45	5252.51	116.72		

We conducted a similar ANOVA on the final performance of optimizers (Table 6). Prior to conducting the ANOVA, we checked the ANOVA assumptions (normal distribution of residuals and homogeneity of variances) and found no violation of assumptions. While the effects of QDO

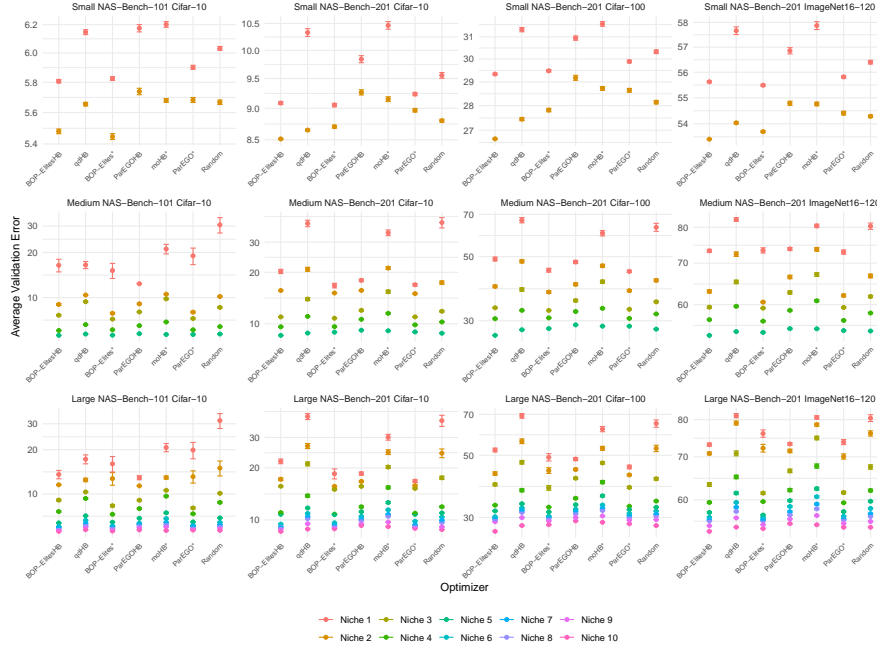


Figure 6: Best solution found in each niche with respect to validation performance. Bars represent standard errors over 100 replications.

and model-based are still significant at an α level of 0.05, the effect of multifidelity no longer is, indicating that full-fidelity optimizer caught up in performance (which is the expected behavior). We again computed confidence intervals based on Tukey’s Honest Significant Difference method for the estimated differences between factor levels: QDO $-6.85[-10.12 - 3.58]$, model-based $-11.08[-15.08, -7.07]$.

Table 6: Results of a four way ANOVA on the average final performance (validation error summed over niches). Type II sums of squares.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Problem	11	1724557.85	156777.99	4130.33	0.0000
Multifidelity	1	97.76	97.76	2.58	0.1155
QDO	1	695.13	695.13	18.31	0.0001
Model-Based	1	1648.94	1648.94	43.44	0.0000
Residuals	45	1708.10	37.96		

We analyzed the ERT of the QD optimizers given the average performance of the respective multi-objective optimizers after half of the optimization budget. For each benchmark problem, we computed the mean validation performance of each multi-objective optimizer after having spent half of its optimization budget and investigated the analogous QD optimizer. We then computed the ratio of ERTs between multi-objective and QD optimizers (see Table 7).

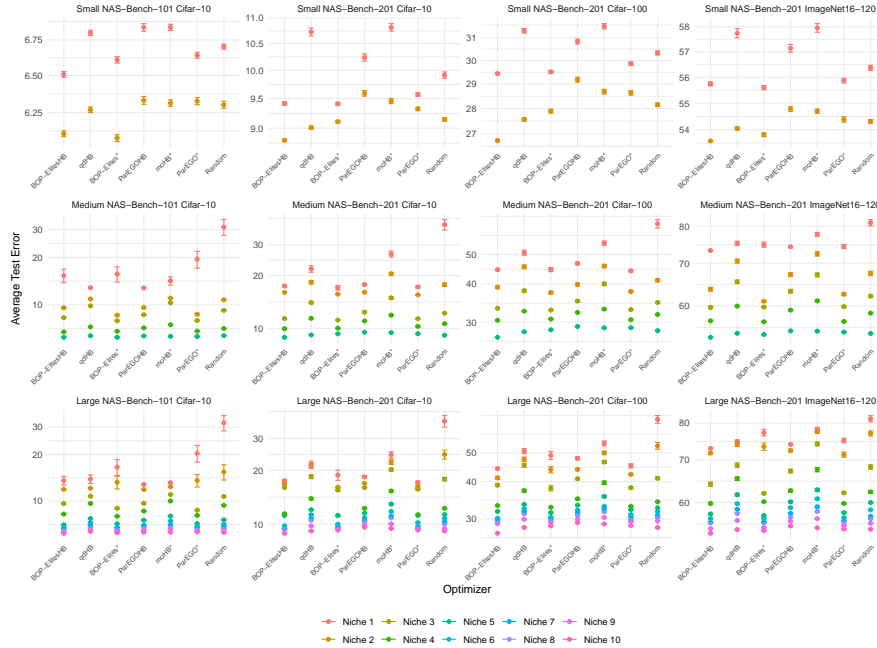


Figure 7: Best solution found in each niche with respect to test performance. Bars represent standard errors over 100 replications.

D Details on Benchmarks on the MobileNetV3 Search Space

In this section, we provide additional details regarding our benchmarks on the MobileNetV3 Search Space. We use of `a_mbv3_d234_e346_k357_w1.2` as a pretrained supernet and rely on accuracy predictors and latency/FLOPS look-up tables as provided by [6]. The search space of architectures is the same as used in [5]. For the model-based optimizers we employ the following encoding of architectures: Given an architecture, we encode each layer in the neural network into a one-hot vector based on its kernel size and expand ratio and we assign zero vectors to layers that are skipped. Besides, we have an additional one-hot vector that represents the input image size. We concatenate these vectors into a large vector that represents the whole neural network architecture and input image size. This is the same encoding as used by [5]. Acquisition function optimization is performed by sampling 1000 architectures uniformly at random.

E Details on Making Once-for-All Even More Efficient

In this section, we provide additional details regarding replacing regularized evolution with MAP-Elites within Once-for-All. We use of `a_mbv3_d234_e346_k357_w1.2` as a pretrained supernet and rely on accuracy predictors and latency look-up tables as provided by [6]. Seven niches were defined via the following latency constraints (in ms): $[0, 15)$, $[0, 18)$, $[0, 21)$, $[0, 24)$, $[0, 27)$, $[0, 30)$, $[0, 33)$. Regularized evolution is run with an initial population of size 100 for 71 generations⁵ resulting in

⁵this is exactly $\lceil (50100 - 7 \cdot 100) / (7 * 100) \rceil$ with 50100 being the budget MAP-Elites is allowed to use

Table 7: ERT ratios of multi-objective and QD optimizers to reach the average performance (after half of the optimization budget) of the respective multi-objective optimizer.

Benchmark	Dataset	Niches	ERT Ratio		
			ParEGOHB/ BOP-ElitesHB	moHB*/ qdHB	ParEGO*/ BOP-Elites*
NAS-Bench-101	Cifar-10	Small	3.94	1.19	1.99
		Medium	0.76	1.43	1.85
		Large	1.47	1.20	1.58
	Cifar-10	Small	4.31	1.96	1.45
		Medium	0.94	0.73	1.34
		Large	1.04	0.72	1.35
NAS-Bench-201	Cifar-100	Small	4.82	1.77	1.46
		Medium	1.35	0.67	1.42
		Large	1.57	0.78	0.98
	ImageNet16-120	Small	4.30	1.20	1.73
		Medium	2.31	0.93	1.14
		Large	2.11	1.15	0.96

7200 architecture evaluations per latency constraint and 50400 architecture evaluations in total. We use a mutation probability of 0.1, a mutation ratio of 0.5 and a parent ratio of 0.25. MAP-Elites searches for optimal architectures jointly for the seven niches and is configured to use a population of size 100 and is run for 500 generations, resulting in 50100 architecture evaluations in total. The number of generations for each regularized evolution run and the MAP-Elites run were chosen in a way so that the total number of architecture evaluations is roughly the same for both methods. We again use a mutation probability of 0.1. Note that the basic MAP-Elites (as used by us) does not employ any kind of crossover. We visualize the best validation error obtained for each niche in Figure 8 (left). MAP-Elites outperforms regularized evolution in almost every niche, making this variant of Once-for-All even more efficient. In the scenario of using Once-for-All for new devices, look-up tables do not generalize and the need for using as few as possible architecture evaluations is of central importance. To illustrate how MAP-Elites compares to regularized evolution in this scenario, we reran the experiments above but this time we used a population of size 50 and 100 generations for MAP-Elites (and therefore 14 generations for each run of regularized evolution). Results are illustrated in Figure 8 (right). Again, MAP-Elites generally outperforms regularized evolution.

F Details on Applying qdNAS to Model Compression

In this section, we provide additional details regarding our application of qdNAS to model compression. BOP-ElitesHB was slightly modified due to the natural tabular representation of the search space. Instead of using a truncated path encoding we simply use the tabular representation of parameters. To optimize the EJIE during the acquisition function optimization step we employ a simple Random Search, sampling 10000 configurations uniformly at random and proposing the configuration with the largest EJIE. Table 8 shows the search space used for tuning NNI pruners on MobileNetV2.

G Analyzing the Effect of the Choice of the Surrogate Model and Acquisition Function Optimizer

In this section, we present results of a small ablation study regarding the effect of the choice of the surrogate model and acquisition function optimizer. In the main benchmark experiments, we observed that our qdNAS optimizers sometimes fail to find any architecture belonging to a certain

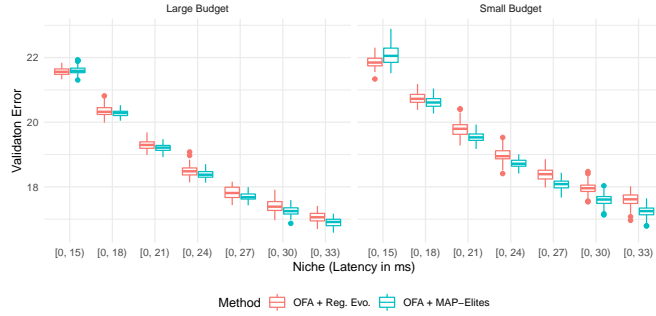


Figure 8: Regularized evolution vs. MAP-Elites within Once-for-All. Left: Large budget of total architecture evaluations. Right: Small budget of total architecture evaluations. Boxplots are based on 100 replications.

Table 8: Search space for NNI pruners on MobileNetV2.

Hyperparameter	Type	Range	Info
pruning_mode	categorical	{conv0, conv1, conv2, conv1andconv2, all}	
pruner_name	categorical	{l1, l2, slim, agp, fpgm, mean_activation, apoz, taylorfo}	
sparsity	continuous	[0.4, 0.7]	
agp_pruning_alg	categorical	{l1, l2, slim, fpgm, mean_activation, apoz, taylorfo}	
agp_n_iters	integer	[1, 100]	
agp_n_epochs_per_iter	integer	[1, 10]	
slim_sparsifying_epochs	integer	[1, 30]	
speed_up	boolean	{TRUE, FALSE}	
finetune_epochs	integer	[1, 27]	fidelity
learning_rate	continuous	[1e-06, 0.01]	log
weight_decay	continuous	[0, 0.1]	
kd	boolean	{TRUE, FALSE}	
alpha	continuous	[0, 1]	
temp	continuous	[0, 100]	

agp_pruning_alg, agp_n_iters*, and agp_n_epochs_per_iter* depend on pruner_name* being agp*. slim_sparsifying_epochs* depends on pruner_name* being slim*. alpha* and temp* depend on kd* being TRUE*. log* in the Info column indicates that this parameter is optimized on a logarithmic scale.

niche (even after having used all available budget). This was predominantly the case for the very small niches in the medium and large number of niches settings (i.e., Niche 1, 2 or 3). Figure 9 shows the relative frequency of niches missed by optimizers (over 100 replications). Note that for the small number of niches settings, relative frequencies are all zero and therefore omitted. In general, model-based multifidelity variants perform better than the full-fidelity optimizers and QD optimizers sometimes perform worse than multi-objective optimizers.

We hypothesized that this could be caused by the choice of the surrogate model used for the feature functions: A random forest cannot properly extrapolate values outside the training set and therefore, if the initial design does not contain an architecture for a certain niche, the optimizer may fail to explore relevant regions in the feature space. We therefore conducted a small ablation study on the NAS-Bench-101 Cifar-10 medium number of niches benchmark problem. BOP-Elites* was configured to either use a random forest (as before) or an ensemble of feed-forward neural networks⁶ (as used by BANANAS [57]) as a surrogate model for the feature function. Moreover, we

⁶with an ensemble size of five networks

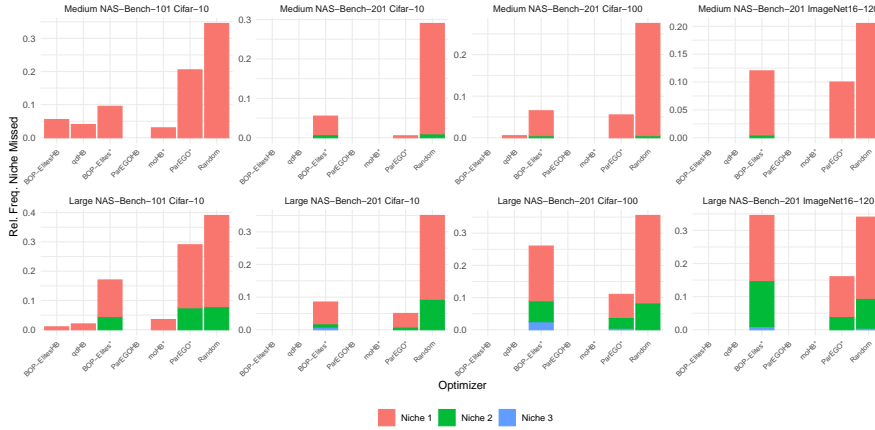


Figure 9: Relative frequency of niches missed by optimizers over 100 replications. For the small number of niches settings, relative frequencies are all zero and therefore omitted.

varied the acquisition function optimizer between a local mutation (as before) or a simple Random Search (generating the same number of candidate architectures but sampling them uniformly at random using adjacency matrix encoding). Optimizers were given a budget of 100 full architecture evaluations and runs were replicated 30 times. Figure 10 shows the anytime performance of these BOP-Elites* variants. We observe that switching to an ensemble of neural networks as a surrogate model for the feature function results in a performance boost which can be explained by the fact that this BOP-Elites* variant no longer struggles with finding solutions in the smallest niche. The relative frequencies of a solution for Niche 1 being missing are: 26.67% for the random forest + Random Search, 16.67% for the random forest + mutation, 3.33% for the ensemble of neural networks + Random Search, and 3.33% for the ensemble of neural networks + mutation. Regarding the other niches, a solution is always found. Results also suggest that the choice of the acquisition function optimizer may be more important in case of using a random forest as a surrogate model for the feature function.

H Judging Quality Diversity Solutions by Means of Multi-Objective Performance Indicators

In this section, we analyze the performance of our qdNAS optimizers in the context of a multi-objective optimization setting. As an example, suppose that niches were mis-specified and the actual solutions (best architecture found for each niche) returned by the QD optimizers are no longer of interest. We still could ask the question of how well QDO performs in solving the multi-objective optimization problem. To answer this question, we evaluate the final performance of all optimizers compared in Section 4 by using multi-objective performance indicators. Figure 11 shows the average Hypervolume Indicator (the difference in hypervolume between the resulting Pareto front approximation of an optimizer for a given run and the best Pareto front approximation found over all optimizers and replications). For these computations, the feature function was transformed to the logarithmic scale for the NAS-Bench-101 problems. As nadir points we used $(100, \log(49979275))'$ for the NAS-Bench-101 problems and $(100, 0.0283)'$ for the NAS-Bench-201 problems obtained by taking the theoretical worst validation error of 100 and feature function upper limits as found in the tabular benchmarks (plus some additional small numerical tolerance).

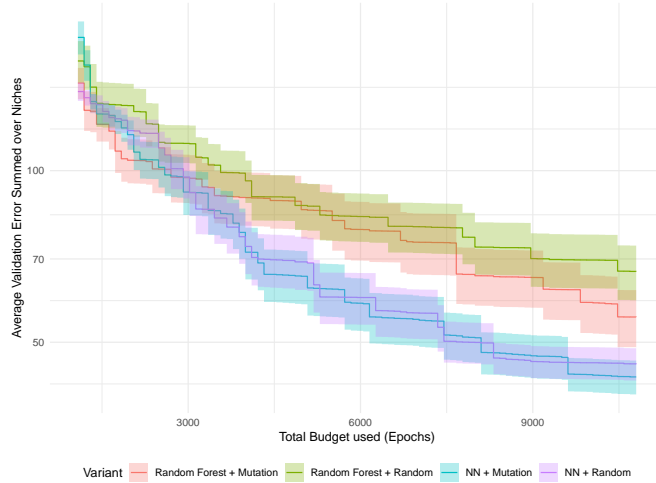


Figure 10: Anytime performance of BOP-Elites* variants configured to either use a random forest or an ensemble of neural networks as a surrogate model for the feature function crossed with either using a local mutation or a Random Search as acquisition function optimizer. NAS-Bench-101 Cifar-10 medium number of niches benchmark problem. Ribbons represent standard errors over 30 replications. x-axis starts after 10 full-fidelity evaluations.

Note that for all optimizers which are not QD optimizers, results with respect to the different number of niches settings (small vs. medium vs. large) are only statistical replications because these optimizers are not aware of the niches. We observe that ParEGOHB and ParEGO* perform well but BOP-ElitesHB also shows good performance in the medium and large number of niches settings. This is the expected behavior, as the number and nature of the niches directly corresponds to the ability of qdNAS optimizers to search along the whole Pareto front, i.e., in the small number of niches settings, qdNAS optimizers have no intention to explore.

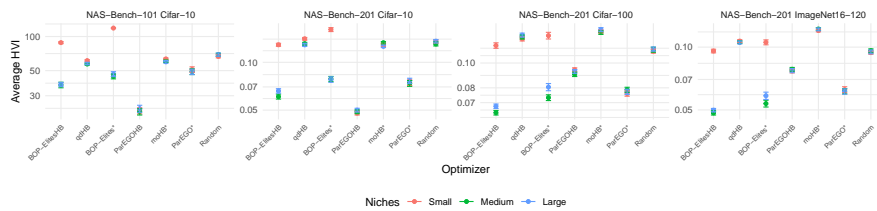


Figure 11: Average Hypervolume Indicator. Bars represent standard errors over 100 replications.

Critical differences plots ($\alpha = 0.05$) of optimizer ranks (with respect to the Hypervolume Indicator) are given in Figure 12. A Friedman test ($\alpha = 0.05$) that was conducted beforehand indicated significant differences in ranks ($\chi^2(6) = 41.61, p < 0.001$). Again, note that critical difference plots based on the Nemenyi test are underpowered if only few optimizers are compared on few benchmark problems.

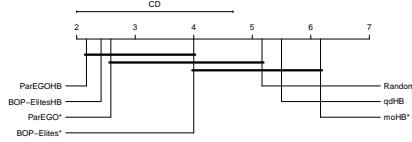


Figure 12: Critical differences plot of the ranks of optimizers with respect to the Hypervolume Indicator.

In Figure 13 we plot the average Pareto front (over 100 replications) for BOP-Elites*, ParEGO* and Random. The average Pareto fronts of BOP-Elites* and ParEGO* are relatively similar, except for the small number of niches settings, where ParEGO* has a clear advantage. Summarizing, qdNAS optimizers can also perform well in a multi-objective optimization setting, but their performance strongly depends on the number and nature of niches.

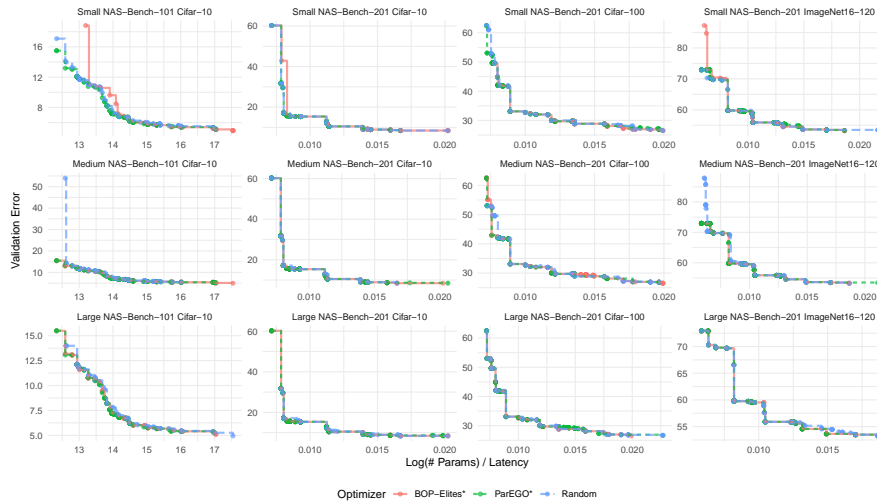


Figure 13: Average Pareto front (over 100 replications) for BOP-Elites*, ParEGO* and Random.

I Technical Details

Benchmark experiments were run on NAS-Bench-101 (Apache-2.0 License) [61] and NAS-Bench-201 (MIT License) [11]. More precisely, we used the `nasbench_full.tfrecord` data for NAS-Bench-101 and the `NAS-Bench-201-v1_1-096897.pth` data for NAS-Bench-201. Parts of our code rely on code released in Naszilla (Apache-2.0 License) [56, 57, 58]. For our benchmarks on the MobileNetV3 search space we used the Once-for-All module [6] released under the MIT License. We rely on `ofa_mbv3_d234_e346_k357_w1.2` as a pretrained supernet and accuracy predictors and resource usage look-up tables as provided by [6]. NNI is released under the MIT License [40]. Stanford Dogs is released under the MIT License [26]. Figure 1 in the main paper has been designed using resources from `Flaticon.com`. Benchmark experiments were run on Intel Xeon E5-2697 instances taking around 939 CPU hours (benchmarks and ablation studies). The model compression application was performed on an NVIDIA DGX A100 instance taking around 3 GPU

days. Total emissions are estimated to be an equivalent of 72.30 kg CO₂. All our code is available at https://github.com/slds-lmu/qdo_nas.

5. Algorithmic Innovations Informed Through Benchmarking

Benchmarking in HPO allows for tracking scientific progress over time through empirical validation. Agreed-upon datasets and tasks, evaluation metrics, and protocols enable fair and standardized comparisons of algorithms and drive innovation. Ablation studies can illustrate why certain algorithms excel or fail and indicate how we can improve them. All of this, however, requires the existence of “good” benchmarks. At the time of writing this thesis and the contributed papers, there was a clear lack of established benchmarks for HPO that are easy to use, reproducible and representative¹. For example, (Feurer and Hutter, 2019) point out that existing benchmarks such as HPOlib (Eggenberger et al., 2013) had not gained similar traction as the COCO platform (Hansen et al., 2021) containing the synthetic BBOB functions (Hansen et al., 2009) used as a benchmarking suite for yearly challenges of the black-box optimization community and called for the development of unified, easy to use and reproducible HPO benchmarking suites. As real-world HPO, i.e., evaluating HPCs via a resampling, is costly, efficient HPO benchmarking naturally focuses on tabular benchmarks (which became especially popular in the field of NAS, see, e.g., Ying et al. 2019; Dong and Yang 2020) or surrogate benchmarks (Eggenberger et al., 2015; Zela et al., 2022). However, at the time of writing this thesis, no benchmarking suite provided unified access to an efficient, large-scale HPO benchmark collection that is widely applicable due to diverse search spaces and not only allows for vanilla single-objective HPO but also multi-fidelity and multi-objective HPO. Moreover, performing automated AC (Hoos, 2012b; Lindauer et al., 2019) of HPO algorithms or designing algorithms via programming by optimization (Hoos, 2012a) requires the availability of a benchmarking suite with representative train and test splits to evaluate automated design choices in an unbiased manner.

In the first contributing article *YAHPO Gym - An efficient multi-objective multi-fidelity benchmark for hyperparameter optimization* we answer the question of how can we design efficient, scalable and representative benchmarks for HPO. With YAHPO Gym, we introduce a surrogate-based benchmarking suite for HPO providing over 700 potentially multi-fidelity and multi-objective HPO benchmark problems across 14 scenarios. Scenarios are defined by a learning algorithm and search space and instances of a scenario vary by concrete datasets and performance metrics. YAHPO Gym is easy to use, requires only a few dependencies and is highly efficient by compressing surrogate models as ONNX binaries making it portable and also easily integrable in other benchmarking suites. We demonstrate that surrogate benchmarks generally allow for more faithful benchmarking of HPO algorithms compared to tabular benchmarks, as long as the surrogate model quality is reasonably high. YAHPO Gym further provides two subsets of selected benchmarking suites suitable for single-objective and multi-objective benchmarking on which we demonstrate the effectiveness of recent algorithms.

¹We want to note that HPO benchmarking suites such as HPOBench (Eggenberger et al., 2021) and HPO-B (Pineda Arango et al., 2021) all became available around the same time as YAHPO Gym which we present as a contribution in this thesis.

In the second contributing article *A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models* we rely on YAHPO Gym to introduce twelve quality diversity optimization benchmark problems based on HPO focusing on interpretability vs. performance, and resource efficiency vs. performance trade-offs. In contrast to a multi-objective optimization setting, interpretability and resource efficiency are treated as secondary objectives with the goal of obtaining diverse solutions that all perform well. By benchmarking established quality diversity optimization algorithms we demonstrate that HPO problems can be formulated as interesting quality diversity optimization problems, bridging a gap between two research communities.

In the third contributing article *Automated benchmark-driven design and explanation of hyperparameter optimizers* we make use of YAHPO Gym as a benchmarking suite to automatically configure a new multi-fidelity HPO algorithm via BO on a meta-level. This is done by optimizing over a search space of building blocks of surrogate model-assisted HB algorithms. The search space includes, for example, parameters such as the batch size, fidelity scaling factor, and survival rate (disentangling the halving parameter of SH and HB based algorithms), sampling method and distribution, and the surrogate model. On a training set of instances, BO identifies a well-performing multi-fidelity algorithm that performs competitively with or better than existing methods and generalizes to a test set of instances. Moreover, we observe that simple configurations (e.g., using an equal batch-size schedule instead of an SH schedule) often perform as well as or better than complicated ones. In an ablation analysis, we further observe that surrogate-based filtering of configurations is useful and again that simple approaches, such as a k -nearest neighbor surrogate model can perform well. Overall, this contribution demonstrates how we can systematically design better HPO algorithms through benchmarking rather than trial and error.

In the fourth contributing article *Mutation is all you need* we present an ablation study of BANANAS (White et al., 2021a), a state-of-the-art NAS method embedded within BO. By systematically varying the choice of architecture representation, surrogate model, acquisition function, and acquisition function optimizer we demonstrate that the strong performance of BANANAS mostly stems from the choice of its acquisition function optimizer which performs a local mutation of the best-performing architecture found so far. In this sense, BANANAS performs a model-guided local search with the local search being the dominant factor for driving its state-of-the-art performance.

In the fifth contributing article *Neural networks as black-box benchmark functions optimized for exploratory landscape features* we shift the focus to general black-box benchmarking functions. For traditional synthetic benchmarking functions such as the BBOB functions, ELA has shown to be useful for telling functions apart as well as for algorithm configuration and selection. To answer the question of how we can automatically generate synthetic functions that match a given ELA feature vector, we rely on a two-step procedure. First, we generate a point cloud and minimize its distance in ELA feature space to a given target vector via CMA-ES. Second, we train a surrogate model, a simple feed-forward neural network, on the optimized point cloud to generate a smooth function that retains the desired ELA features. We demonstrate that this method allows for both interpolation within known benchmark spaces and extrapolation to generate new problems. Moreover, algorithm performance on our regenerated set of BBOB functions closely resembles algorithm performance on the original BBOB function instances, demonstrating the practical usability of our method for generating functions.

5.1. YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization

Contributing article:

F. Pfisterer*, L. Schneider*, J. Moosbauer, M. Binder, and B. Bischl. YAHPO Gym - An efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, pages 3/1–39, 2022. <https://proceedings.mlr.press/v188/pfisterer22a.html>.

Copyright information:

This article is licensed under the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).

Author contributions:

Florian Pfisterer and Lennart Schneider share the first authorship. Their overall contributions can be described as follows. The core idea for the system (YAHPO Gym) originated from Florian Pfisterer who also developed the initial code and the first prototype of the system. Florian Pfisterer re-implemented the underlying software with the help of Lennart Schneider. Florian Pfisterer, Lennart Schneider, and Martin Binder collected samples from relevant benchmarks and performance datasets. Lennart Schneider contributed several improvements to software, stability, and functionality as well as automated tuning. Martin Binder, Julia Moosbauer, and Bernd Bischl advised throughout this process. Lennart Schneider and Florian Pfisterer jointly developed the experiments, which were executed and analyzed by Lennart Schneider who also contributed implementations of relevant baseline algorithms. Florian Pfisterer and Lennart Schneider jointly authored the resulting manuscript with input and improvements by Martin Binder, Julia Moosbauer, and Bernd Bischl.

Supplementary material available at:

- Python and R package: https://github.com/slds-lmu/yahpo_gym
- Code: https://github.com/slds-lmu/yahpo_exps
- Data: https://github.com/slds-lmu/yahpo_data
- arXiv version: <https://arxiv.org/abs/2109.03670>

YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization

Florian Pfisterer^{1,2} Lennart Schneider^{1,2} Julia Moosbauer¹ Martin Binder¹ Bernd Bischl¹

¹Department of Statistics, LMU Munich, Germany

²Equal contributions

Abstract When developing and analyzing new hyperparameter optimization methods, it is vital to empirically evaluate and compare them on well-curated benchmark suites. In this work, we propose a new set of challenging and relevant benchmark problems motivated by desirable properties and requirements for such benchmarks. Our new surrogate-based benchmark collection consists of 14 scenarios that in total constitute over 700 multi-fidelity hyperparameter optimization problems, which all enable multi-objective hyperparameter optimization. Furthermore, we empirically compare surrogate-based benchmarks to the more widely-used tabular benchmarks, and demonstrate that the latter may produce unfaithful results regarding the performance ranking of HPO methods. We examine and compare our benchmark collection with respect to defined requirements and propose a single-objective as well as a multi-objective benchmark suite on which we compare 7 single-objective and 7 multi-objective optimizers in a benchmark experiment. Our software is available at [https://github.com/slds-lmu/yahpo_gym].

1 Introduction

Hyperparameter optimization (HPO) of machine learning (ML) models is a crucial step for achieving good predictive performance [43]. Over the last ten years, a large and still growing set of HPO tuning methods based on different principles has been developed [31, 66, 38]. A particularly interesting development are multi-fidelity methods, which make use of relatively cheap approximations of a given true objective, thereby achieving good performance relatively quickly [44, 21, 35], as well as multi-objective methods, which allow for simultaneous optimization of multiple objectives [40]. While different HPO methods found considerable adoption in practice, it is by no means clear which method performs best under which circumstances. In order to investigate this, it is necessary to evaluate these methods on testbeds that are ideally *i)* highly efficient, *ii)* include a sufficient amount of representative and diverse benchmark instances and *iii)* are easy to set up and integrate with different optimizer APIs. Furthermore, benchmarks have found use in *meta-learning* [70, 74, 59] and *meta-optimization* [49, 53]. In those settings, a larger number of potentially relevant optimization problems is required in order to obtain results that generalize beyond the set of (meta-)training instances. Simultaneously, those applications require a large number of evaluations that make obtaining real evaluations prohibitively expensive, indicating a need for benchmarks that are cheap to query.

Several benchmarks that aim to address this, each of which are collections of multiple benchmark instances, have been proposed [69, 15, 60, 19]. Benchmark instances can be classified into four categories: (i) synthetic functions, (ii) benchmarks incorporating *real* evaluations, (iii) *tabular* benchmarks based on pre-evaluated grid points, and (iv) *surrogate* benchmarks making use of meta-models that approximate the relationship between configurations and performance metrics. Each category has various advantages and drawbacks. Synthetic functions can be evaluated quickly but are often not representative for the type of problems encountered in practice; real evaluations on the other hand are often prohibitively expensive, especially in the context of larger

benchmarks and neural architecture search (NAS). Tabular benchmarks, while cheap to evaluate, rely on a pre-defined grid which changes the optimization problem and can potentially lead to biases. Surrogate benchmarks are also cheap to query but require high quality surrogates in order to avoid introducing bias. While benchmark suites have found some use in scientific publications, they are not used ubiquitously. This lack of permeation – and consequently the lack of a standard test bed – can result in researchers choosing benchmark problems that favor their own method, leading to the publication of biased results. The problem of *cherry picking*, also termed *rigging the lottery* [14], can be ameliorated through the use of standardized testing infrastructure along with a detailed definition of evaluation criteria that are widely adapted.

We therefore observe a clear need for benchmark libraries that provide unified interfaces to a variety of cheap to evaluate, realistic, and practically relevant benchmarking problems that are defined across diverse search spaces. In this work, we propose *YAHPO Gym*, a *surrogate-based* benchmark library including a collection of over 700 benchmark instances defined across 14 *scenarios*. Scenarios are comprised of evaluations of one given machine learning algorithm on different datasets (= instances) and therefore share the same search space and performance metrics. It contains a versioned set of surrogate models that allow for *multi-fidelity* evaluations of *multiple objectives*. Our library is licensed under the *Apache 2.0* license and can be freely used and extended by the community. Usage and available functionality is extensively documented¹.

Contributions: We introduce YAHPO Gym, a surrogate-based benchmark for machine-learning HPO. We conceptually demonstrate that tabular benchmarks may induce bias in performance estimation and ranking of HPO methods, and that this happens to a lesser degree with surrogate benchmarks. We argue that our surrogate benchmark YAHPO Gym meets all desiderata for a good benchmark, providing faithful results, fast evaluation, relevant problems and realistic objective landscapes both on local as well as global scales. In order to demonstrate this, we conduct an extensive evaluation of the proposed surrogates indicating that our surrogate models indeed provide high quality approximations. We propose two benchmark suites for *single-objective* and *multi-objective* evaluation comprised of a subset of our instances and demonstrate how they can be used with YAHPO Gym in a *multi-fidelity* and a *multi-objective* optimization benchmark.

2 Related Work

Several efforts to provide unified testbeds for black-box optimization exist. For general purpose black-box optimization, COCO [29] provides a collection of various synthetic black-box benchmark functions, while *kurobako* [56] is a collection of various general black-box optimizers and benchmark problems. Similarly, *Bayesmark* [69] includes several benchmarks for Bayesian Optimization on real problems and *LassoBench* [64] provides a benchmark for high-dimensional optimization problems. *HPOlib* [15] was one of the first to propose a common test bed for empirically assessing the performance of HPO methods. It provides a common API to access synthetic test functions, real-world HPO problems, tabular benchmarks as well as some surrogate benchmarks and found use in empirical benchmark studies [6]. Its successor *HPOBench* [19] offers similar capabilities, focussing on reproducible containerized benchmarks. It offers 12 benchmark scenarios and more than 100 test instances. Recently, [60] introduced *HPO-B*, a large-scale reproducible (tabular) benchmark for black-box HPO based on OpenML [71]. *HPO-B*² relies on 16 search spaces that were evaluated sparsely on 101 datasets. *PROFET* [37] in contrast is not based on real datasets but uses a generative meta-model to generate synthetic but realistic benchmark instances. In the past, tabular benchmarks have been used frequently to speed up experiments in the context of HPO [66, 23, 72, 22] and NAS (c.f. [50]). Eggensperger et al. [17] compared

¹Documentation and data are available at https://github.com/slds-lmu/yahpo_gym.

²We consider the published v2 version for comparison. Surrogates are only available in the v3 version.

Table 1: Comparison of HPO Benchmark Suites.

Suite	Types	#Collections	#HPs	MF	MO	TF	Async	H	Time [†]	Memory [†]
YAHPO Gym	S	14	2-38	✓	✓	✓	(-)	✓	0.4*s	0.1 GB
HPOBench	R/T/S	12	4-26	✓	✓	(-)	-	(-)	12.2s	0.2 GB
HPO-B (v2)	T/(S)	16	2-18	-	-	✓	-	-	18.8s	3.7 GB

MF: Multi-fidelity; MO: Multi-objective; TF: Transfer-HPO; Async: Asynchronous evaluation; H: hierarchical search spaces.

✓: fully supported; (-): partially supported; -: not supported; R/T/S:real/tabular/surrogate.

[†]: Runtime and memory footprint for 300 iterations of Random Search on an SVM instance. *: allowing for batched evaluation, YAHPO Gym takes only 0.13s.

different instance surrogate models for 9 different HPO problems and concluded that the results of benchmarks run on surrogate models generally closely mimic those of benchmarks using the actual evaluations that they are derived from, if performance measures of the surrogate models indicate that they predict the underlying objective values sufficiently well (cross-validated Spearman’s ρ between 0.9 and 1 [17]). Similar observations have been made in the context of algorithm configuration [18] and NAS [65].

We compare YAHPO Gym with the recently published benchmarks HPOBench [19] and HPO-B [60] in Table 1. Our library relies on high quality surrogates that allow for *multi-fidelity* as well as *multi-objective* evaluation. While existing benchmark suites could in principle be used to construct multi-objective benchmarks, they do not offer full support: HPOBench contains only few instances that allow evaluating multiple metrics and offers no unified API to query those, while HPO-B does not support multiple objectives at all. Furthermore, neither propose a concrete evaluation protocol, opening up a multiplicity of (benchmark) design choices which can lead to inconclusive results (c.f. [55]). Instead of relying on *containerization* to allow for portability, our library relies on neural network surrogates compressed using ONNX [3], allowing for reproducibility and portability while simultaneously being extremely fast and efficient due to minimal overhead. This is demonstrated in a small experiment where we measure runtime and memory consumption for evaluating 300 random configurations on SVM search spaces also shown in Table 1, demonstrating that our software is more time and memory efficient. See details in Supplement B.2. While YAHPO Gym provides the flexibility to design and execute any subset of the provided benchmarks, we also propose two fully specified testbeds for single- and multi-objective optimization that were specifically selected to cover a diverse set of relevant instances while being less extensive. See details in Supplement E.2 and Supplement E.3.

3 Background

3.1 Hyperparameter Optimization

An ML *learner* or *inducer* \mathcal{I} configured by hyperparameters $\lambda \in \Lambda$ maps a dataset $\mathcal{D} \in \mathbb{D}$ to a model \hat{f} , i.e., $\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}, (\mathcal{D}, \lambda) \mapsto \hat{f}$. HPO methods for ML aim to identify a well-performing hyperparameter configuration (HPC) $\lambda \in \tilde{\Lambda}$ for \mathcal{I}_λ [10]. Typically, the considered search space $\tilde{\Lambda} \subset \Lambda$ is a subspace of the set of all possible HPCs: $\tilde{\Lambda} = \tilde{\Lambda}_1 \times \tilde{\Lambda}_2 \times \dots \times \tilde{\Lambda}_d$, where $\tilde{\Lambda}_i$ is a bounded subset of the domain of the i -th hyperparameter Λ_i . This $\tilde{\Lambda}_i$ can be either real, integer, or category valued, and the search space can contain dependent hyperparameters, leading to a possibly hierarchical search space. We formally define the (potentially multi-objective) HPO problem as:

$$\lambda^* \in \arg \min_{\lambda \in \tilde{\Lambda}} c(\lambda), \quad \text{with} \quad c : \tilde{\Lambda} \rightarrow \mathbb{R}^m, \quad (1)$$

where λ^* denotes the theoretical optimum and c maps an arbitrary HPC to (possibly multiple) target metrics. The classical HPO problem is defined as $\lambda^* \in \arg \min_{\lambda \in \tilde{\Lambda}} \widehat{\text{GE}}(\lambda)$, i.e., the goal is

to minimize the estimated generalization error, see [10] for further details. Instead of optimizing only for predictive performance, other metrics such as model sparsity or computational efficiency of prediction (e.g., MACs and FLOPs or model size and memory usage) could be included, resulting in a multi-objective HPO problem [62, 30, 7, 57, 27]. $c(\lambda)$ is a black-box function, as it usually has no closed-form mathematical representation, and analytic gradient information is generally not available. Furthermore, the evaluation of $c(\lambda)$ can take a significant amount of time. Therefore, the minimization of $c(\lambda)$ forms an *expensive black-box* optimization problem.

Many HPO problems allow for approximations of the objective to a varying fidelity, making *multi-fidelity optimization* a viable option [44, 62, 35]. For example, in the context of fitting neural networks, it is possible to stop or pause training runs early when performance does not indicate a promising final result [67]. Another possibility is given by reducing the fraction of the dataset $\mathcal{D}_{\text{train}}$ used for training [38], since the complexity of evaluating $c(\lambda)$ is often at least linear in $|\mathcal{D}_{\text{train}}|$. Formally, the possibility of multi-fidelity evaluation can be represented in the form of a “budget” hyperparameter which we denote by λ_{budget} as a component of λ .

3.2 Hyperparameter Optimization Benchmarks

Benchmark suites are comprised of a set of benchmark *instances* that each define an optimization problem to be solved. We formally define benchmark instances adapted from [19] as:

Definition 1 (Benchmark Instance) *A benchmark instance consists of a function $g : \Lambda \rightarrow \mathbb{R}^m$, $m \in \mathbb{N}^+$, and a bounded hyperparameter space $\tilde{\Lambda}$ which is the Cartesian product of hyperparameters $\tilde{\Lambda}_1, \dots, \tilde{\Lambda}_d$. Multi-fidelity benchmarks can be queried at lower fidelities by varying the budget parameter $\tilde{\Lambda}_{\text{budget}} \in \tilde{\Lambda}$. While hyperparameters $\tilde{\Lambda}_i$ can be continuous, integer, ordinal or categorical, we require at least ordinal scales for the fidelity parameter(s) Λ_{budget} . We call a benchmark instance multi-objective if the number of objectives $m > 1$ and single-objective otherwise.*

We consider HPO benchmark instances estimating the generalization error $g(\lambda) = \widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \lambda)$ given an inducer \mathcal{I} , resampling \mathcal{J} , and performance metric(s) ρ , along with other possibly relevant metrics (computational cost, memory, ...). *Real* instances are based on actually performing these evaluations during the benchmark, while *tabular* instances are based on a fixed set of pre-recorded evaluations. Instances based on *surrogates* in turn approximate the functional relationship between λ and $g(\lambda)$. For clarity, we provide more precise definitions of *synthetic*, *tabular* and *surrogate* instances in Supplement B.3. *Real* instances rely on live evaluations of the generalization error and are therefore often prohibitively computationally expensive, especially when considering larger benchmarks or meta-learning scenarios across many tasks [70, 59, 24]. Practitioners therefore often rely on *tabular* or *surrogate* benchmarks for large benchmark studies because they are often cheaper to evaluate by orders of magnitude. For *tabular* benchmarks, a large collection of pre-computed hyperparameter performance mappings is provided, which serves as a look-up table during runs of HPO methods. This has the downside of constraining the search space to precomputed evaluations, essentially turning the optimization problem from a *continuous/mixed space* to a *discrete* optimization problem. *Surrogate* benchmarks can strike a balance between the efficiency and faithful approximation to the real problem by learning the functional relationship between hyperparameters and performance values yielding an approximation $\hat{g}(\lambda)$ of $g(\lambda)$. This allows evaluations across the full search space $\tilde{\Lambda}$ while being considerably cheaper to evaluate. The usefulness of surrogates in turn relies on the approximation quality of the surrogate model. We present an in-depth analysis of approximation qualities of the surrogates employed in YAHPO Gym in Supplement E.1.

Definition 2 (Benchmark Scenario) *A benchmark scenario consists of a set of K functions $g_k : \Lambda \rightarrow \mathcal{Y} \subseteq \mathbb{R}^m$, $m \in \mathbb{N}^+$, $k \in \{1, \dots, K\}$ corresponding to a set of Benchmark Instances. Each instance within a scenario shares the same bounded hyperparameter space $\tilde{\Lambda}$ (and therefore fidelity parameters) as well as the same co-domain \mathcal{Y} .*

Table 2: YAHPO Gym Benchmarks.

Scenario	Search Space	#Instances	Target Metrics	Fidelity	H
rbv2_super	38D: Mixed	103	9: perf(6) + rt(2) + mem	fraction	✓
rbv2_svm	6D: Mixed	106	9: perf(6) + rt(2) + mem	fraction	✓
rbv2_rpart	5D: Mixed	117	9: perf(6) + rt(2) + mem	fraction	
rbv2_aknn	6D: Mixed	118	9: perf(6) + rt(2) + mem	fraction	
rbv2_glmnet	3D: Mixed	115	9: perf(6) + rt(2) + mem	fraction	
rbv2_ranger	8D: Mixed	119	9: perf(6) + rt(2) + mem	fraction	✓
rbv2_xgboost	14D: Mixed	119	9: perf(6) + rt(2) + mem	fraction	✓
nb301	34D: Categorical	1	2: perf(1) + rt(1)	epoch	✓
lcbench	7D: Numeric	34	6: perf(5) + rt(1)	epoch	
iaml_super	28D: Mixed	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	✓
iaml_rpart	4D: Numeric	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	
iaml_glmnet	2D: Numeric	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	
iaml_ranger	8D: Mixed	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	✓
iaml_xgboost	13D: Mixed	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	✓

Mixed = numeric and categorical hyperparameters; perf = performance measures; rt = train/predict time; mem = memory consumption; inp = interpretability measures; H = Hierarchical search space. We do not include the fidelity parameter in the search space dimensionality.

A scenario is therefore a collection of instances sharing the same search space and objective(s), e.g., allowing for hyperparameter transfer learning between instances of the scenario. *Benchmark Suites* in turn are sets of instances that do not need to share the same objectives, but instead can consist of instances stemming from different scenarios.

4 YAHPO Gym

Motivated by the need for efficient and faithful benchmarks for HPO, we develop YAHPO Gym based on a set of *Criteria for HPO Benchmarks* discussed in Supplement B.1. YAHPO Gym is explicitly designed to use surrogate-based benchmarks only. It consists of a collection of 14 *scenarios* that can be evaluated across a total of ~ 700 instances. Each benchmark instance consists of an objective function that is parameterized in the form of a `ConfigSpace` Python object [48], making the search space computer-readable and readily usable with a range of existing HPO implementations. The objective function generates a prediction using the instance surrogate model, which is a compressed neural network. Table 2 provides an overview of all benchmark scenarios available in YAHPO Gym. We describe data sources as well as the full search spaces in Supplement F. We want to highlight the *rbv2_super* collection, which reflects an AutoML pipeline: It is, to our knowledge, the first available benchmark simulating a combined algorithm and hyperparameter selection problem [68] in the form of a high dimensional hierarchical search space by introducing the algorithm as an additional tunable hyperparameter.

In YAHPO Gym, every scenario allows for querying objective values at lower fidelities, enabling efficient benchmarking of multi-fidelity HPO methods. Analogously, every benchmark allows for returning multiple target metrics as criteria, enabling benchmarking of multi-objective HPO methods. Finally, almost all benchmark scenarios provide problems on a large number of instances (mostly ranging from 34 to 119), allowing for benchmarking of transfer-learning HPO methods. Predictions as well as sampling can be made reproducible through seeding. In order to achieve *portability* while still being *efficient*, YAHPO Gym uses fitted neural networks compressed via ONNX [3] as surrogate models. Our neural networks are ResNets for tabular data [26] consisting of up to 8 layers with a width of up to 512 and hyperparameters individually tuned for each scenario. We refer the reader to Supplement D for details regarding architecture and fitting procedure. Surrogate models have very small memory and inference time overhead and are compatible

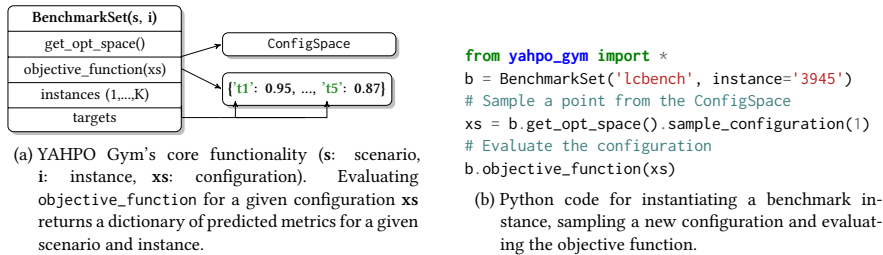


Figure 1: API overview.

across platforms and operating systems. In contrast to other benchmarks, evaluating $c(\lambda)$ requires only 10 – 100 ms and only 100 MB of memory. In fact, YAHPO Gym’s current infrastructure is so lightweight, it can easily be integrated in any existing toolbox or benchmark suite.

4.1 Suites: YAHPO-SO & YAHPO-MO

Together with YAHPO Gym, we propose two carefully selected *benchmark suites*. They constitute a proposal for surrogate-based benchmarks of HPO problems. We call those YAHPO-SO (single-objective, 20 instances) and YAHPO-MO (multi-objective, 25 instances). Together with the set of instances, we provide specific evaluation criteria, such as the budget available for optimization and number of stochastic replications as well as metrics to be used and fully specified search spaces which can be obtained from our software. Instances were selected across all scenarios taking into account approximation quality of the underlying surrogate and diversity. We consider those benchmarks a first draft for such a benchmark set (version v1.0) and explicitly invite the community to jointly work on a larger, more comprehensively evaluated set of benchmark instances. Details with respect to how instances were selected, and a full list of included instances, can be found in Supplement C.2. We conduct a benchmark providing anytime performance for a large variety of baselines on the proposed benchmark suites.

5 Tabular or Surrogate Benchmarks?

Consider the true objective $c(\lambda)$ of a *real* benchmark instance with $c : \tilde{\Lambda} \rightarrow \mathbb{R}$ in the single-objective setting. In a *tabular* benchmark, the domain of the objective function is implicitly discretized into a finite grid $\tilde{\Lambda}_{\text{discrete}}$ of the original domain and pre-evaluated at these points and the benchmark objective $\hat{c}_{\text{tabular}}(\lambda)$ is thus the original $c(\lambda)$ restricted to $\tilde{\Lambda}_{\text{discrete}}$. The extent to which discretization affects the faithfulness of tabular benchmarks depends on the nature and dimensionality of the search space: It disregards local structure in the response function and might even impose fixed fidelity schedules, should evaluations not be available at all budget levels. In order to assess the magnitude of this effect, we investigate the practical effects of discretization in the following experiment by comparing 8 black-box optimizers on *tabular*, *surrogate* and *real* versions of 5 synthetic multi-fidelity functions of varying dimensionality (Branin2D, Currin2D, Hartmann3D/6D, and Borehole8D [35]). The tabular benchmark is constructed by drawing and evaluating 10^6 points from a grid. Surrogates are then fitted using those points. We compare Random Search (RS), several versions of Bayesian optimization (BO) and Hyperband (HB, [44]) across all settings. BO is configured with algorithm surrogate model either a Gaussian process (BO_GP), ensemble of feed-forward neural networks (BO_NN, [73]) or random forest (BO_RF, [12]) and acquisition function optimizer either Nelder-Mead/exhaustive search³ (*_DF [54]) or Random

³for tabular benchmarks

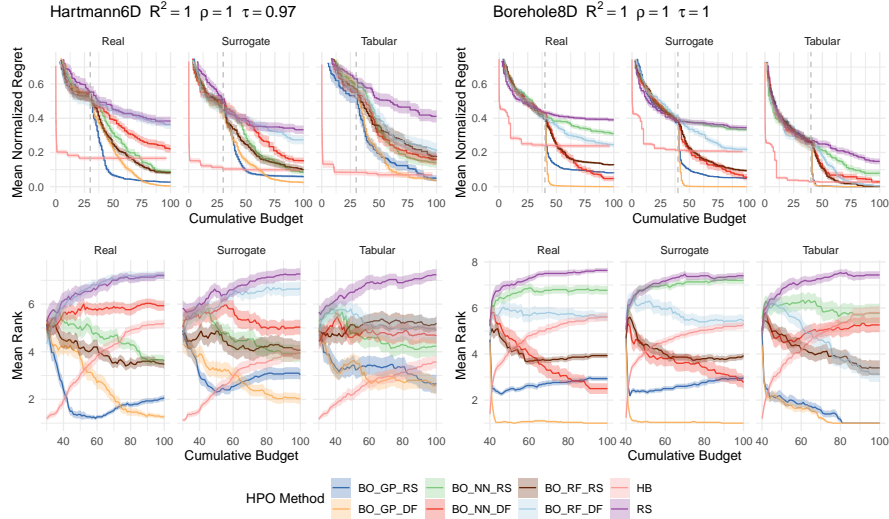


Figure 2: Mean normalized regret (top) and mean ranks (bottom) of different HPO methods on different benchmarks. Ribbons represent standard errors. The gray vertical line indicates the cumulative budget used for the initial design of BO methods. Performance measures of the surrogate benchmarks are stated after the benchmark function. 30 replications.

Search (*_RS). We describe additional details regarding the benchmark setup in Supplement E.1 and briefly present results: Figure 2 shows the anytime performance and mean rank of each HPO method split for the real, surrogate, and tabular benchmark on the Hartmann6D and Borehole8D test functions. We observe very similar performance traces of HPO methods on surrogate versions of benchmarks compared to real versions (Figure 2, top). However, in tabular benchmarks, we notice that for some problems, the BO methods converge substantially faster to a lower mean normalized regret (especially for BO_GP_*), which can possibly be explained by the much simpler infill optimization problem solved in the tabular case. Moreover, Hyperband appears to consistently perform better on tabular benchmarks. We further investigate average rankings over all replications (Figure 2, bottom). Each benchmark function yields an average ranking of HPO methods (e.g., with respect to final performance). Using consensus rankings, we can arrive at a single ranking over all benchmark functions [51] for a given benchmark type. We use the optimization based symmetric difference (SD) [36] minimizing rank reversals to compare both the surrogate and tabular inferred consensus rankings with the “ground truth” real function consensus ranking. We observe that consensus rankings obtained using surrogate benchmarks (permutation order 2) match more closely than tabular benchmarks (permutation order 5). We again provide additional details in Supplement E.1.

6 A Benchmark of HPO Methods on YAHPO Gym

We now demonstrate how YAHPO Gym can be used in practice to benchmark different HPO methods. We benchmark 7 single-objective HPO methods on YAHPO-SO and 7 multi-objective HPO methods on YAHPO-MO and want to answer the following research questions: **(RQ1)** *Do multi-fidelity (single-objective) HPO methods improve over full-fidelity methods?* **(RQ2)** *Do advanced multi-objective HPO methods improve over Random Search?*

6.1 RQ1: Do multi-fidelity (single-objective) HPO methods improve over full-fidelity methods?

We compare Random Search and SMAC (SMAC4HPO facade; [47]) to the multi-fidelity methods Hyperband [44], BOHB [21], DEHB [4], SMAC-HB (SMAC4MF facade; [47]) and optuna ([2]; TPE sampler and median pruner following successive halving steps). More details on the experimental setup and HPO methods is given in Supplement E.2. All optimizers are run for a total budget of $\lceil 20 + 40 \cdot \sqrt{\text{SEARCH_SPACE_DIM}} \rceil$ full-fidelity evaluations with 30 replications. Figure 3a shows the average rank of HPO methods with respect to their anytime performance. Figure 3b and Figure 3c show critical difference plots ($\alpha = 0.05$) of mean ranks after 25% and 100% of the optimization budget. The corresponding Friedman tests indicate significant differences ($p < 0.001$) in both cases. We observe that all multi-fidelity optimizers outperform Random Search with respect to intermediate performance (25% of optimization budget) and optuna, BOHB, SMAC-HB and Hyperband also outperform SMAC. With respect to final performance, SMAC takes the lead closely followed by SMAC-HB with other multi-fidelity optimizers slightly falling behind. We conclude that multi-fidelity HPO methods indeed improve over full-fidelity methods, but only with respect to intermediate performance. Our results are in line with what has been reported in other benchmarks [19] with the exception that optuna seems more competitive in our benchmark, while DEHB is less competitive. One reason for this difference might be that we include hierarchical search spaces in contrast to previous work.

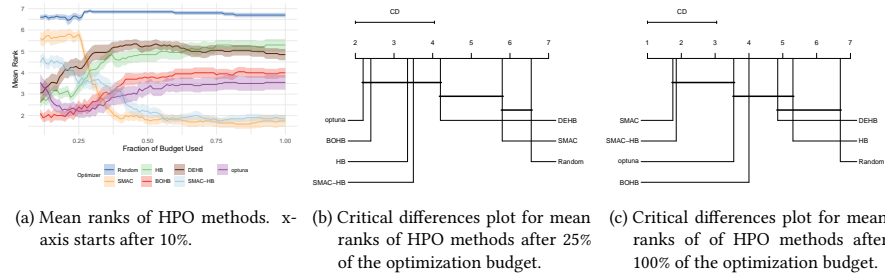


Figure 3: Results of YAHPO-SO single-objective benchmark across 7 optimizers (20 instances).

6.2 RQ2: Do advanced multi-objective HPO methods improve over Random Search?

We compare Random Search, Random Search $\times 4$ (Random Search with quadrupled budget as a strong baseline), ParEGO [40], SMS-EGO [61], EHVI [20], MEGO [33] and MIES [46] on multi-objective HPO problems with 2 – 4 objectives. More details on the experimental setup and HPO methods is given in Supplement E.3. All optimizers are run for a total budget of $\lceil 20 + 40 \cdot \sqrt{\text{SEARCH_SPACE_DIM}} \rceil$ full-fidelity evaluations for 30 replications. Figure 4a shows the average rank of HPO methods with respect to their anytime performance (determined based on the normalized Hypervolume Indicator). Figure 4b and Figure 4c show critical difference plots ($\alpha = 0.05$) of these ranks after 25% and 100% of the optimization budget. The corresponding Friedman tests indicate significant differences ($p < 0.001$) in both cases. We observe that not all methods significantly improve over Random Search with respect to final performance, i.e., EHVI and SMS-EGO fail to do so. Especially with respect to intermediate performance (25% of optimization budget), Random $\times 4$ outperforms all competitors. However, with respect to final performance, MEGO, ParEGO and MIES yield similar performance catching up to Random $\times 4$. We conclude that, in general, advanced multi-objective HPO methods improve over Random Search but also want to highlight that optimizer performance strongly varies with respect to the different benchmark instances.

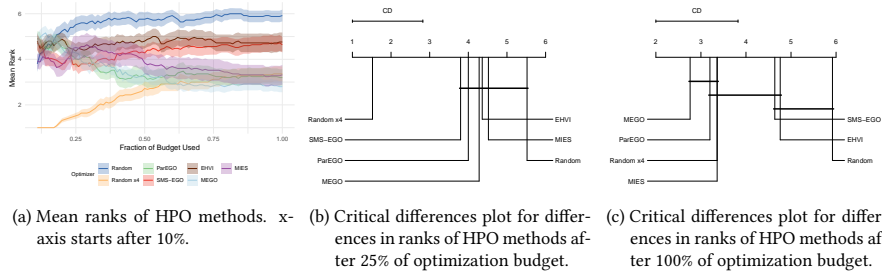


Figure 4: Results of the YAHPO-MO multi-objective benchmark across 7 optimizers (25 instances).

In total, both benchmarks described in this section took the equivalent of 139.57 CPU days using YAHPO Gym. We estimate that the YAHPO-SO benchmark, would take 14.75 CPU years when running real benchmarks, while our benchmark using YAHPO Gym took only 397.51 CPU hours, essentially speeding up evaluation by a factor of ~ 300 .

7 Conclusions, Limitations and Broader Impact

We present YAHPO Gym, a multi-fidelity, multi-objective benchmark for HPO. Our benchmark is based on surrogates, which strike a favorable trade-off between faithfulness and efficiency, which we demonstrate in various experiments throughout our paper before conducting a large scale benchmark of modern single- and multi-objective optimizers. An as of yet under-explored domain are asynchronous optimization algorithms, which have recently gained popularity [45]. This has been studied in surrogate-based benchmarks by predicting runtimes and pausing the objective function for the predicted runtime, lowering computational demand for benchmarks but leading to a large waiting time [21]. In future work we plan on introducing faster-than-real time asynchronous benchmarking based on predicted runtimes.

Limitations. YAHPO Gym is based on surrogate models and therefore heavily relies on the faithfulness of those models in order to allow for valid conclusions. We have comprehensively evaluated surrogate models and provide a detailed report of performance metrics, hoping to demonstrate the faithfulness of our surrogates, but can only do so to a certain degree. We are furthermore aware that the real HPO problems modeled in our surrogates are in fact stochastic, and results can vary depending on randomness of the fitting procedure, data splits or initialization. We therefore provide a set of *noisy* surrogate models that intend to model the stochasticity of the problems using an ensemble of neural networks, but simultaneously allow for full control of the stochastic process by using random seeds.

Broader Impact. This manuscript presents a set of surrogate-based benchmarks for HPO. As such, our work does not have direct implications on society or individuals, but can lead to such indirectly if new methods are developed based on it. We would like to emphasize the possible **societal & environmental benefits**. First, we hope our benchmarks can improve the state of benchmarking in hyperparameter optimization contexts, leading to better tracking of progress in the discipline. Second, and more important, we hope that experiments based on YAHPO Gym can drastically reduce **computational cost** of hyperparameter optimization experiments. This type of experiments is usually extremely expensive, if real experiments are run for the evaluation of each HPC, which can be sped up by large factors if cheap approximations through surrogates are available.

8 Reproducibility Checklist

1. For all authors...
 - (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
 - (b) Did you describe the limitations of your work? [Yes] See Section 7.
 - (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 7.
 - (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
 - (a) Did you state the full set of assumptions of all theoretical results? [N/A]
 - (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
 - (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., requirements.txt with explicit version), an instructive README with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] The full code for experiments, figures and table can be obtained from the following GitHub repositories:
 - i. Software: https://github.com/slds-lmu/yahpo_gym
 - ii. Documentation: https://slds-lmu.github.io/yahpo_gym/
 - iii. Surrogates & Search Spaces: https://github.com/slds-lmu/yahpo_data
 - iv. Code for Results: https://github.com/slds-lmu/yahpo_exps
 - (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] We make the full data used to train our surrogates available at <https://syncandshare.lrz.de/getlink/fiCMkzqj1bv1LFCUyvZKMLvd/>.
 - (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] See https://github.com/slds-lmu/yahpo_exps.
 - (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]
 - (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] See Supplement F for search spaces, the code repository as well as the software repository for further fixed hyperparameters.
 - (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] This is explicitly guaranteed by our software.
 - (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] Partially, see sections throughout the supplementary material.
 - (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

- (i) Did you compare performance over time? [Yes] Anytime performances are reported in all relevant figures throughout the paper.
 - (j) Did you perform multiple runs of your experiments and report random seeds? [Yes] We perform 30 replications for each run. Random seeds can be obtained from the accompanying code.
 - (k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All figures reporting experimental results include error bars.
 - (l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] Surrogate benchmarks.
 - (m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We state the total computation as well as CO₂ equivalent in the respective section and briefly summarize here: Tuning and fitting surrogates required a total of 45 GPU-days (116 kg CO₂-equivalent on NVIDIA DGX-A100 instances) while the main experiments require 139.57 CPU days across all replications (263 kg CO₂ equivalent). The tabular vs. surrogate benchmark required 22 CPU-hours (2 kg CO₂) equivalent.
 - (n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] We report tuning of surrogates in the supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
- (a) If your work uses existing assets, did you cite the creators? [Yes] Yes, throughout the paper and explicitly in Supplement F for datasets we base our surrogates on.
 - (b) Did you mention the license of the assets? [Yes] Yes, see Supplement F.
 - (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes, trained surrogates are available at https://github.com/slds-lmu/yahpo_data.
 - (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Data is meta-data about ML experiments and we do not consider any personal data. All used data is available via OSS Licenses and no consent was required.
 - (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Data is only metadata about ML experiments.
5. If you used crowdsourcing or conducted research with human subjects...
- (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No crowd sourcing.
 - (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] No IRB was required.
 - (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Acknowledgements. The authors of this work take full responsibilities for its content. This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics Data Applications (ADACenter)

within the framework of BAYERN DIGITAL II (20-3410-2-9-8). This work has been carried out by making use of AI infrastructure hosted and operated by the Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften and funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre (www.lrz.de). The authors gratefully acknowledge the computational and data resources provided by the *ARCC Teton HPC* [1].

References

- [1] Advanced Research Computing Center. Teton computing environment. <https://doi.org/10.15786/M2FY47>, 2018. University of Wyoming.
- [2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- [3] ONNX authors. ONNX. <https://github.com/onnx/onnx>, 2022.
- [4] N. Awad, N. Mallik, and F. Hutter. DEHB: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. *arXiv:2105.09821 [cs.LG]*, 2021.
- [5] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 2546–2554, 2011.
- [6] J. Bergstra, B. Komer, C. Eliasmith, and D. Warde-Farley. Preliminary evaluation of hyperopt algorithms on HPOLib. In *ICML Workshop on Automatic Machine Learning*, 2014.
- [7] M. Binder, J. Moosbauer, J. Thomas, and B. Bischl. Multi-objective hyperparameter tuning and feature selection using filter ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 471–479, 2020.
- [8] M. Binder, F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl. mlr3pipelines - Flexible machine learning pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021.
- [9] M. Binder, F. Pfisterer, and B. Bischl. Collecting empirical data about hyperparameters for data driven AutoML. In *ICML Workshop on Automatic Machine Learning*, 2020.
- [10] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, and M. Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv:2107.05847 [stat.ML]*, 2021.
- [11] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchet, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and Vanschoren J. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.
- [12] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [13] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10, 2016.
- [14] M. Dehghani, Y. Tay, A. A. Gritsenko, Z. Zhao, N. Houlsby, F. Diaz, D. Metzler, and O. Vinyals. The benchmark lottery. *arXiv:2107.07002 [cs.LG]*, 2021.
- [15] K. Eggenberger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.
- [16] K. Eggenberger, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Surrogate benchmarks for hyperparameter optimization. In *MetaSel@ ECAI*, pages 24–31, 2014.

- [17] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1114–1120, 2015.
- [18] K. Eggensperger, M. Lindauer, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1):15–41, 2018.
- [19] K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- [20] M. T. M. Emmerich. Single- and multi-objective evolutionary design optimization assisted by Gaussian random field metamodels. *PhD Dissertation*, 2005.
- [21] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446, 2018.
- [22] M. Feurer, B. Letham, F. Hutter, and E. Bakshy. Practical transfer learning for Bayesian optimization. *arXiv:1802.02219 [stat.ML]*, 2021.
- [23] M. Feurer, T. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI15)*, volume 15, pages 1128–1135, 2015.
- [24] P. Gijbbers, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 151–152, 2021.
- [25] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495, 2017.
- [26] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34, 2021.
- [27] J. Guerrero-Viu, S. Hauns, S. Izquierdo, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M. Lindauer, and F. Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. In *8th ICML Workshop on Automated Machine Learning*, 2021.
- [28] C. Guo and F. Berkhahn. Entity embeddings of categorical variables. *arXiv:1604.06737 [cs.LG]*, 2016.
- [29] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.
- [30] D. Horn and B. Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

- [31] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523, 2011.
- [32] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.
- [33] S. Jeong and S. Obayashi. Efficient global optimization (EGO) for multi-objective problem and data mining. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2138–2145, 2005.
- [34] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.
- [35] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity Bayesian optimisation with continuous approximations. In *International Conference on Machine Learning*, pages 1799–1808, 2017.
- [36] J. G. Kemeny and J. L. Snell. *Mathematical Models in the Social Sciences*. MIT Press, Cambridge, MA, USA, 1972.
- [37] A. Klein, Z. Dai, F. Hutter, N. Lawrence, and J. Gonzalez. Meta-surrogate benchmarking for hyperparameter optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019.
- [38] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54, pages 528–536, 2017.
- [39] A. Klein, L. C. Tiao, T. Lienart, C. Archambeau, and M. Seeger. Model-based asynchronous hyperparameter and neural architecture search. *arXiv:2003.10865 [cs.LG]*, 2020.
- [40] J. Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.
- [41] M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kothoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, 4(44):1903, 2019.
- [42] M. Lang, B. Bischl, and D. Surmann. batchtools: Tools for R to work on batch systems. *The Journal of Open Source Software*, 2017.
- [43] N. Lavesson and P. Davidsson. Quantifying the impact of learning algorithm parameter tuning. In *Proc. of AAAI*, volume 6, pages 395–400, 2006.
- [44] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.
- [45] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Bentzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, 2020.

- [46] R. Li, M. T. M. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. C. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):29–64, 2013.
- [47] M. Lindauer, K. Eggenberger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhopf, R. Sass, and F. Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.
- [48] M. Lindauer, K. Eggenberger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter. BOAH: A tool suite for multi-fidelity Bayesian optimization & analysis of hyperparameters. *arXiv:1908.06756 [cs.LG]*, 2019.
- [49] M. Lindauer, M. Feurer, K. Eggenberger, A. Biedenkapp, and F. Hutter. Towards assessing the impact of Bayesian optimization’s own hyperparameters. *arXiv:1908.06674 [cs.LG]*, 2019.
- [50] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.
- [51] O. Mersmann, H. Trautmann, B. Naujoks, and C. Weihs. Benchmarking evolutionary multi-objective optimization algorithms. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.
- [52] C. Molnar, G. Casalicchio, and B. Bischl. Quantifying model complexity via functional decomposition for better post-hoc interpretability. In P. Cellier and K. Driessens, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 193–204, 2020.
- [53] J. Moosbauer, M. Binder, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers. *arXiv:2111.14756 [cs.LG]*, 2021.
- [54] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.
- [55] C. Nießl, M. Herrmann, C. Wiedemann, G. Casalicchio, and A.-L. Boulesteix. Over-optimism in benchmark studies and the multiplicity of design and analysis options when interpreting their results. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1441, 2021.
- [56] T. Ohta and H. V. Yamazaki. Kurobako. <https://github.com/optuna/kurobako>, 2022.
- [57] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in Neuroscience*, 14:667, 2020.
- [58] V. Perrone, R. Jenatton, M. W. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.
- [59] F. Pfisterer, J. N. van Rijn, P. Probst, A. C. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 241–242, 2021.
- [60] S. Pineda Arango, H. S. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

- [61] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted \mathcal{S} -metric selection. In G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, editors, *Parallel Problem Solving from Nature PPSN X*, pages 784–794, 2008.
- [62] R. Schmucker, M. Donini, V. Perrone, M. B. Zafar, and C. Archambeaut. Multi-objective multi-fidelity hyperparameter optimization with application to fairness. In *NeurIPS Workshop on Meta-Learning*, volume 2, 2020.
- [63] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.
- [64] K. Šehić, A. Gramfort, J. Salmon, and L. Nardi. LassoBench: A high-dimensional hyperparameter optimization benchmark suite for lasso. *arXiv:2111.02790 [cs.LG]*, 2021.
- [65] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv:2008.09777 [cs.LG]*, 2020.
- [66] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012.
- [67] K. Swersky, J. Snoek, and R. P. Adams. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stat.ML]*, 2014.
- [68] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.
- [69] R. Turner. Uber bayesopt benchmark. <https://github.com/uber/bayesmark>, 2022.
- [70] J. Vanschoren. Meta-Learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 35–61. Springer International Publishing, 2019.
- [71] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013.
- [72] M. Volpp, L. P. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel. Meta-learning acquisition functions for transfer learning in Bayesian optimization. *International Conference on Learning Representations*, 2020.
- [73] C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.
- [74] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Learning hyperparameter optimization initializations. *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015.
- [75] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*, pages 199–214, 2016.

- [76] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7105–7114, 2019.
- [77] L. Zimmer. data_2k_lw.zip. figshare. Dataset. <https://doi.org/10.6084/m9.figshare.11662422.v1>, Apache License, Version 2.0, 2020.
- [78] L. Zimmer. nasbench301_full_data. figshare. Dataset. <https://doi.org/10.6084/m9.figshare.13286105.v1>, Apache License, Version 2.0, 2020.
- [79] L. Zimmer, M. Lindauer, and F. Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079–3090, 2021.
- [80] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

A Maintenance of YAHPO Gym

Following [19], we present a maintenance plan for YAHPO Gym.

- Who is maintaining the benchmarking library?
YAHPO Gym is developed and maintained by the *Statistical Learning and Data Science Group at LMU Munich*.
- How can the maintainer of the dataset be contacted (e.g., email address)?
Questions should be submitted via an issue on the Github repository at https://github.com/slds-lmu/yahpo_gym.
- Is there an erratum?
No.
- Will the library be updated?
We plan on adding new instances as well as continuously updating existing instances should need occur. Changes will be communicated via Github releases as well as a *CHANGELOG*.
- Will older versions of the benchmarking library continue to be supported/hosted/maintained?
Old versions are available via Github releases in the git repositories. We aim to support old versions on a best-effort basis with limited support for older versions.
- If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?
We have detailed how additional benchmarks can be added in the documentation https://slds-lmu.github.io/yahpo_gym/extending.html. We have furthermore made available the full code used to tune, fit and export surrogate models used in YAHPO Gym. The code is easily extendable for future datasets.
- Which dependencies does YAHPO Gym have?
YAHPO Gym currently relies on the following dependencies (versions used throughout experiments in brackets):
 - onnxruntime (1.10.0)
 - pyyaml (5.4.1)
 - configspace (0.4.20)
 - pandas (1.3.5)

B Benchmark Suites

B.1 Criteria for Benchmark Suites and Instances

To allow for a more systematic assessment of the quality of benchmarking instances, we define criteria that guided the development of YAHPO Gym and which should be satisfied to make a compelling argument for the use of any HPO benchmark.

- I. **Representativity & Diversity of Tasks** The goal of benchmark suites is to allow for a ranking of HPO methods according to their performance on future problems. Instances should therefore cover response surfaces encountered in relevant problem domains.
- II. **Difficulty and Structure** Benchmarks must be non-trivial, i.e., they should contain instances of sufficient difficulty to identify rankings between optimizers. Search spaces should reflect

search spaces that are encountered frequently in practice including mixed spaces with interactions as well as hierarchical spaces and sufficient dimensionality.

- III. **Faithfulness** Rankings based on approximations (e.g., for *tabular* and *surrogate* instances) should reflect true rankings. The performance of surrogate models \hat{g} should be close enough to g based on performance metrics such as Spearman’s ρ .
- IV. **Efficiency** Benchmark experiments often require repeated evaluation of several optimizers across several datasets leading to considerable computational (and consequentially environmental cost [63]). Benchmarks should therefore strive for computational efficiency.
- V. **Ease of use** Benchmark software needs to be accessible and portable across operating systems and programming languages. In practice, systems which do not require complex set up or establishment of databases might lead to more widespread adoption. Meta-data such as search spaces should be available and machine-readable. As benchmarks allow for embarrassingly parallel execution, parallelization should be supported.
- VI. **Reproducibility** While performance estimation in practice often includes stochastic components, it is important that benchmark suites can be made reproducible through the use of random seeds. Additionally, software dependencies and versions should be clearly communicated and design components should be fixed and versioned to avoid cherry picking.
- VII. **Stochasticity** Performance estimates obtained in real instances are realizations of random variables. In order to reflect this in practice, instances should allow for repeated evaluations.

While we consider the above requirements for good benchmarking suites, we furthermore want to highlight other properties that might be relevant for benchmarking suites.

- A. **Multi-fidelity** Multi-fidelity methods have been shown to considerably speed up evaluation. Benchmark instances should therefore allow for querying performances at multiple fidelities.
- B. **Runtime** In practice, HPO evaluations, especially for complex AutoML scenarios, can have very heterogeneous runtimes [66], which should also be reflected in a realistic benchmark by providing access to (estimated) runtimes which could subsequently be used to more accurately benchmark cost-efficient optimization methods.
- C. **Asynchronous Evaluation** Although technically non-trivial, benchmarks should ideally allow the comparison of parallel HPO methods, allowing to compare, e.g., asynchronous HPO procedures [45, 39].
- D. **Multi-Objective** In many scenarios, users are not only interested in maximizing a single performance metric such as accuracy, but instead multiple relevant metrics such as calibration, inference time, memory usage, and many others. We therefore consider including multi-objective HPO problems an important characteristic of a benchmark suite.
- E. **Meta-Learning** Last but not least, in many cases, data collections are used to test scenarios for *meta-learning* [70, 59, 24] or *transfer learning* [75, 58]. For these scenarios, the availability of data across a large amount of datasets is often useful.

B.2 Comparison to other Benchmark Suites

While a variety of benchmarking suites for optimization such as COCO [29], HPOLib [16], ASlib [11] and others exist, we do not go into detail and instead refer the reader to [19] where those libraries are discussed in more detail. We instead compare YAHPO Gym to the most similar suites: HPOBench [19] and HPO-B [60] and discuss and justify assessments made in Table 1.

Evaluations in Table 1 follow the doctrine “*the documentation is the product*” and we therefore consider only features that are explicitly documented in the accompanying manuscript and doc-

umentation, not considering other features. We note that all three libraries could theoretically be used or extended for additional tasks such as multi-objective evaluations but instead focus on scenarios where the considered property is explicitly included in the documented API. We furthermore note that several important aspects such as ease of use are not easily quantifiable and assessments made are therefore subjective. We derive assessments made in this section based on the criteria defined in Supplement B.1.

- I. **Representativity** YAHPO Gym contains 14 across diverse search spaces for widely used ML algorithms trained on representative datasets. Search spaces are often mixed and sometimes include dependent hyperparameters resulting in a hierarchical search space. While theoretically possible, none of the instances in HPOBench currently contain hierarchical search spaces. HPO-B only supports continuous search spaces.
- II. **Difficulty** To the best of our knowledge, it is not yet clear how to assess the difficulty of a benchmark instance. We therefore instead focus on showing that benchmark instances in YAHPO Gym are not trivial, e.g., constant across the full search space.
- III. **Faithfulness** We evaluate the quality of fitted surrogates in Supplement D.2. To the best of our knowledge, analyses that establish the faithfulness of tabular benchmarks have not been conducted for tabular benchmarks previously.
- IV. **Efficiency** We consider efficiency with respect to two aspects: *computational cost* and *memory consumption*. Tabular benchmarks often keep the full data in memory, essentially limiting the amount of parallel optimization runs on a given hardware required, e.g., for replications of stochastic benchmark experiments. Moreover, surrogate benchmarks are often based on un-optimized models fitted for each single instance. As a result, the required metadata (and memory consumption when multiple models are kept in memory) is often comparatively large. Our surrogates in contrast are highly optimized, compressed neural networks fitted across an entire scenario. Our surrogates are furthermore portable across platforms, alleviating concerns regarding software dependencies. Prediction on a surrogate requires only 10-100 ms and around 100 MB of memory allowing for a high degree of parallelization. In a small experiment, we estimate runtime and memory overhead for 300 iterations of Random Search on comparable SVM search spaces in Table 1 using the Python memory profiler (<https://pypi.org/project/memory-profiler/>). Since memory profiling is not accurate for HPO-Bench due to external processes, we estimate memory consumption using `htop`. Differences partially stem from more expensive setup in other libraries, but we consider 300 iterations of Random Search a representative use-case for many scenarios. Benchmarks were conducted on an AMD Ryzen 5 3600 6-Core CPU.
- V. **Ease of use** YAHPO Gym does not require setting up containerization or any database and has only four dependencies that are both widely used and mature. All metadata required can be downloaded from a single, versioned metadata repository⁴. The modules API is simple to use (see, e.g., Figure 1). Other benchmarking suites either require *benchmark instance specific* software dependencies that can differ from benchmark instance to instance. While HPOBench has solved this using *containerization* adding considerable computational overhead, our surrogates only rely on a single fixed version of ONNX and can therefore completely ignore the problem.
- VI. **Reproducibility** surrogates used in the benchmarking suites proposed along with YAHPO Gym are deterministic. Reproducibility therefore only requires ensuring seeding of any stochastic procedures in the optimization algorithm. Furthermore, we fix several design choices that might lead to differences between benchmarks: *i*) search spaces $\tilde{\Lambda}$ are fixed

⁴https://github.com/slds-lmu/yahpo_data

for each scenario and should be used in benchmarks *ii*) target metrics and exact evaluation protocol are fixed within the benchmark suites (see Supplement C.2) to ensure comparability.

Additional properties A. – E. described in Supplement B.1 are compared in Table 1 and described in more detail below.

- A. **Multi-fidelity** Only surrogate based benchmarks allow doing so for the full range of available fidelity steps. This essentially enforces evaluation at fixed fidelities in tabular benchmarks, e.g., disallowing evaluation of differing fidelity schedules. In contrast, surrogates in YAHPO Gym allow for evaluation at all fidelity steps.
- B. **Runtime** All surrogates in YAHPO Gym allow for querying the predicted runtime for training a configuration, essentially allowing benchmarking methods that take into account runtimes.
- C. **Asynchronous Evaluation** To our knowledge, none of the existing benchmark suites allow for asynchronous evaluation (except for *real* instances in *HPO-Bench*). YAHPO Gym currently allows for asynchronous evaluation, but this is considered an experimental feature. We hope to be able to fully allow asynchronous benchmarking in future versions of our benchmark.
- D. **Multi-Objective** YAHPO Gym explicitly includes multiple objective for each scenario and allows the user to subset the returned targets explicitly. In contrast, HPO-Bench contains only few multi-objective benchmarks and does not explicitly document how they are supposed to be used.
- E. **Transfer Learning** All considered suites allow for transfer learning. In contrast to *HPO-Bench* and *HPO-B*, YAHPO Gym includes the (to our knowledge) largest collection of instances for a given scenario for the *rbv2** scenarios consisting of up to 119 instances. Only few collections in HPOBench contain enough instances for meta-learning.

We furthermore define a *single objective* as well as a *multi-objective* benchmark task that include a evaluation protocol with respect to instances, search spaces, evaluation budget and target metrics. This allows for reproduction and extension by practitioners without additional design choices and provides a singular point of references.

B.3 A Benchmark Instance

In order to improve differentiation, we formally define four different types of benchmark instances derived from Definition 1. We therefore only consider benchmarks based on *tabular*, *surrogate* and *real* instances in our manuscript.

Definition 3 (Synthetic Benchmark Instance) *A synthetic benchmark instance is a benchmark instance, where $g : \lambda \rightarrow \mathbb{R}^m$ is a mathematically tractable function.*

Synthetic instances, such as the ones, e.g., included in *COCO* [29] rely on mathematically tractable test functions (e.g., Rosenbrock-2D) as response surface. While they provide cheap evaluations, problem structures in such functions are qualitatively distinct from test functions encountered in HPO scenarios, and the resulting optimization problem is therefore often not representative for optimization problems typically encountered in HPO.

Definition 4 (Tabular Benchmark Instance) *A tabular benchmark instance returns function evaluations $g(\lambda)$ from a table of pre-recorded performance results. Performance results are typically obtained by estimating $\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \lambda)$ for given \mathcal{I} , \mathcal{J} and ρ . In contrast to synthetic and surrogate instances, the search space Λ is discretized and g can therefore be only evaluated at discrete points $\tilde{\lambda} \in \Lambda$.*

Definition 5 (Surrogate Benchmark Instance) *A surrogate benchmark returns predictions $\hat{g}(\lambda)$ of machine learning models trained to infer the functional relationship between λ and function evaluations $g(\lambda)$ based on a set of pre-recorded performance results.*

For clarity, we would like to differentiate in terminology between the *instance surrogate* of a surrogate benchmark, and the algorithm surrogate potentially used by an HPO method, e.g., the Gaussian process as surrogate model in BO explicitly mentioning the algorithm surrogate where required. The instance surrogate model \hat{g} or the tabular data should approximate the true relationship between λ and the target metrics reasonably well. We consider a mapping \hat{g} to be *faithful* if:

1. cross-validated performance metrics are sufficiently good with respect to metrics such as R^2 and Spearman's ρ . We typically consider a cutoff $\rho > 0.7$ for including a surrogate.
2. if the induced ranking of optimizers on a given \hat{g} closely resembles the true rankings on the original underlying optimization problem (in general, the *real* setting relying on g).
3. learning curves of HPO methods on \hat{g} closely resemble the true performance curves.

Definition 6 (Real Benchmark Instance) *A real benchmark instance returns function evaluations $g(\lambda)$. Performance results are typically obtained by estimating $\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \lambda)$ for given \mathcal{I}, \mathcal{J} and ρ .*

Since the same benchmark instance can be provided as a *real*, *tabular*, or *surrogate* instance, we speak of different *versions* of that instance where required.

C YAHPO Gym

In the following we will provide additional details on general aspects of YAHPO Gym. A detailed description of included surrogates can be found in Supplement D and a detailed description of used data and included search spaces can be found in Supplement F.

C.1 Usage

The `yahpo_gym` software can be directly installed from *GitHub*⁵ and only requires downloading one additional GitHub repository containing metadata⁶ in an initial setup step.

HPO Benchmarking

To ensure interoperability with different optimizer API's, YAHPO Gym offers only evaluation of the objective function using the `BenchmarkSet.objective_function(xs)` method (where `xs` is a hyperparameter configuration to be evaluated). This allows for use with many different optimizers (see, e.g., examples provided in the accompanying notebooks). We furthermore allow for querying the search space using `BenchmarkSet.get_opt_space(xs)` in order to ensure that optimizers are ran on comparable search spaces. We provide additional details with respect to exact setups.

Transfer HPO

Different forms of Transfer HPO are available in YAHPO Gym and can be setup analogous by querying the objective function across different instances of the scenario. We present examples in the modules documentation.

⁵https://github.com/slids-lmu/yahpo_gym

⁶https://github.com/slids-lmu/yahpo_data

C.2 Benchmark Suites: YAHPO-SO & YAHPO-MO

This section provides additional details with respect to the two benchmark sets proposed with YAHPO Gym. Both suites can be obtained via `get_suites(<type>, <version>)` specifying the type of the benchmark (currently supporting “single” for YAHPO-SO and “multi” for YAHPO-MO) and the version (currently 1.0).

- Optimizers should use the search spaces included in YAHPO Gym in order to establish that differences in performance do not depend on differing search spaces.
- Optimization should be run for $\lceil 20 + 40 \cdot \sqrt{\text{SEARCH_SPACE_DIM}} \rceil$ steps. Each step is equivalent to a full budget evaluation, essentially allowing multi-fidelity method the same number of full budget equivalents. We report the budgets for each scenario in Table 3 and Table 4.
- Target metrics to be used with the single-objective and multi-objective suite are reported in Table 3 and Table 4.
- We encourage reporting *mean normalized regret* and *mean ranks* for the anytime performance of an optimizer. Reported values are based on the target metric for YAHPO-SO and the normalized Hypervolume Indicator for YAHPO-MO.
- In order to assess variance, we encourage reporting averages and standard errors across 30 replications with differing random seeds.

We will now go on to discuss criteria for inclusion of tasks in the respective benchmarks.

In light of the criteria defined in Supplement B.1, we strive for diversity by including instances from all included scenarios. We consider only surrogates that are *faithful* (measured via Spearman’s ρ reported for each target below). Our benchmarks are made available through a fully documented API. Inference on a surrogate model is highly efficient taking usually only 10-100 milliseconds per batch. Benchmarks are furthermore reproducible and allow for parallelization and runtime prediction on a continuous range of fidelities. We include search spaces for all problems in Supplement F.

We furthermore briefly want to discuss selecting a budget that depends on the scenario at hand. We consider the search space dimension to be a relevant input for determining the overall optimization budget that should be used for optimization. Our formula ensures, that optimization runs for a minimum of 77 iterations (`iaml_glmnet`, 2D) and a maximum of 267 (`rbv2_super`, 38D) iterations, which we consider useful bounds for the respective search space dimensionality, especially given that multi-fidelity allows for evaluations at a fraction of the full budget.

C.3 R package

While we focus on the Python module in the manuscript, YAHPO Gym offers an R interface that is equivalent in functionality. We do not present the API in detail here since it follows the same principles and naming conventions as the Python module. Further information is available from the package documentation. Listing 1 contains the sample R-code used to first draw a random configuration from the search space and then evaluate the drawn configuration.

D YAHPO Gym Surrogates

On an implementation level, YAHPO Gym consists of a (versioned) Python module / R package `yahpo_gym` and a (versioned) set of required metadata (including fitted surrogate models) which we will call `yahpo_data` in the following. The core contribution in YAHPO Gym is a set of surrogate

Table 3: YAHPO-SO (v1): Collection of single-objective benchmark instances. We indicate surrogate approximation quality using Spearman’s ρ .

	Scenario	Instance	Target	ρ	Budget
1	lcbench	167168	val_accuracy	0.94	126
2	lcbench	189873	val_accuracy	0.97	126
3	lcbench	189906	val_accuracy	0.97	126
4	nb301	CIFAR10	val_accuracy	0.98	250
5	rbv2_glmnet	375	acc	0.80	90
6	rbv2_glmnet	458	acc	0.85	90
7	rbv2_ranger	16	acc	0.93	134
8	rbv2_ranger	42	acc	0.98	134
9	rbv2_rpart	14	acc	0.92	110
10	rbv2_rpart	40499	acc	0.97	110
11	rbv2_super	1053	acc	0.31	267
12	rbv2_super	1457	acc	0.70	267
13	rbv2_super	1063	acc	0.57	267
14	rbv2_super	1479	acc	0.36	267
15	rbv2_super	15	acc	0.75	267
16	rbv2_super	1468	acc	0.77	267
17	rbv2_xgboost	12	acc	0.93	170
18	rbv2_xgboost	1501	acc	0.89	170
19	rbv2_xgboost	16	acc	0.91	170
20	rbv2_xgboost	40499	acc	0.96	170

```

library("yahpogym")
library("paradox")
library("bbotk")
# Instantiate the BenchmarkSet
b = BenchmarkSet$new('lcbench', instance='3945')
# Get the objective
objective = b$get_objective('3945', check_values = FALSE)
# Sample a point from the ConfigSpace
xdt = generate_design_random(b$get_search_space(), 1)$data
xss_trafoed = transform_xdt_to_xss(xdt, b$get_search_space())
# Evaluate the configuration
objective$eval_many(xss_trafoed)

```

Listing 1: R-code to sample and evaluate a configuration using YAHPO Gym.

models⁷ based on neural networks. This section provides additional details with respect to the fitting procedures of surrogate models as well as a rigorous evaluation of the final surrogates.

D.1 Setup and Training

Previous work [17, 18, 65] suggests that tree based regression methods such as random forests [12] are very suited as instance surrogate models for (single-objective) benchmarks. However, in YAHPO Gym we want to predict multiple target metrics for each instance of a benchmark collection efficiently and compactly. As a result, we use neural network surrogates because they 1) can naturally handle multiple outputs and do not require a model for each target metric and 2)

⁷available at https://github.com/slds-lmu/yahpo_data

Table 4: YAHPO-MO (v1): Collection of multi-objective benchmark instances. We indicate surrogate approximation quality using Spearman’s ρ (averaged over targets).

	Scenario	Instance	Targets	ρ	Budget
1	iaml_glmnet	1489	mmce,nf	0.86	77
2	iaml_glmnet	1067	mmce,nf	0.73	77
3	iaml_ranger	1489	mmce,nf,ias	0.93	134
4	iaml_ranger	1067	mmce,nf,ias	0.92	134
5	iaml_super	1489	mmce,nf,ias	0.82	232
6	iaml_super	1067	mmce,nf,ias	0.82	232
7	iaml_xgboost	40981	mmce,nf,ias	0.88	165
8	iaml_xgboost	1489	mmce,nf,ias	0.92	165
9	iaml_xgboost	40981	mmce,nf,ias,rammodel	0.89	165
10	iaml_xgboost	1489	mmce,nf,ias,rammodel	0.92	165
11	lcbench	167152	val_accuracy,val_cross_entropy	0.98	126
12	lcbench	167185	val_accuracy,val_cross_entropy	0.91	126
13	lcbench	189873	val_accuracy,val_cross_entropy	0.93	126
14	rbv2_ranger	6	acc,memory	0.90	134
15	rbv2_ranger	40979	acc,memory	0.73	134
16	rbv2_ranger	375	acc,memory	0.85	134
17	rbv2_rpart	41163	acc,memory	0.85	110
18	rbv2_rpart	1476	acc,memory	0.80	110
19	rbv2_rpart	40499	acc,memory	0.83	110
20	rbv2_super	1457	acc,memory	0.66	267
21	rbv2_super	6	acc,memory	0.68	267
22	rbv2_super	1053	acc,memory	0.45	267
23	rbv2_xgboost	28	acc,memory	0.80	170
24	rbv2_xgboost	182	acc,memory	0.79	170
25	rbv2_xgboost	12	acc,memory	0.76	170

should scale better than a random forest (fitted on each target metric) if the dimensionality of the data (especially in the number of features) increases.

Surrogate models used in YAHPO Gym are based on ResNet architectures for tabular data [26]. Instead of relying on a fixed architecture, we tune the neural network for each *Scenario* using optuna [2]. We used the Adam optimizer for a maximum of 100 epochs (early stopping with patience of 10) with L2 loss. Surrogates were trained jointly for each benchmark scenario (for all instances and target metrics). We use a stratified train/validation/test split of 0.6/0.2/0.2, using the validation data to determine the surrogate model architecture and report performances on the test set. The search space as well as the fully reproducible code for fitting can be obtained at YAHPO Gym. Tuning and fitting of a single *Scenario* takes 3 GPU days on average on an NVIDIA DGX-A100 instance, we therefore estimate a one time cost of 45 GPU days for establishing the full benchmark.

We adapt the architecture proposed in [26] in multiple ways:

Feature- and Output-Scaling Hyperparameters as well as resulting performance metrics (e.g learning rates of log-loss values) often vary across orders of magnitudes. We have practically observed that transforming target metrics to the unit cube prior to training and reverse-transforming afterwards massively improves quality of the resulting surrogates. Available scaling techniques include *Neg-Exp* and *Log* transformation before scaling to $[0, 1]$. We furthermore include clamping to ensure that predictions are in valid ranges. Non-numeric features were transformed via entity embeddings [28].

Ensembles In order to allow for an estimate of variance, we make *noisy* versions of our surrogates available together with the standard *deterministic* set of surrogates. Ensembles consist of replications of the architecture determined during tuning and fitted on different permutations of the data with differing initial weights. The prediction step is the weighted average over predictions from ensemble members with weights α_i sampled from a Dirichlet distribution.

We additionally consider scenarios that allow simulating **asynchronous evaluation** and therefore predict the time of the training procedure using our surrogates. YAHPO Gym currently supports asynchronous scheduling by estimating the runtime of training a model and then idling the system for the estimated time. This is implemented via `objective_function_timed` in `yahpo_gym` but currently considered in an experimental status.

In future work, we hope to propose and evaluate a surrogate-based benchmark explicitly allowing for benchmarking of asynchronous scheduling strategies based on surrogate predictions. To enable more realistic scheduling, we hope to furthermore include memory constraints using predicted peak memory consumption for a training run.

D.2 Surrogate Quality

We provide an overview over surrogate quality measured on the test set using Spearman’s ρ averaged across all instances in Table 5. Metrics are routinely ≥ 0.9 except for few instances / target metrics and even surpasses performances for surrogate models reported, e.g., in [65]. We furthermore depict real and predicted learning curves for four randomly drawn configurations in Figure 5. Note that in our work, learning curves are predicted only based on hyperparameters, and not based on initial, low-fidelity observations (as done in learning curve prediction tasks). Our surrogates therefore solve a much harder task. Surrogates in general predict the learning curves with a high degree of precision.

Table 5: Average surrogate performance (Spearman’s ρ) across all instances per scenario/target. We abbreviate `cross_entropy` (ce) and `balanced_accuracy`(bac) for brevity.

Scenario	ρ
<code>iaml_glmnet</code>	mmce:0.97,f1:0.9,auc:0.92,logloss:0.97,rammodel:0.97,timetrain:0.95,mec:0.9,ias:0.91,nf:0.97
<code>iaml_ranger</code>	mmce:0.99,f1:0.98,auc:1,logloss:0.95,rammodel:1,timetrain:0.91,mec:0.88,ias:0.98,nf:1
<code>iaml_rpart</code>	mmce:0.99,f1:0.96,auc:0.99,logloss:0.96,rammodel:1,timetrain:0.96,mec:0.71,ias:0.96,nf:0.96
<code>iaml_super</code>	mmce:0.93,f1:0.95,auc:0.89,logloss:0.93,rammodel:0.71,timetrain:0.61,mec:0.94,ias:0.65,nf:0.92
<code>iaml_xgboost</code>	mmce:0.97,f1:0.98,auc:0.97,logloss:0.93,rammodel:0.86,timetrain:0.71,mec:0.95,ias:0.84,nf:0.99
<code>lcbench</code>	time:0.94,val_accuracy:0.95,val_ce:0.97,val_bac:0.98,test_ce:0.99,test_bac:0.98
<code>nb301</code>	val_accuracy:0.98,runtime:0.94
<code>rbv2_aknn</code>	acc:0.99,bac:0.99,auc:0.98,brier:1,f1:0.91,logloss:0.99,timetrain:0.64,memory:0.83
<code>rbv2_glmnet</code>	acc:0.99,bac:0.95,auc:0.91,brier:1,f1:0.96,logloss:0.99,timetrain:0.79,memory:0.82
<code>rbv2_ranger</code>	acc:0.99,bac:0.98,auc:0.95,brier:1,f1:0.92,logloss:1,timetrain:0.84,memory:0.66
<code>rbv2_rpart</code>	acc:0.98,bac:0.96,auc:0.93,brier:0.99,f1:0.93,logloss:0.98,timetrain:0.72,memory:0.86
<code>rbv2_super</code>	acc:0.82,bac:0.78,auc:0.73,brier:0.91,f1:0.91,logloss:0.89,timetrain:0.69,memory:0.71
<code>rbv2_svm</code>	acc:0.99,bac:0.98,auc:0.94,brier:0.99,f1:0.91,logloss:0.99,timetrain:0.76,memory:0.84
<code>rbv2_xgboost</code>	acc:0.98,bac:0.96,auc:0.94,brier:0.99,f1:0.92,logloss:0.98,timetrain:0.93,memory:0.78

Some of the targets available require further study and we therefore discourage their use in benchmarks. Those are *rampredict* & *ramtrain* (`iaml_*` scenarios) as well as *timepredict* (`rbv2_*` scenarios). Reasons for this assessment are partially poor surrogates, but we also assume that the underlying data is at fault: Prediction times are often very small and heavily influenced by system load, while correct estimation of required memory are relatively difficult to obtain in general.

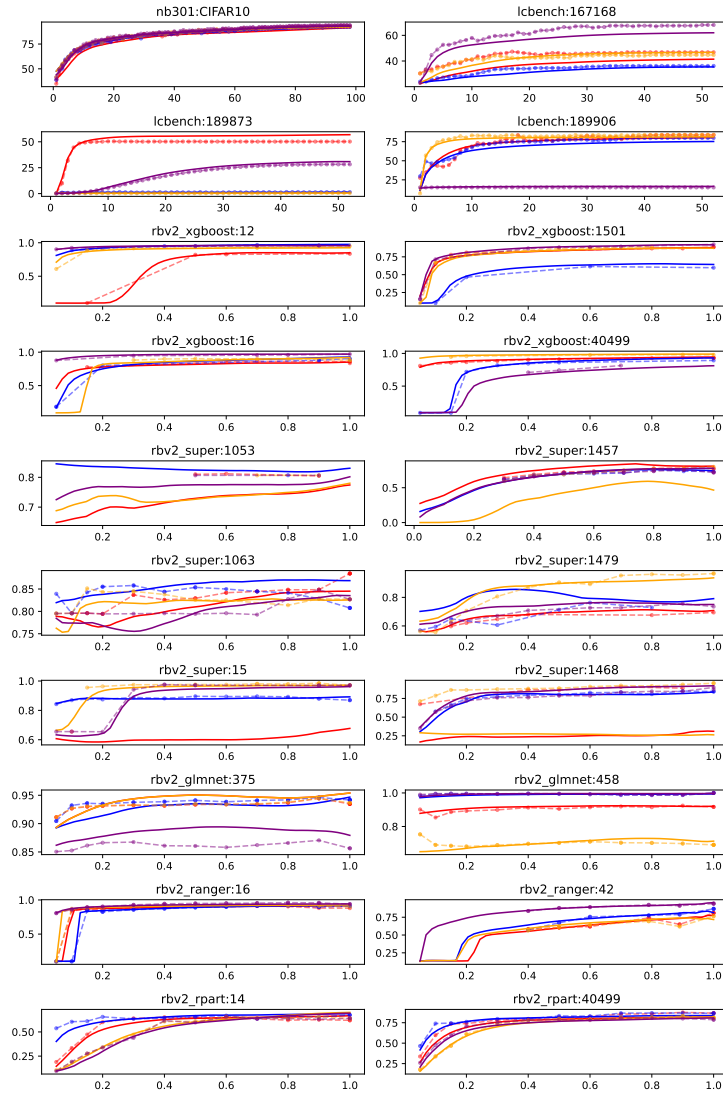


Figure 5: Predicted learning curves (lines) together with true learning curves (dotted) for four randomly drawn configurations (differentiated by colour) out of each instance in YAHPO-SO reporting the respective target metric.

D.3 Instance Difficulty

We quantify difficulty of instances using the empirical cumulative distribution function (ECDF), assuming that difficult instances have only a small probability mass close to the optimum. ECDFs

for all instances in YAHPO-SO are shown in Figure 6. Differences between real evaluations and surrogate predictions can stem from the sampling procedure (random on surrogates vs. unknown sampling for real evaluations), as well as biases in the surrogates. All evaluations are made at maximal fidelity. We furthermore provide ECDF plots for all optimizers in Figure 7. This allows for a different perspective on the quality of solutions found by the different optimizers.

E Experiments

E.1 Tabular vs. Surrogate Benchmarks

Resolution of Tabular Benchmarks. In practice, the resolution of grid points needs to be low for high dimensional spaces to limit the resulting table to a usable size. With purely categorical search spaces, often used in NAS, an exhaustive (i.e., $\tilde{\Lambda}_{\text{discrete}} = \tilde{\Lambda}$) tabular benchmark is often possible, as in, e.g., NAS-Bench-101 [76], which contains “only” 423k unique architectures. Multi-fidelity evaluations essentially add an additional dimension to the optimization problem when considering tabular data, since each evaluation now needs to be stored at multiple fidelity steps. If fidelity steps are not available at all budget levels, optimization benchmarks can be restricted to fixed fidelity progression (e.g., geometric progression as used in Hyperband).

Discrete Search Spaces. The modification of the search space from $\tilde{\Lambda}$ to $\tilde{\Lambda}_{\text{discrete}}$ can be handled in one of two ways: One can let HPO methods operate on the original search space $\tilde{\Lambda}$ and transparently “round” values to the nearest point contained in $\tilde{\Lambda}_{\text{discrete}}$. This effectively presents the optimization algorithm with a locally constant objective function. Alternatively, one can inform the HPO algorithm about the discrete nature of $\tilde{\Lambda}_{\text{discrete}}$, and possibly even modify the optimization procedure. As an example, consider the acquisition function optimization step within the BO framework: In the context of tabular benchmarks, the problem of optimizing the infill criterion becomes trivial because one can perform an exhaustive search over all points not yet evaluated to determine the next candidate(s) for evaluation. Note that we could also proceed to use a 1-Nearest-Neighbor model to evaluate HPCs in tabular benchmarks. This essentially results in a surrogate benchmark because we now rely on a performance model for the evaluation. In contrast to approximation by discretization, in a surrogate benchmark the domain of the objective function is not explicitly altered. Instead, predictions of an instance surrogate regression model $\hat{g}(\cdot)$ are returned as function evaluations, $\hat{c}_{\text{surrogate}} : \tilde{\Lambda} \rightarrow \mathbb{R}^m$, $\lambda \mapsto \hat{g}(\lambda)$. The drawback here is that values returned by the surrogate model may misrepresent the local structure of the problem as well. Beyond the resolution of the surrogate model training data, these structures are interpolated and influenced by the inductive bias implied by the model.

Experimental Setup. As a *real* benchmark, we consider the original synthetic benchmark function, while we generate a grid containing at most 10^6 points for the tabular version, storing these pre-evaluated points in a look-up table together with their function value. The resolution of the grid is the same for all functions along the budget parameter dimension, with 10 grid points ranging from 2^{-9} to 1 on a 2^x scale. For all other parameters of the domain, an equidistant grid was generated by using $\lfloor (10^5)^{\frac{1}{d}} \rfloor$ grid points for each dimension $d = 1, \dots, D$. With the same data we employ a similar surrogate neural network as used in YAHPO Gym. We compare the following methods on real, surrogate, and tabular benchmarks: All HPO methods were run for a total budget of 100 evaluations reflecting 100 full fidelity evaluations. The synthetic test functions used in the experiments [35] include a multi-fidelity parameter allowing for the use of multi-fidelity methods such as Hyperband. Of the methods investigated, only HB makes use of the fidelity parameter, while all other methods perform full budget evaluations. As a surrogate, we train a Wide & Deep Network [13]. More details can be found in https://github.com/slds-lmu/yahpo_exps. BO variants used Expected Improvement [34] as acquisition function and an initial design of $5 \cdot D$ points sampled uniformly at random. The Gaussian process surrogate model used a Matérn $3/2$

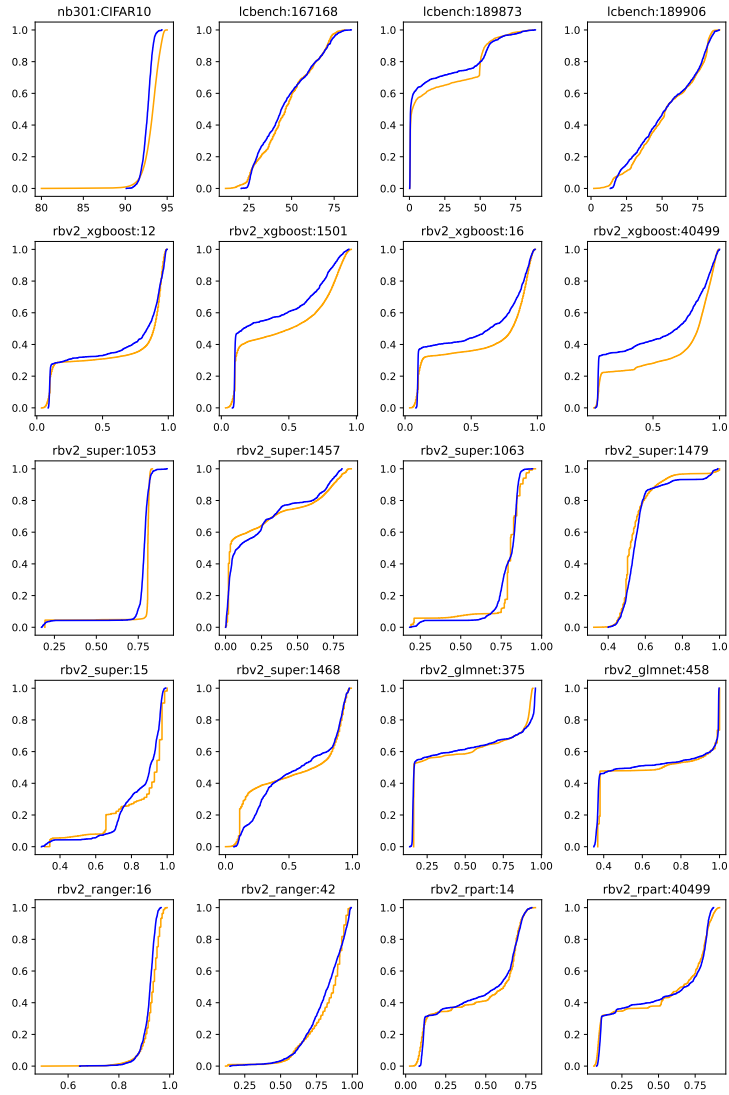


Figure 6: Empirical cumulative distribution function (ECDF) for surrogate predictions (blue) and real evaluations (orange).

kernel. Nelder-Mead as acquisition function optimizer was terminated if the relative change in the maximum fell below $1e - 4$. Tabular benchmarks used an exhaustive search for optimizing the acquisition function in the scenario of *_DF. Random Search as acquisition function optimizer

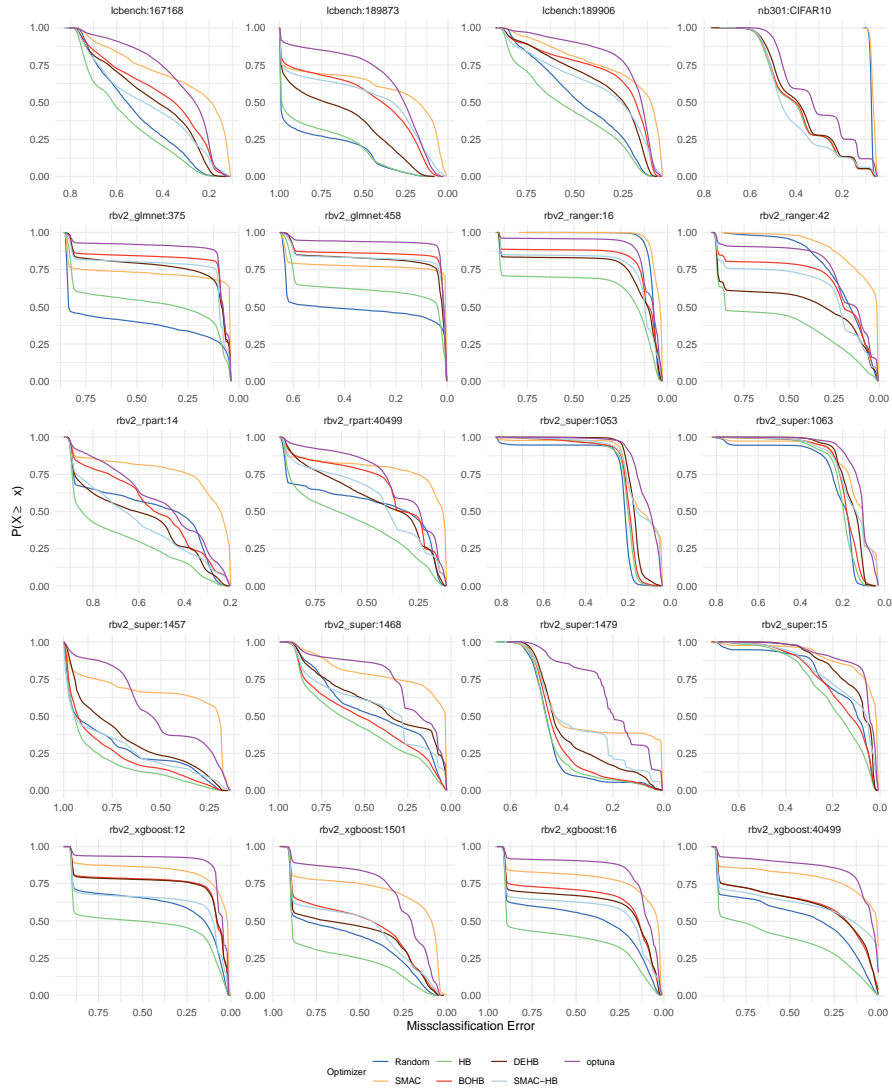


Figure 7: Empirical cumulative distribution function (ECDF) for optimizer traces on YAHPO-SO.

was allowed 10^4 evaluations.

Evaluation. For evaluation, we computed the mean normalized regret for each HPO method separately on the real, surrogate and tabular benchmarks (where the normalized regret for an HPO

Table 6: Consensus Rankings of HPO Methods for Real, Surrogate and Tabular Benchmarks.

Benchmark	Consensus Ranking (CR)	Permutation Order
Real	BO_GP_DF > BO_GP_RS > BO_RF_RS > BO_NN_RS > BO_NN_DF > HB > BO_RF_DF > RS	-
Surrogate	BO_GP_DF > BO_GP_RS > BO_RF_RS > BO_NN_RS > HB > BO_NN_DF > BO_RF_DF > RS	2
Tabular	BO_GP_DF > BO_GP_RS > BO_RF_DF > HB > BO_RF_RS > BO_NN_DF > BO_NN_RS > RS	5

method given a cumulative budget is defined as the difference between the value of the best HPC found by any algorithm and the value of the best HPC found by this method, scaled by the range of objective function values as found by any method, see also [60]). Based on the normalized regret, we also computed the mean rank of each HPO method.

Results for the Branin2D, Currin2D and Hartmann3D benchmark functions are given in Figure 8.

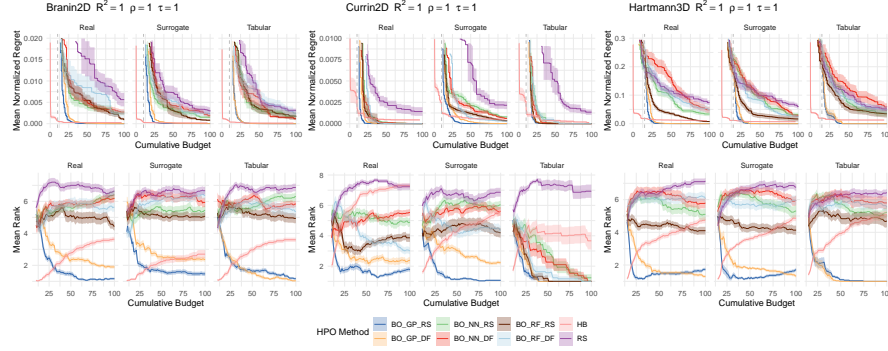


Figure 8: Mean normalized regret (top) and mean ranks (bottom) of different HPO methods on different benchmarks. Ribbons represent standard errors. The gray vertical line indicates the cumulative budget used for the initial design of BO methods. Performance measures of the surrogate benchmarks are stated after the benchmark function. 30 replications.

Differences between tabular and real/surrogate benchmarks can be explained by the fact that the inner optimization problem of BO methods is much easier to solve when only a finite set of potential candidates must be evaluated (i.e., by exhaustive search). We also observe that for the BO performance on the tabular benchmarks, there is no substantial difference in whether the acquisition function optimization is solved exactly or via a Random Search. We employ the rank-based symmetric difference (SD) method which aims to find a consensus ranking that minimizes the average number of rank reversals for the individual benchmark function rankings. We limit ourselves to the scenario of considering the set of all linear orders of HPO methods as candidates for a consensus ranking (SD/L). By comparing the consensus ranking obtained via the surrogate/tabular benchmarks to the consensus ranking obtained using the real benchmarks, we determine the faithfulness of surrogate and tabular benchmarks. We observe that the consensus ranking obtained using the surrogate benchmarks matches the real one more closely than rankings obtained using tabular benchmarks (Table 6).

E.2 Single-Objective Benchmark on YAHPO-SO

Instances and Evaluation Protocol. We use the set of instances and target variables defined for the YAHPO-SO benchmark suite in Supplement C.2 and detailed in Table 3. We furthermore follow the described evaluation protocol, using available search spaces and optimization budgets including 30 replications to assess variance in results. As an evaluation criterion, we report mean normalized

regret (based on the target metric), see Figure 9. Table 7 provides additional info on all optimizers used in the benchmark. Random Search simply samples configurations uniformly at random. SMAC is a model based full-fidelity optimizer using a random forest as surrogate model and Expected Improvement as acquisition function [34]. We use the SMAC4HPO facade [47]. Hyperband randomly samples new configurations and allocates more fidelity to promising configurations by relying on repeated successive halving (SH; [32]). BOHB combines BO with Hyperband and uses a Tree Parzen Estimator (TPE; [5]) as surrogate model. DEHB is a model-free successor of BOHB which relies on differential evolution instead of BO. We use the software defaults regarding the choice of mutation and crossover. SMAC-HB also combines BO with Hyperband but uses a random forest as surrogate model (SMAC4MF facade; [47]). Our optuna optimizer uses a TPE as surrogate model and a median pruner [25] that follows a fixed SH schedule. A configuration is stopped by the pruner if its best intermediate result (at a given fidelity level determined by the SH schedule) is worse compared to the median of the other configurations on the same fidelity level.

Table 7: Optimizers used in the single-objective benchmark.

Optimizer	Software	Reference	Version
Random Search	-	-	-
SMAC (SMAC4HPO)	https://github.com/automl/SMAC3	[47]	1.1.1
Hyperband	https://github.com/automl/HpBandSter	[44]	0.7.4
BOHB	https://github.com/automl/HpBandSter	[21]	0.7.4
DEHB	https://github.com/automl/DEHB	[4]	67ac239
SMAC-HB (SMAC4MF)	https://github.com/automl/SMAC3	[47]	1.1.1
optuna	https://optuna.org/	[2]	2.10.0

E.3 Multi-Objective Benchmark on YAHPO-MO

Instances and Evaluation Protocol. We use the set of instances and target variables defined for the YAHPO-MO benchmark suite in Supplement C.2 and detailed in Table 4. We furthermore follow the described evaluation protocol, using available search spaces and optimization budgets including 30 replications to assess variance in results. As an evaluation criterion, we report the mean Hypervolume Indicator [80] computed on normalized targets (see Figure 10). Nadir points and reference Pareto fronts were obtained empirically over all replications of all HPO methods on a given benchmark instance. Table 8 provides additional info on all optimizers used in the benchmark. Random Search simply samples configurations uniformly at random. Random Search (x4) at each step samples four configurations uniformly at random (in parallel). We include this variant as a strong baseline. ParEGO is a model based optimizer relying on a scalarization of the objectives which we then model using a random forest as surrogate model. As acquisition function we use Expected Improvement [34]. SMS-EGO is a model based optimizer that uses a surrogate model for each objective (again, we use random forests) and proposes candidates based on the S -metric [61]. EHVI is a model based optimizer using a surrogate model for each objective (again, we use random forests) and proposes candidates based on their Expected Hypervolume Improvement [20]. MEGO is a model based optimizer using a surrogate model for each objective (again, we use random forests) and proposes candidates by considering the Expected Improvement for each objective which gives rise to a multi-objective optimization problem of the acquisition functions themselves. For the final candidate selection, we sample uniformly at random over the Pareto optimal (with respect to the Expected Improvements) candidates. MIES is a mixed integer evolutionary optimizer (plus survival scheme, $\mu = \lfloor \text{budget}/6 \rfloor$, $\lambda = \lfloor \mu/4 \rfloor^8$). We use Gaussian mutation ($p = 0.2$)

⁸where budget is the optimization budget for a given instance, i.e., number of total evaluations

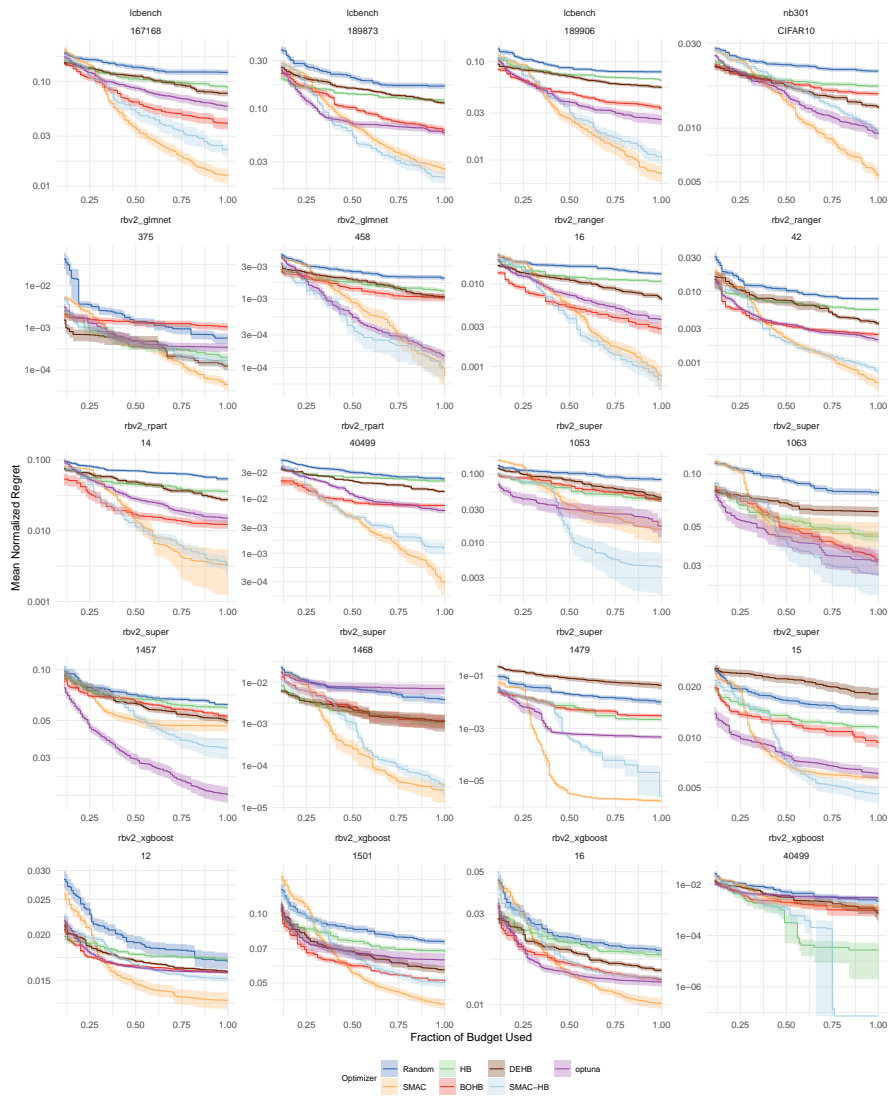


Figure 9: Mean normalized regret of HPO methods separate for each benchmark instance. x-axis starts after 10%.

for numerical parameters and discrete uniform mutation ($p = 0.2$) for categorical parameters. For recombination, we use uniform crossover ($p = 0.2$). As parent selection we perform a tournament selection of parents using nondominated sorting. For survival, we select the best individuals based on nondominated sorting.

Table 8: Optimizers used in the multi-objective benchmark.

Optimizer	Software	Reference	Version
Random Search	-	-	-
Random Search (x4)	-	-	-
ParEGO	https://github.com/mlr-org/mlr3mbo	[40]	1f59e13
SMS-EGO	https://github.com/mlr-org/mlr3mbo	[61]	1f59e13
EHVI	https://github.com/mlr-org/mlr3mbo	[20]	1f59e13
MEGO	https://github.com/mlr-org/mlr3mbo	[33]	1f59e13
MIES	https://github.com/mlr-org/miesmuschel	[46]	3483f11

F Scenarios, Search Spaces and Data Sources

Random Bot V2 (rbv2_)

All scenarios prefixed with *rbv2_* use data described in [9]. Data contains results from several ML algorithms trained across up to 119 datasets evaluated for a large amount of random evaluations. Table 9 lists all hyperparameters of the search space of the *rbv2_* scenarios. Targets are given by accuracy (acc), balanced accuracy (bac), AUC (auc), Brier Score (brier), F1 (f1), log loss (LogLoss), time for training the model (timetrain), and memory usage (memory).

Surrogates are fitted on subsets of the full data available from [9], such that a minimum of 1500 and a maximum of 200000 (depending on the scenario) evaluations are available for each instance in each scenario. All scenarios consist of a pre-processing step (missing data imputation) and a subsequently fitted ML algorithm. Instance ID's correspond to OpenML [71] *dataset* ids through which dataset properties can be queried⁹. OpenML tasks corresponding to each dataset can be obtained from [9]. We abbreviate the *num.impute.selected.cpo* hyperparameter with *imputation* throughout the tables. We fix the *repl* parameter to 10 for experiments.

NAS-Bench-301 (nb301)

nb301 uses data of the NAS-Bench-301 benchmark ([78], see also [65]). Table 10 lists all hyperparameters of the search space of the *nb301* scenario. Targets are given by the validation accuracy (val_accuracy) and the training time (runtime).

LCBench (lcbench)

The *lcbench* collection uses data of the LCBench benchmark [77], as described in [79]. Table 11 lists all hyperparameters of the search space of the *lcbench* scenario. Targets are given by the validation accuracy (val_accuracy), validation cross entropy (val_crossentropy), validation balanced accuracy (val_balanced_accuracy), test cross entropy (test_crossentropy), test balanced accuracy (test_balanced_accuracy) and the training time (time). Instance ID's correspond to OpenML [71] *task* ids through which task properties can be queried¹⁰. The task with the ID 167083 exhibited unnatural learning curves and was therefore excluded.

⁹https://www.openml.org/d/<dataset_id>

¹⁰https://www.openml.org/t/<task_id>

Table 9: Search spaces of YAHPO Gym’s *rbv2_* scenarios. † indicates the parent in case dependencies between hyperparameters exist. The *super* scenario inherits dependencies from previous scenarios, while additional dependencies on the *learner_id* are introduced, indicated by a prefix.

<i>rbv2_glmnet</i>			
Hyperparameter	Type	Range	Info
alpha	continuous	[0, 1]	
s	continuous	[0.001, 1097]	log
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

<i>rbv2_rpart</i>			
Hyperparameter	Type	Range	Info
cp	continuous	[0.001, 1]	log
maxdepth	integer	[1, 30]	
minbucket	integer	[1, 100]	
minsplit	integer	[1, 100]	
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

<i>rbv2_svm</i>			
Hyperparameter	Type	Range	Info
kernel	categorical	{linear, polynomial, radial}	
cost	continuous	[4.5e-05, 2.2e4]	log
gamma	continuous	[4.5e-05, 2.2e4]	log, † kernel
tolerance	continuous	[4.5e-05, 2]	log
degree	integer	[2, 5]	† kernel
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

<i>rbv2_aknn</i>			
Hyperparameter	Type	Range	Info
k	integer	[1, 50]	
distance	categorical	{l2, cosine, ip}	
M	integer	[18, 50]	
ef	integer	[7, 403]	log
ef_construction	integer	[7, 403]	log
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

<i>rbv2_xgboost</i>			
Hyperparameter	Type	Range	Info
booster	categorical	{gblinear, gtree, dart}	
nrounds	integer	[7, 2980]	log
eta	continuous	[0.001, 1]	log, † booster
gamma	continuous	[4.5e-05, 7.4]	log, † booster
lambda	continuous	[0.001, 1097]	log
alpha	continuous	[0.001, 1097]	log
subsample	continuous	[0.1, 1]	
max_depth	integer	[1, 15]	† booster
min_child_weight	continuous	[272, 148.4]	log, † booster
colsample_bytree	continuous	[0.01, 1]	† booster
colsample_bylevel	continuous	[0.01, 1]	† booster
rate_drop	continuous	[0, 1]	† booster
skip_drop	continuous	[0, 1]	† booster
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

<i>rbv2_ranger</i>			
Hyperparameter	Type	Range	Info
num.trees	integer	[1, 2000]	
sample.fraction	continuous	[0.1, 1]	
mtry.power	integer	[0, 1]	
respect.unordered.factors	categorical	{ignore, order, partition}	
min.node.size	integer	[1, 100]	
splitrule	categorical	{gini, extratrees}	
num.random.splits	integer	[1, 100]	† splitrule
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

<i>rbv2_super</i>			
Hyperparameter	Type	Range	Info
svm.kernel	categorical	{linear, polynomial, radial}	
svm.cost	continuous	[4.5e-05, 2.2e4]	log
svm.gamma	continuous	[4.5e-05, 2.2e4]	log
svm.tolerance	continuous	[4.5e-05, 2]	log
svm.degree	integer	[2, 5]	
glmnet.alpha	continuous	[0, 1]	
glmnet.s	continuous	[0.001, 1097]	log
rpart.cp	continuous	[0.001, 1]	log
rpart.maxdepth	integer	[1, 30]	
rpart.minbucket	integer	[1, 100]	
rpart.minsplit	integer	[1, 100]	
ranger.num.trees	integer	[1, 2000]	
ranger.sample.fraction	continuous	[0.1, 1]	
ranger.mtry.power	integer	[0, 1]	
ranger.respect.unordered.factors	categorical	{ignore, order, partition}	
ranger.min.node.size	integer	[1, 100]	
ranger.splitrule	categorical	{gini, extratrees}	
ranger.num.random.splits	integer	[1, 100]	
aknn.k	integer	[1, 50]	
aknn.distance	categorical	{l2, cosine, ip}	
aknn.M	integer	[18, 50]	
aknn.ef	integer	[7, 403]	log
aknn.ef_construction	integer	[7, 403]	log
xgboost.booster	categorical	{gblinear, gtree, dart}	
xgboost.nrounds	integer	[7, 2980]	log
xgboost.eta	continuous	[0.001, 1]	log
xgboost.gamma	continuous	[4.5e-05, 7.4]	log
xgboost.lambda	continuous	[0.001, 1097]	log
xgboost.alpha	continuous	[0.001, 1097]	log
xgboost.subsample	continuous	[0.1, 1]	
xgboost.max_depth	integer	[1, 15]	
xgboost.min_child_weight	continuous	[272, 148.4]	log
xgboost.colsample_bytree	continuous	[0.01, 1]	
xgboost.colsample_bylevel	continuous	[0.01, 1]	
xgboost.rate_drop	continuous	[0, 1]	
xgboost.skip_drop	continuous	[0, 1]	
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	
learner_id	categorical	{aknn, glmnet, ranger, rpart, svm, xgboost}	

Table 10: Search space of the *nb301* scenario. We summarize multiple parameters (using, e.g., {3 – 5} if parameters with suffix 3 through 5 are present).

Hyperparameter	Type	Range	Info
NetworkSelectorDatasetInfo_COLON_darts_COLON_edge_normal_{0-13}	categorical	{max_pool_3x3, avg_pool_3x3, skip_connect, sep_conv_3x3, sep_conv_5x3, dil_conv_3x3, dil_conv_5x3}	
NetworkSelectorDatasetInfo_COLON_darts_COLON_edge_reduce_{0-13}	categorical	{max_pool_3x3, avg_pool_3x3, skip_connect, sep_conv_3x3, sep_conv_5x3, dil_conv_3x3, dil_conv_5x3}	
NetworkSelectorDatasetInfo_COLON_darts_COLON_inputs_node_normal_{3-5}	categorical	{0_1, 0_2, 1_2}	
NetworkSelectorDatasetInfo_COLON_darts_COLON_inputs_node_reduce_{3-5}	categorical	{0_1, 0_2, 1_2}	
epoch	integer	[1, 98]	budget

Table 11: Search space of the *lcbench* scenario.

Hyperparameter	Type	Range	Info
epoch	integer	[1, 52]	budget
batch_size	integer	[16, 512]	log
learning_rate	continuous	[1e-04, 0.1]	log
momentum	continuous	[0.1, 0.9]	
weight_decay	continuous	[1e-05, 0.1]	
num_layers	integer	[1, 5]	
max_units	integer	[64, 1024]	log
max_dropout	continuous	[0, 1]	

Interpretable AutoML (*iaml*_)

All scenarios prefixed with *iaml*_ rely on data that were newly collected by us. Different *mlr3* [41] learners (“*classif.glmnet*”, “*classif.rpart*”, “*classif.ranger*”, “*classif.xgboost*”) were incorporated into an ML pipeline with minimal preprocessing (removing constant features, fixing unseen factor levels during prediction and missing value imputation for factor variables by sampling from non-missing training levels) via *mlr3pipelines* [8]. Hyperparameters of the learners were sampled uniformly at random (for the search spaces, see Table 12) and the ML pipeline performance (classification error - *mmce*, F1 score - *f1*, AUC - *auc*, logloss - *logloss*) was evaluated via 5-fold cross-validation on the following OpenML [71] datasets (dataset id): 40981, 41146, 1489, 1067. Each pipeline was then refitted and used for prediction on the whole data to estimate training and predict time (*timetrain*, *timepredict*) and RAM usage (during training and prediction, *ramtrain* and *rampredict* as well as model size, *rammodel*). Moreover, interpretability measures as described in [52] were computed for all models: number of features used (*nf*), interaction strength of features (*ias*) and main effect complexity of features (*mec*). To our best knowledge, this is the first publicly available benchmark that combines performance, resource usage and interpretability of models allowing for the construction of interesting multi-objective benchmarks. Hyperparameter configurations were evaluated at different fidelity steps (training sizes of the following fractions: 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1) achieved via incorporating resampling in the ML pipeline. The super learner scenario was constructed by using the data of all four base learners introducing conditional hyperparameters in the form of branching. In total, 5451872 different configurations were evaluated. Data collection was performed on the *moran* partition of the *ARCC Teton HPC* cluster of the University of Wyoming using *batchtools* [42] for job scheduling and took around 9.8 CPU years. Surrogate models were then fitted on the available data as described in Supplement D.1. Table 12 lists all hyperparameters of the search spaces of the *iaml*_ scenarios. Instance ID’s correspond to OpenML [71] *dataset* ids through which dataset properties can be queried¹¹.

¹¹https://www.openml.org/d/<dataset_id>

Table 12: Search spaces of YAHPO Gym’s *iaml* scenarios. † indicates the parent in case dependencies between hyperparameters exist. The *super* scenario inherits dependencies from previous scenarios, while additional dependencies on the learner are introduced, indicated by a prefix.

<i>iaml_glmnet</i>				<i>iaml_rpart</i>			
Hyperparameter	Type	Range	Info	Hyperparameter	Range	Type	Info
alpha	continuous	[0, 1]		cp	continuous	[1e-04, 1]	log
s	continuous	[1e-04, 1000]	log	maxdepth	integer	[1, 30]	
trainsize	continuous	[0.03, 1]	budget	minbucket	integer	[1, 100]	
				minsplit	integer	[1, 100]	
				trainsize	continuous	[0.03, 1]	budget

<i>iaml_ranger</i>				<i>iaml_xgboost</i>			
Hyperparameter	Type	Range	Info	Hyperparameter	Type	Range	Info
num.trees	integer	[1, 2000]		booster	categorical	{gblinear, gbtrees, dart}	
replace	boolean	{TRUE, FALSE}		nrounds	integer	[3, 2000]	log
sample.fraction	continuous	[0, 1]		eta	continuous	[1e-04, 1]	log, † booster
mtry.ratio	continuous	[0, 1]		gamma	continuous	[1e-04, 7]	log, † booster
respect.unordered.factors	categorical	{ignore, order, partition}		lambda	continuous	[1e-04, 1000]	log
min.node.size	integer	[1, 100]		alpha	continuous	[1e-04, 1000]	log
splitrule	categorical	{gini, extratrees}		subsample	continuous	[0, 1]	
num.random.splits	integer	[1, 100]	† splitrule	max_depth	integer	[1, 15]	† booster
trainsize	continuous	[0.03, 1]	budget	min_child_weight	continuous	{exp(1), 150}	log, † booster
				colsample_bytree	continuous	[0.01, 1]	† booster
				colsample_bylevel	continuous	[0.01, 1]	† booster
				rate_drop	continuous	[0, 1]	† booster
				skip_drop	continuous	[0, 1]	† booster
				trainsize	continuous	[0.03, 1]	budget

<i>iaml_super</i>			
Hyperparameter	Type	Range	Info
learner	categorical	{glmnet, rpart, ranger, xgboost}	
glmnet.alpha	continuous	[0, 1]	
glmnet.s	continuous	[1e-04, 1000]	log
rpart.cp	continuous	[1e-04, 1]	log
rpart.maxdepth	integer	[1, 30]	
rpart.minbucket	integer	[1, 100]	
rpart.minsplit	integer	[1, 100]	
ranger.num.trees	integer	[1, 2000]	
ranger.replace	boolean	{TRUE, FALSE}	
ranger.sample.fraction	continuous	[0, 1]	
ranger.mtry.ratio	continuous	[0, 1]	
ranger.respect.unordered.factors	categorical	{ignore, order, partition}	
ranger.min.node.size	integer	[1, 100]	
ranger.splitrule	categorical	{gini, extratrees}	
ranger.num.random.splits	integer	[1, 100]	
xgboost.booster	categorical	{gblinear, gbtrees, dart}	
xgboost.nrounds	integer	[3, 2000]	log
xgboost.eta	continuous	[1e-04, 1]	log
xgboost.gamma	continuous	[1e-04, 7]	log
xgboost.lambda	continuous	[1e-04, 1000]	log
xgboost.alpha	continuous	[1e-04, 1000]	log
xgboost.subsample	continuous	[0, 1]	
xgboost.max_depth	integer	[1, 15]	
xgboost.min_child_weight	continuous	{exp(1), 150}	log
xgboost.colsample_bytree	continuous	[0.01, 1]	
xgboost.colsample_bylevel	continuous	[0.01, 1]	
xgboost.rate_drop	continuous	[0, 1]	
xgboost.skip_drop	continuous	[0, 1]	
trainsize	continuous	[0.03, 1]	budget

5.2. A Collection of Quality Diversity Optimization Problems Derived from Hyperparameter Optimization of Machine Learning Models

Contributing article:

L. Schneider, F. Pfisterer, J. Thomas, and B. Bischl. A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2136–2142, 2022. <https://dl.acm.org/doi/10.1145/3520304.3534003>.

Copyright information:

This article is licensed under the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).

Author contributions:

Lennart Schneider is the first author of this manuscript, contributing the following. Lennart Schneider and Florian Pfisterer jointly developed the idea to frame multi-objective HPO problems present in YAHPO Gym as quality diversity optimization problems, drawing inspiration from previous projects they worked on. Lennart Schneider implemented this re-formulation of existing benchmark problems within YAHPO Gym. He further conducted benchmark experiments on these newly constructed problems, analyzed the results, and created all visualizations presented in the manuscript. Lennart Schneider drafted the first version of the manuscript under the supervision of Florian Pfisterer and Janek Thomas. All authors contributed collaboratively to the revision process.

Supplementary material available at:

- Code: https://github.com/slds-lmu/qdo_yahpo
- arXiv version: <https://arxiv.org/abs/2204.14061>



A Collection of Quality Diversity Optimization Problems Derived from Hyperparameter Optimization of Machine Learning Models

Lennart Schneider
LMU Munich
Munich, Germany
lennart.schneider@stat.uni-muenchen.de

Janek Thomas
LMU Munich
Munich, Germany
janek.thomas@stat.uni-muenchen.de

Florian Pfisterer
LMU Munich
Munich, Germany
florian.pfisterer@stat.uni-muenchen.de

Bernd Bischl
LMU Munich
Munich, Germany
bernd.bischl@stat.uni-muenchen.de

ABSTRACT

The goal of Quality Diversity Optimization is to generate a collection of diverse yet high-performing solutions to a given problem at hand. Typical benchmark problems are, for example, finding a repertoire of robot arm configurations or a collection of game playing strategies. In this paper, we propose a set of Quality Diversity Optimization problems that tackle hyperparameter optimization of machine learning models - a so far underexplored application of Quality Diversity Optimization. Our benchmark problems involve novel feature functions, such as interpretability or resource usage of models. To allow for fast and efficient benchmarking, we build upon YAHPO Gym, a recently proposed open source benchmarking suite for hyperparameter optimization that makes use of high performing surrogate models and returns these surrogate model predictions instead of evaluating the true expensive black box function. We present results of an initial experimental study comparing different Quality Diversity optimizers on our benchmark problems. Furthermore, we discuss future directions and challenges of Quality Diversity Optimization in the context of hyperparameter optimization.

CCS CONCEPTS

• Computing methodologies → Search methodologies.

KEYWORDS

Machine Learning, Hyperparameter Optimization, Quality Diversity Optimization, Benchmarking

ACM Reference Format:

Lennart Schneider, Florian Pfisterer, Janek Thomas, and Bernd Bischl. 2022. A Collection of Quality Diversity Optimization Problems Derived from Hyperparameter Optimization of Machine Learning Models. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion)*, July 9–13, 2022, Boston, MA, USA. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3520304.3534003>



This work is licensed under a Creative Commons Attribution International 4.0 License. *GECCO '22 Companion*, July 9–13, 2022, Boston, MA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9268-6/22/07.
<https://doi.org/10.1145/3520304.3534003>

1 INTRODUCTION

Quality Diversity Optimization (QDO) aims to generate a collection of diverse, high-performing solutions. Classical algorithms are Novelty Search with Local Competition [10] and MAP-Elites [13], which rely on the concepts of evolutionary computation. While the development of new QDO algorithms has seen significant progress [6–8], the community lacks larger testbeds of benchmark problems. In this paper, we propose a set of QDO problems derived from hyperparameter optimization (HPO) of machine learning (ML) models. So far, HPO has been an underexplored area for the application of QDO. Our benchmark problems involve feature functions of high practical importance, such as interpretability or resource usage of models, making the application of QDO algorithms particularly relevant due to their illuminating properties. As an example, consider a random forest [4] in a binary classification setting. Depending on the choice of hyperparameters, a random forest can potentially contain many deep trees. On the one hand, we expect such a random forest to yield good performance, on the other hand, a random forests with thousands of deep trees is not interpretable. We could for example be interested in relating the performance of a random forest to interpretability measures such as the number of features the model uses or the interaction strength of features. In general, a random forest that uses few features and has low interaction strength is expected to perform worse than a large and deep forest, but this strongly depends on the concrete data set at hand. It is therefore of high interest to illuminate the relationship of performance and interpretability and QDO algorithms are very well suited for this task.

1.1 Hyperparameter Optimization

An ML *learner* or *inducer* I configured by hyperparameters $\lambda \in \Lambda$ maps a data set $\mathcal{D} \in \mathbb{D}$ to a model \hat{f} , i.e.,

$$I : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}, (\mathcal{D}, \lambda) \mapsto \hat{f}.$$

HPO methods aim to identify a high-performing hyperparameter configuration (HPC) $\lambda \in \tilde{\Lambda}$ for \mathcal{I}_λ [3]. The so-called search space $\tilde{\Lambda} \subset \Lambda$ is typically a subspace of the set of all possible HPCs: $\tilde{\Lambda} = \tilde{\Lambda}_1 \times \tilde{\Lambda}_2 \times \dots \times \tilde{\Lambda}_d$, where $\tilde{\Lambda}_i$ is a bounded subset of the domain of the i -th hyperparameter Λ_i . $\tilde{\Lambda}_i$ can be either continuous, discrete, or categorical. It can also depend on other hyperparameters, meaning that $\tilde{\Lambda}_i$ is only active when $\tilde{\Lambda}_j$ takes certain values, resulting in a possibly hierarchical search space. The classical (single-objective)

HPO problem is defined as:

$$\lambda^* \in \arg \min_{\lambda \in \tilde{\Lambda}} \widehat{GE}(\lambda), \quad (1)$$

i.e., the goal is to minimize the estimated generalization error. This typically involves a costly resampling procedure that can take a significant amount of time (see [3] for further details). $\widehat{GE}(\lambda)$ is a black-box function, as it generally has no closed-form mathematical representation, and analytic gradient information is generally not available. Therefore, the minimization of $\widehat{GE}(\lambda)$ forms an *expensive black-box* optimization problem. Furthermore, it is typically multimodal and noisy, as $\widehat{GE}(\lambda)$ is only a stochastic estimate of the true unknown generalization error.

1.2 Quality Diversity Optimization

The goal of QDO is to generate a collection of diverse yet high-performing solutions. Diversity is defined via behavioral niches on so-called feature functions (behavior functions), whereas performance is defined on an objective function (fitness function). In the following, we assume that a single multi-objective function $f: \Lambda \rightarrow \mathbb{R}^m$, $m \geq 2$ returns both the objective function as well as the feature functions that span the behavior space. Without loss of generality, we assume that the first dimension returned by f is the objective function, whereas the dimensions $2, \dots, m$ are the feature functions. We denote by y_i the i -th output value of $f(\lambda)$. We make few assumptions regarding the nature of niches but note that a uniform grid of k niches is classically constructed by dividing the behavior space into equally sized hyperrectangles [13]. In this case, a niche $N_j \subseteq \mathbb{R}^{m-1}$ is simply defined by lower (l_{ij}) and upper (u_{ij}) boundaries on each feature function $i \in \{2, \dots, m\}$, with $l_{ij} = u_{ij-1}$ for $j \in \{2, \dots, k\}$. A point λ “belongs” to niche N_j if $\forall i \in \{2, \dots, m\} : y_i \in [l_{ij}, u_{ij})$. The best-performing point of a niche is typically referred to as a so-called elite. In this paper, we propose several benchmark instances for QDO problems, which we formally introduce below.

DEFINITION 1 (QDO BENCHMARK PROBLEM). *A QDO benchmark problem consists of a function $f: \Lambda \rightarrow \mathbb{R}^m$, $m \geq 2$ returning both the objective function and feature function values, a bounded search space $\tilde{\Lambda}$, and a set of k behavioral niches $\{N_1, \dots, N_k\}$. $f(\lambda) = \mathbf{y}$ returns m values $\mathbf{y} = (y_1, \dots, y_m)$, where y_1 is the objective function value and y_2, \dots, y_m are feature function values. Based on the niches and their properties (e.g., boundaries), solutions are assigned to these niches and collected in an archive.*

2 PROPOSED BENCHMARKS

We propose a collection of twelve QDO benchmark problems belonging to two different contexts: First, we are interested in illuminating interpretability (measured by the number of features used in a model and their interaction strength) and performance trade-offs of ML models. Second, we examine performance trade-offs with respect to model size (memory requirement) and inference time (i.e., time required to make a prediction). As ML models, we select random forest [4] and extreme gradient boosting (XGBoost) models [5] in binary classification settings. As an objective function, we choose the classification error in the case of formulating a minimization problem or the accuracy (1 - classification error) in the case of

formulating a maximization problem. In the following, we assume that the accuracy is to be maximized. Our benchmark problems rely on YAHPO Gym [15], a recently proposed open source benchmarking suite for HPO. YAHPO Gym provides so-called surrogate-based benchmarks: For a given learner and a given data set, a large amount of HPCs (typically sampled uniformly at random from the search space) were evaluated, and surrogate models have been fitted on these data. When optimizing a benchmark problem (instance) in YAHPO Gym, the costly real black-box evaluation of an HPC is skipped, and instead, the surrogate model’s prediction is returned. This allows for efficient benchmarking of HPO problems, as YAHPO Gym typically requires as little as one millisecond for predicting the performance metric of an HPC and induces minimal memory overhead [15]. In general, performance estimation of ML models is noisy. YAHPO Gym can handle both scenarios of providing a deterministic or noisy HPO benchmark. In the following, we employ YAHPO Gym in the deterministic setting and leave the construction of noisy QDO problems to future research.

We build upon YAHPO Gym’s `iaml_ranger` and `iaml_xgboost` benchmark scenarios. A scenario is a collection of benchmark instances that share the same learner and search space but contain multiple data sets. Both `iaml_ranger` and `iaml_xgboost` allow for HPO on four different data sets with the following OpenML [19] IDs: 41146, 40981, 1489, 1067¹.

2.1 Accuracy and Interpretability

Eight of our proposed QDO benchmark problems belong to the context of performance and interpretability. We suggest benchmarking all instances of the `iaml_ranger` and `iaml_xgboost` scenarios because previous benchmarks on YAHPO Gym have shown that these are interesting problems with relevant search spaces. The search spaces (genotype spaces) are given in Table 1 and Table 2. These search spaces are specifically tailored for existing QDO algorithms and implementations, which often do not support categorical parameters or hierarchical dependencies but only numeric parameters. We discuss this limitation in Section 3. The objective function is given by the accuracy that is to be maximized. The behavior space is spanned by two feature functions: the number of features (NF) used by a model and their interaction strength (IAS). These measures were proposed to quantify interpretability for any ML model in a model-agnostic way [12]. A feature is regarded as being used by a model if changing the feature changes the prediction of the model. NF is then estimated via a sampling procedure as described in Algorithm 1 of [12]. The IAS is based on accumulated local effects (ALE) [1] of a model and is given by the scaled approximation error between the ALE main effect model (the sum of first-order ALE effects) and the prediction function of the model. For more details, see Section 3.2 of [12]. A model with low NF and IAS is more interpretable than a model with high NF or high IAS, but optimal performance is often reflected in either a high NF or IAS or both. However, this strongly depends on the data set at hand; as ML models can always overfit, resulting in bad generalization performance, simply opting for models having a high NF and IAS is prone to poor performance. It is therefore a priori unknown

¹These IDs correspond to OpenML data set ids through which data set properties can be queried via <https://www.openml.org/d/<id>>.

whether low or high NF / IAS will result in good performance, making the application of QDO algorithms particularly attractive due to the illuminating aspect of QDO. In line with most QDO algorithms and their implementations, we propose to define niches, using a discrete uniform grid archive with ranges $[0, p]$ for NF and $[0, 1]$ for IAS. Here, p is the number of features present in a data set (e.g., 14 for data set 40981). We propose to use a bin size of $p + 1$ for NF (i.e., models in each bin use exactly $0, 1, \dots$ or p features) and 100 for IAS, resulting in 2100, 1500, 600, and 2200 niches overall for the data sets above. We summarize all benchmark problems in Table 3. In Figure 1, we visualize the exemplary `iaml_ranger_40981` problem via a heatmap of the accuracy (color) of different elites, which allows for visually inspecting the alignment of the feature functions and objective function. We present similar heatmaps for all remaining benchmark problems in Appendix A.

2.2 Accuracy and Resource Usage

Four of our proposed QDO problems belong to the context of performance and resource usage. We suggest benchmarking all instances of the `iaml_ranger` scenario. The search space is again given in Table 1. The objective function is given by the accuracy that is to be maximized. The behavior space is spanned by two feature functions: the size of the model if stored on disk (in MB) and the time required to make a prediction on the same data set the model has been trained on (in seconds). These measures are typically of interest in the context of deploying ML models in production environments on different hardware with varying restrictions regarding RAM and latency of predictions [14]. A larger random forest typically consists of plenty and deep trees, and prediction time is both affected by the depth of the individual trees and the number of trees in the forest. In general, a large random forest with deep trees should result in a better performance, but this again depends on the data set at hand. As before, we propose to define niches via a discrete uniform grid archive (for more details, see Table 3). In Figure 1, we visualize the exemplary `iaml_ranger_41146` problem via a heatmap of the accuracy (color) of different elites, which allows for visually inspecting the alignment of the feature functions and objective function. We present similar heatmaps for all remaining benchmark problems in Appendix A.

Table 1: Search space of `iaml_ranger` scenarios.

Hyperparameter	Type	Range	Trafo
<code>num.trees</code>	integer	[1, 2000]	
<code>mtry.ratio</code>	continuous	[0, 1]	
<code>min.node.size</code>	integer	[1, 100]	
<code>sample.fraction</code>	continuous	[0.1, 1]	

2.3 Benchmark Experiments and Results

We benchmark a basic variant of MAP-Elites [13], a basic variant of CMA-ME [8], and Random Search (i.e., sampling HPCs uniformly at random). To illustrate the potential of fast prototyping of new methods, we also include an optimizer that is a mix between MAP-Elites and CMA-ME, where half of the evaluations of a batch are proposed using a Gaussian emitter and the other half using an improvement

Table 2: Search space of `iaml_xgboost` scenarios.

Hyperparameter	Type	Range	Trafo
<code>alpha</code>	continuous	[1e-04, 1000]	log
<code>lambda</code>	continuous	[1e-04, 1000]	log
<code>nrounds</code>	integer	[3, 2000]	log
<code>subsample</code>	continuous	[0.1, 1]	
<code>colsample_bylevel</code>	continuous	[0.01, 1]	
<code>colsample_bytree</code>	continuous	[0.01, 1]	
<code>eta</code>	continuous	[1e-04, 1]	log
<code>gamma</code>	continuous	[1e-04, 7]	log
<code>max_depth</code>	integer	[1, 15]	
<code>min_child_weight</code>	continuous	[exp(1), 150]	log

"log" in the Trafo column indicates that this parameter is optimized on a logarithmic scale, i.e., the range is given by $[\log(\text{lower}), \log(\text{upper})]$, and values are re-transformed via the exponential function prior to evaluating the HPC.

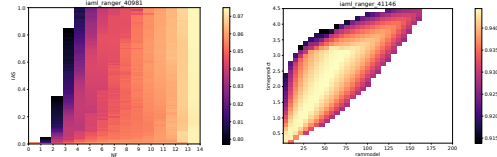


Figure 1: Heatmaps of the `iaml_ranger_40981` interpretability (left) and `iaml_ranger_41146` resource usage (right) benchmark problems. The color gradient represents the accuracy (objective function), with black indicating lowest accuracy and bright yellow/white indicating highest accuracy.

emitter, which we abbreviate as "Gauss.+Imp.". We implement all optimizers in `pyribs` [18]. All optimizer use a batch size of 100 and run for 1000 iterations, resulting in 100000 evaluations in total. For MAP-Elites, we use $\sigma_0 = 0.1$ as the standard deviation of the Gaussian distribution. For CMA-ME, we set $\sigma_0 = 0.1$ as the initial step size and use "filter" as a selection rule. All optimizers use a single emitter. We want to note that we did not perform any tuning of hyperparameters of the optimizers themselves. Therefore, results are preliminary and better performance of an optimizer could be obtained by more carefully chosen design choices. All runs are replicated ten times with different random seeds. We always normalize all parameters of the search space to the unit cube and optimize within these normalized bounds (during evaluation of an HPC, parameters are then re-transformed to their original scale). Note that integer-valued parameters are treated as continuous. However, prior to evaluating an HPC, those parameter values are then rounded to the nearest integer. As performance metrics, we report the coverage (percent of niches occupied with a solution), the QD-Score [16] (sum of best accuracies of occupied niches), and the overall best accuracy found. Results for the interpretability context are given in Table 4 for the `iaml_ranger` scenarios and in Table 5 for the `iaml_xgboost` scenarios. Results for the resource usage context are given in Table 6. We also visualize the anytime QD-Score in Figure 2, Figure 3, and Figure 4. In general, MAP-Elites

Table 3: Summary of our QDO benchmark problems.

Problem	Learner	Data set	Search space	Objective	Features	Niches
Illuminating interpretability						
iaml_ranger_41146	rf	41146	Table 1	acc	NF + IAS	uniform on $[0, 20] \times [0, 1]$ 21, 100 bins
iaml_ranger_40981	rf	40981	Table 1	acc	NF + IAS	uniform on $[0, 14] \times [0, 1]$ 15, 100 bins
iaml_ranger_1489	rf	1489	Table 1	acc	NF + IAS	uniform on $[0, 5] \times [0, 1]$ 6, 100 bins
iaml_ranger_1067	rf	1067	Table 1	acc	NF + IAS	uniform on $[0, 21] \times [0, 1]$ 22, 100 bins
iaml_xgboost_41146	XGBoost	41146	Table 2	acc	NF + IAS	uniform on $[0, 20] \times [0, 1]$ 21, 100 bins
iaml_xgboost_40981	XGBoost	40981	Table 2	acc	NF + IAS	uniform on $[0, 14] \times [0, 1]$ 15, 100 bins
iaml_xgboost_1489	XGBoost	1489	Table 2	acc	NF + IAS	uniform on $[0, 5] \times [0, 1]$ 6, 100 bins
iaml_xgboost_1067	XGBoost	1067	Table 2	acc	NF + IAS	uniform on $[0, 21] \times [0, 1]$ 22, 100 bins
Illuminating resource usage						
iaml_ranger_41146	rf	41146	Table 1	acc	rm + tp	uniform on $[1, 200] \times [0.19, 4.5]$ 33, 33 bins
iaml_ranger_40981	rf	40981	Table 1	acc	rm + tp	uniform on $[1, 40] \times [0.10, 0.65]$ 33, 33 bins
iaml_ranger_1489	rf	1489	Table 1	acc	rm + tp	uniform on $[1, 200] \times [0.19, 4.5]$ 33, 33 bins
iaml_ranger_1067	rf	1067	Table 1	acc	rm + tp	uniform on $[1, 78] \times [0.13, 1.55]$ 33, 33 bins

"rf" = random forest. "acc" = accuracy. "rm" = rammodel. "tp" = timepredict. Niches of the uniform grid archive are constructed by dividing each dimension of the feature space as indicated in the Niches column into equally spaced bins, forming hyperrectangles.

shows the strongest performance but is sometimes outperformed by the Gauss.+Imp. optimizer. The performance of CMA-ME varies strongly between benchmark problems, whereas MAP-Elites and Gauss.+Imp. appear to be more consistent in their performance. Random Search performs overall poorly, indicating that structural information of the problems can be efficiently leveraged by more sophisticated optimizers. We release all our code for running the benchmarks and analyzing results via the following GitHub repository: https://github.com/slds-lmu/qdo_yahpo.

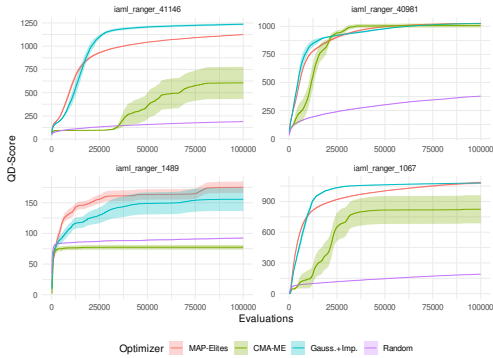


Figure 2: Anytime QD-Score for the iaml_ranger benchmarks (interpretability). Averaged over 10 replications. Ribbons represent standard errors.

3 OUTLOOK

The benchmark problems proposed in this paper pose novel applications for QDO methods. QDO is interesting for HPO problems

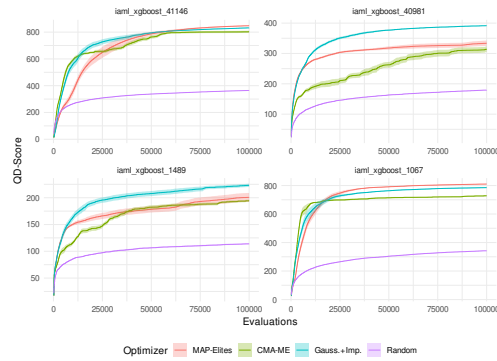


Figure 3: Anytime QD-Score for the iaml_xgboost benchmarks (interpretability). Averaged over 10 replications. Ribbons represent standard errors.

in contexts where users are interested in criteria that go beyond traditional performance metrics. This can be highly relevant in the context of ML models that should be interpretable to a certain degree or ML models that must be deployed across a diverse set of systems, ranging from FPGAs to compute clusters. We propose several such benchmarks but hope that HPO problems for QDO can go far beyond what is proposed in the context of this work. Furthermore, we identify two properties of existing HPO algorithms and implementation that might help to drive greater adaptation of QDO methods.

Table 4: Benchmark results on iaml_ranger (interpretability).

	data set: 41146			data set: 40981		
Algorithm	Coverage %	QD-Score	Max Acc	Coverage %	QD-Score	Max Acc
MAP-Elites	58.95	1124.69	0.9444	80.91	1026.84	0.8747
CMA-ME	31.43	604.66	0.9444	79.03	1007.62	0.8745
Gauss.+Imp.	64.61	1235.72	0.9444	80.63	1026.35	0.8746
Random	9.69	188.52	0.9443	29.39	378.26	0.8746
	data set: 1489			data set: 1067		
Algorithm	Coverage %	QD-Score	Max Acc	Coverage %	QD-Score	Max Acc
MAP-Elites	34.73	175.24	0.9112	58.10	1083.08	0.8706
CMA-ME	14.57	77.41	0.9109	43.87	823.86	0.8700
Gauss.+Imp.	30.83	155.62	0.9111	57.57	1080.00	0.8705
Random	17.63	92.51	0.9109	10.22	191.19	0.8705

Averaged over 10 replications. "Max Acc" = maximum accuracy. Best performance is highlighted in bold.

Table 5: Benchmark results on iaml_xgboost (interpretability).

	data set: 41146			data set: 40981		
Algorithm	Coverage %	QD-Score	Max Acc	Coverage %	QD-Score	Max Acc
MAP-Elites	47.94	847.67	0.9446	26.00	333.70	0.8870
CMA-ME	45.38	802.77	0.9444	24.61	314.66	0.8849
Gauss.+Imp.	47.03	831.91	0.9446	30.76	391.40	0.8868
Random	20.10	364.31	0.9432	14.07	179.34	0.8822
	data set: 1489			data set: 1067		
Algorithm	Coverage %	QD-Score	Max Acc	Coverage %	QD-Score	Max Acc
MAP-Elites	38.85	200.69	0.9147	42.90	810.19	0.8824
CMA-ME	37.92	194.44	0.9132	38.47	728.03	0.8814
Gauss.+Imp.	43.52	223.11	0.9147	41.55	785.12	0.8821
Random	22.55	114.04	0.9052	18.19	342.40	0.8736

Averaged over 10 replications. "Max Acc" = maximum accuracy. Best performance is highlighted in bold.

Table 6: Benchmark results on iaml_ranger (resource usage).

	data set: 41146			data set: 40981		
Algorithm	Coverage %	QD-Score	Max Acc	Coverage %	QD-Score	Max Acc
MAP-Elites	41.50	422.99	0.9444	29.44	279.65	0.8749
CMA-ME	38.82	395.85	0.9444	27.29	259.23	0.8749
Gauss.+Imp.	41.36	421.55	0.9444	29.30	278.35	0.8749
Random	31.25	318.20	0.9443	24.92	236.56	0.8746
	data set: 1489			data set: 1067		
Algorithm	Coverage %	QD-Score	Max Acc	Coverage %	QD-Score	Max Acc
MAP-Elites	44.52	435.75	0.9112	41.04	387.44	0.8709
CMA-ME	42.84	419.21	0.9112	39.53	373.20	0.8709
Gauss.+Imp.	44.34	434.06	0.9112	40.77	384.92	0.8709
Random	36.74	358.33	0.9109	38.06	358.90	0.8705

Averaged over 10 replications. "Max Acc" = maximum accuracy. Best performance is highlighted in bold.

3.1 Mixed Search Spaces

Available implementations of QDO methods, such as pyrribs, consider only continuous search spaces. In many practical applications,

search spaces include categorical and conditionally active hyperparameters – so-called hierarchical, mixed search spaces [17]. While existing methods can theoretically be extended to such spaces, a lack of available implementations might inhibit adoption of QDO

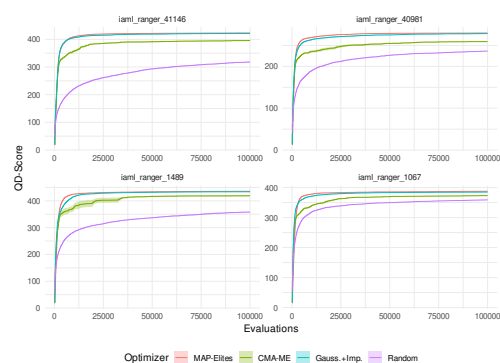


Figure 4: Anytime QD-Score for the iaml_ranger benchmarks (resource usage). Averaged over 10 replications. Ribbons represent standard errors.

methods. Simultaneously, this might also open up a richer set of readily available problem instances, and our proposed benchmarks can be trivially extended to include such search spaces ².

3.2 Overlapping Niches

Similarly, while not formally required, niches in the QDO literature have traditionally been defined as being disjoint, i.e., a point can only belong to a single niche. Real-world applications in the context of HPO provide an interesting field of study for methods that go beyond those assumptions. As an example, consider again the setting where an ML model should be deployed across diverse hardware. In this context, niches could overlap, as a smaller model (e.g., fitting on an FPGA or microcontroller) is always also a candidate for larger devices (e.g., fitting on a laptop or workstation).

The field of hardware-aware neural architecture search [2] seeks to find neural network architectures that are optimal for specific hardware. If such models are now required for a variety of different hardware architectures, QDO could be applied to find models for each of those niches while simultaneously exploiting similarities between architectures. In this setting, niches cannot be defined on simple feature functions, as a solution can fall into multiple niches that defy definition based on simple rules.

Such applications form an interesting new playground for QDO methods, especially when considering that HPO problems are usually considered to be *expensive*. This might provide a valuable new avenue for future method development towards approaches that require fewer function evaluations [9] or that can efficiently make use of evaluations at multiple fidelity levels [11].

²for the full search spaces of our benchmark problems, see [15]

ACKNOWLEDGMENTS

The authors of this work take full responsibilities for its content. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8). This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A.

REFERENCES

- [1] Daniel W. Apley and Jingyu Zhu. 2020. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82, 4 (2020), 1059–1086.
- [2] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. 2021. Hardware-Aware Neural Architecture Search: Survey and Taxonomy. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Zhi-Hua Zhou (Ed.), 4322–4329.
- [3] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. 2021. Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges. *arXiv:2107.05847 [cs, stat]* (2021).
- [4] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
- [5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.
- [6] Antoine Cully. 2021. Multi-Emitter MAP-Elites: Improving Quality, Diversity and Data Efficiency with Heterogeneous Sets of Emitters. In *Proceedings of the 2021 Genetic and Evolutionary Computation Conference*, 84–92.
- [7] Matthew C. Fontaine and Stefanos Nikolaidis. 2021. Differentiable Quality Diversity. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34, 10040–10052.
- [8] Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. 2020. Covariance Matrix Adaptation for the Rapid Illumination of Behavior Space. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 94–102.
- [9] Paul Kent and Juergen Branke. 2020. BOP-Elites, a Bayesian Optimisation Algorithm for Quality-Diversity Search. *arXiv:2005.04320 [cs, math, stat]* (2020).
- [10] Joel Lehman and Kenneth O. Stanley. 2011. Evolving a Diversity of Virtual Creatures Through Novelty Search and Local Competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 211–218.
- [11] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.
- [12] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. 2020. Quantifying Model Complexity via Functional Decomposition for Better Post-Hoc Interpretability. In *Machine Learning and Knowledge Discovery in Databases*, Peggy Cellier and Kurt Driessens (Eds.), 193–204.
- [13] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating Search Spaces by Mapping Elites. *arXiv:1504.04909 [cs, q-bio]* (2015).
- [14] Feng Nan, Joseph Wang, and Venkatesh Saligrama. 2016. Pruning Random Forests for Prediction on a Budget. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2342–2350.
- [15] Florian Pfisterer, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. 2021. YAHPO Gym – An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization. *arXiv:2109.03670 [cs, stat]* (2021).
- [16] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. 2016. An Extended Study of Quality Diversity Algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, 19–20.
- [17] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 847–855.
- [18] Bryon Tjanaka, Matthew C. Fontaine, Yulun Zhang, Sam Sommerer, Nathan Dennler, and Stefanos Nikolaidis. 2021. pyribs: A Bare-Bones Python Library for Quality Diversity Optimization. <https://github.com/icaros-usc/pyribs>.
- [19] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explor.* 15, 2 (2013), 49–60.

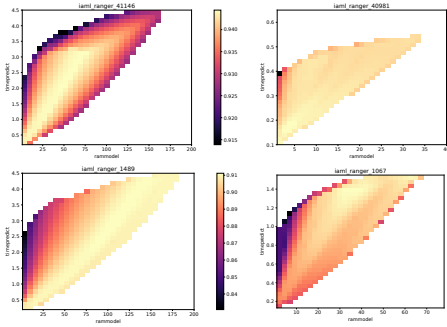


Figure 7: Heatmaps of the iaml_ranger resource usage benchmark problems.

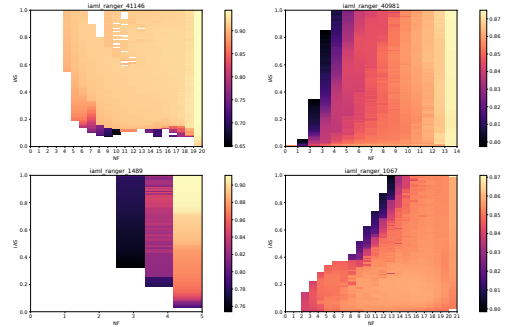


Figure 5: Heatmaps of the iaml_ranger interpretability benchmark problems.

A HEATMAPS FOR ALL BENCHMARK PROBLEMS

In this section, we provide heatmaps for all our benchmark problems. Three Gaussian emitters with a standard deviation of 0.1, 0.2 or 0.3 and one random emitter were run each with a batch size of 25 for 10000 iterations resulting in 1000000 evaluations in total. The color gradient represents the accuracy (objective function), with black indicating lowest accuracy and bright yellow/white indicating highest accuracy. Figure 5 shows heatmaps for the iaml_ranger interpretability benchmark problems. Figure 6 shows heatmaps for the iaml_xgboost interpretability benchmark problems. Figure 7 shows heatmaps for the iaml_ranger resource usage benchmark problems.

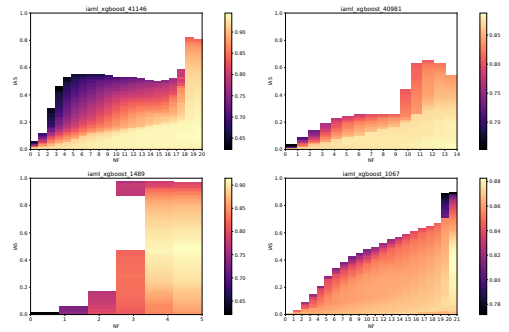


Figure 6: Heatmaps of the iaml_xgboost interpretability benchmark problems.

B TECHNICAL DETAILS

Benchmark experiments were conducted on a single Intel Core i7-10510U CPU and took around 7 hours in total. YAHPO Gym v1.0 was used. For more details on how the iaml_ranger and iaml_xgboost benchmark scenarios of YAHPO Gym were constructed, please see [15].

5.3. Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers

Contributing article:

J. Moosbauer*, M. Binder*, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers. *IEEE Transactions on Evolutionary Computation*, 26(6):1336–1350, 2022. <https://ieeexplore.ieee.org/document/9913342>.

Copyright information:

This article is licensed under the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).

Author contributions:

Lennart Schneider served as a co-author of this manuscript, contributing the following. The idea for the algorithmic framework (SMASHY) considered in the project originated from Martin Binder and Bernd Bischl, with input from Julia Moosbauer. The idea of putting it into context as automatic configuration and analysis of algorithm components for hyperparameter optimizers originated from Julia Moosbauer and Martin Binder. Code for the algorithmic framework was developed by Martin Binder based on a previous project. The code for the experiments for AC was written by Marc Becker and Martin Binder. Benchmarks of competing state-of-the-art algorithm implementations were conducted by Julia Moosbauer. Code for the experiments for the algorithm analysis and the analysis of artifacts was written by Lennart Schneider (ablation studies), Martin Binder, and Julia Moosbauer. The manuscript was written by Julia Moosbauer and Martin Binder, with refinement from Lennart Schneider, Lars Kotthoff, Marc Becker, Florian Pfisterer, and Bernd Bischl. Lennart Schneider, Martin Binder, Florian Pfisterer, Michel Lang, Lars Kotthoff, and Bernd Bischl advised throughout the whole project.

Supplementary material available at:

- Code: https://github.com/slds-lmu/paper_2021_benchmarking_special_issue
- R package: <https://github.com/mlr-org/smashy>
- arXiv version: <https://arxiv.org/abs/2111.14756>

Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers

Julia Moosbauer¹, Martin Binder, Lennart Schneider², Florian Pfisterer³, Marc Becker, Michel Lang, Lars Kotthoff⁴, and Bernd Bischl⁵

Abstract—Automated hyperparameter optimization (HPO) has gained great popularity and is an important component of most automated machine learning frameworks. However, the process of designing HPO algorithms is still an unsystematic and manual process: new algorithms are often built on top of prior work, where limitations are identified and improvements are proposed. Even though this approach is guided by expert knowledge, it is still somewhat arbitrary. The process rarely allows for gaining a holistic understanding of which algorithmic components drive performance and carries the risk of overlooking good algorithmic design choices. We present a principled approach to automated benchmark-driven algorithm design applied to multifidelity HPO (MF-HPO). First, we formalize a rich space of MF-HPO candidates that includes, but is not limited to, common existing HPO algorithms and then present a configurable framework covering this space. To find the best candidate automatically and systematically, we follow a programming-by-optimization approach and search over the space of algorithm candidates via Bayesian optimization. We challenge whether the found design choices are necessary or could be replaced by more naive and simpler ones by performing an ablation analysis. We observe that using a relatively simple configuration (in some ways, simpler than established methods) performs very well as long as some critical configuration parameters are set to the right value.

Index Terms—Algorithm analysis, algorithm design, automated machine learning (AutoML), hyperparameter optimization (HPO), multifidelity.

Manuscript received 21 September 2021; revised 15 March 2022 and 28 July 2022; accepted 15 September 2022. Date of publication 6 October 2022; date of current version 1 December 2022. The work of Lennart Schneider was supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics-Data-Applications (ADACenter) within the Framework of BAYERN DIGITAL II under Grant 20-3410-2-9-8. The work of Michel Lang was supported in part by the Research Center Trustworthy Data Science and Security (<https://rc-trust.ai>). The work of Lars Kotthoff was supported by NSF under Grant 1813537. (Julia Moosbauer and Martin Binder contributed equally to this work.) (Corresponding author: Julia Moosbauer.)

Julia Moosbauer, Martin Binder, Lennart Schneider, Florian Pfisterer, Marc Becker, and Bernd Bischl are with the Department of Statistics, Ludwig-Maximilians-Universität München, 80539 Munich, Germany (e-mail: julia.moosbauer@stat.uni-muenchen.de; martin.binder@stat.uni-muenchen.de; lennart.schneider@stat.uni-muenchen.de; florian.pfisterer@stat.uni-muenchen.de; marc.becker@stat.uni-muenchen.de; bernd.bischl@stat.uni-muenchen.de).

Michel Lang is with the Research Center Trustworthy Data Science and Security, 44227 Dortmund, Germany (e-mail: michel.lang@tu-dortmund.de).

Lars Kotthoff is with the Department of Computer Science, University of Wyoming, Laramie, WY 82071 USA (e-mail: larsko@uwyo.edu).

This article has supplementary downloadable material available at <https://doi.org/10.1109/TEVC.2022.3211336>, provided by the authors.

Digital Object Identifier 10.1109/TEVC.2022.3211336

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see <https://creativecommons.org/licenses/by/4.0/>

I. INTRODUCTION

MACHINE learning (ML) is, in many regards, an optimization problem, and many ML methods can be expressed as algorithms that perform loss minimization with respect to a given objective function. The higher-level task of selecting the ML method and its configuration is often framed as an optimization problem as well, sometimes referred to as a hyperparameter optimization (HPO) [1] or combined algorithm selection and HPO (CASH) problem [2]. Successfully addressing this problem can lead to large performance gains compared to simply using defaults, and in the context of automated ML (AutoML), the use of HPO can make ML more accessible to nonexperts. Because of their potential benefits to ML performance and usability, it is of particular interest to design optimization algorithms that perform particularly well on the HPO problem.

Optimization problems arise in many fields of science and engineering, but as the no-free-lunch theorem states, there is no one optimization algorithm that solves all problems equally well [3]. To design suitable optimizers, it is therefore important to understand the characteristics of HPO.

- 1) *Black-Box*: The objective usually provides no analytical information [4], such as a gradient. Thus, the application of many traditional optimization methods, such as BFGS, is rendered inappropriate or at least questionable.
- 2) *Complex Search Space*: The search space of the optimization problem is often high-dimensional and may contain continuous, integer-valued, and categorical dimensions. Often, there are dependencies between dimensions or even specific hyperparameter values [5].
- 3) *Expensive*: A single evaluation of the objective function may take hours or days. Thus, the total number of possible function evaluations is often severely limited [4].
- 4) *Low-Fidelity Approximations Possible*: An approximation of the true objective value at lower expense can often be obtained, for example, through a partial evaluation [6].
- 5) *Low Effective Dimensionality*: The landscape of the objective function can usually be approximated well by a function of a small subset of all dimensions [7].

Recent HPO and AutoML research has focused on finding and improving optimization algorithms that work particularly well under these conditions. A common approach is to tackle HPO by estimating a local or global structure of the objective landscape by some form of the predictive model.

This introduces additional overhead and complexity with the aim of reducing the overall number of expensive objective evaluations necessary to find an approximate optimum. Typical representatives of this approach are Bayesian optimization (BO) [8] algorithms and frameworks based on BO, which are global optimization schemes based on a nonlinear regression model, e.g., a Gaussian process or random forest. They have shown significant improvements in performance compared to other methods [9] but carry a significant overhead. Furthermore, BO is somewhat difficult to parallelize due to its sequential nature, although many variants exist (e.g., [10], [11], [12], and [13]).

Multifidelity HPO (MF-HPO) algorithms aim to accelerate the optimization process by exploiting cheaper proxy functions of the objective function itself (e.g., by training ML models on a smaller subsample of the available training data, or by running fewer training iterations). Bandit-based algorithms like Hyperband (HB) [14] have become particularly popular because of their good tradeoff between optimization performance and simplicity.

Progress in the field of HPO often consists of iterative improvements of established algorithms. Considerable work exists, for example, to improve the limitations of HB: asynchronous successive halving (ASHA) [15] proposes a sophisticated way to make efficient use of parallel resources, BO HB (BOHB) [16] improves performance during later parts of a run by incorporating surrogate assistance into HB, and asynchronous BOHB (A-BOHB) [17] unites a bandit-based optimization scheme using model-based guidance with asynchronous parallelization.

While these conceptual extensions of HPO all have their respective merit, it is often somewhat overlooked that the simplicity of an optimization algorithm (i.e., how difficult modifications and extensions are, and on how many dependencies a system relies [18]) heavily influences its adoption in practice. Random search (RS), for example, still enjoys great popularity, as it is extremely simple to implement and parallelize, has almost no overhead, and is able to take advantage of the aforementioned low effective dimensionality [7]. Furthermore, algorithmic developments identify and address limitations of prior research, but rarely question core algorithmic choices that have been made in the original implementation. Many multifidelity algorithms, for example, are extensions and further developments of HB that take the fixed successive halving (SH) schedule [19] for granted. The process of designing a good MF-HPO optimizer in practice—and many other algorithmic solutions in science in general—can therefore often feel somewhat like a “manual stochastic local search on the meta level.” The drawback of this manual procedure is that the design space of all HPO algorithms is not systematically searched, and parts of the design space are excluded by prior algorithmic decisions. If “established” algorithms are not challenged, there is a risk that algorithms that work well will be overlooked, and it is often hard to identify what algorithmic components make a difference. In particular, it is possible that overly complicated algorithms are developed by extending “established” designs, only some of which contribute meaningfully to performance

gains. Sometimes certain technical components of an algorithm, which are neither exposed nor discussed in detail, may also influence performance significantly.

A. Contributions

We make a principled demonstration of how HPO algorithm design can be performed systematically and automatically with a benchmark-driven approach following the programming-by-optimization paradigm [20]. In particular, the contributions of this work are as follows.

- 1) *Formalization*: We formalize the design space of MF-HPO algorithms and demonstrate that established MF-HPO algorithms represent instances within this space.
- 2) *Framework*: Based on this formalization, we present a rich, configurable framework for MF-HPO algorithms, whose software implementation we call surrogate model-assisted HB (SMASHY).
- 3) *Configuration*: Based on the formalization and framework, we follow an empirical approach to design an MF-HPO algorithm by optimization, given a large benchmark suite. This configuration procedure does not only consider performance but also, e.g., the simplicity of the design.
- 4) *Benchmark*: As in general any HPO algorithm will be applied in a diverse set of application scenarios, we evaluate the performance of our newly designed algorithm on a representative set of problems that were not previously used for its configuration (i.e., a clean test-set approach on the meta-level) and compare them with established implementations of HPO methods.
- 5) *Explanation*: For the resulting MF-HPO system, we systematically assess and explain the effect of different design choices on overall algorithmic performance. Furthermore, we investigate the behavior of algorithmic design components in the context of specific problem scenarios; i.e., we investigate which algorithmic components lead to performance improvements for simple HPO with numeric hyperparameters, AutoML pipeline configuration, and neural architecture search.

II. RELATED WORK

HPO is one of the most essential components of current AutoML methods [1], and MF-HPO has recently become more prominent, given that cheap, low-fidelity evaluations have proven useful to speed up optimization, especially for expensive HPO of complex ML algorithms on larger data sets [14]. While AutoML tools have historically relied on a limited set of HPO methods, we argue that the optimal HPO method depends on problem characteristics, and therefore a systematic development of HPO methods under consideration of problem characteristics is required. Approaches toward such systematic development have often relied on a *high-level* language or template that allows expressing solution algorithms for a given problem class, e.g., to solve constraint satisfaction problems [21], [22], [23], satisfiability problems [24], scheduling problems [25], or general multiobjective combinatorial problems [26], [27].

Even if a high-level language is available, manual configuration of such frameworks is laborious and requires expert knowledge. This motivates the design philosophy of “programming by optimization” [20] (PBO), which advocates for allowing algorithmic choices in a software system (instead of fixing them at the time of implementation) and automatic configuration by optimization for a given problem context.

As one approach to automatic and efficient algorithm configuration, racing-based strategies have been used to design optimization algorithms. For example, iterated F-RACE [28] has been used for the automatic design of multiobjective ant colony optimization algorithms [26]. Similarly, IRACE [29] has been used for the automatic design multiobjective evolutionary algorithms [27] or to meta-configure the parameters IRACE itself [30]. Another commonly used framework is SMAC [5], which extends the sequential model-based optimization paradigm (SMBO, see also Section IV-A2) to an algorithm configuration setting. This is achieved through the use of an intensification procedure that governs across how many problem instances each configuration is evaluated, trading off computational cost against confidence regarding the superiority of a given configuration. While such intensification mechanisms have been used in other work before [31], [32], SMAC also uses instance features describing properties of a problem instance are used to train the empirical performance model predicting the performance of a configuration on a new problem instance. Besides racing and sequential model-based approaches, genetic algorithms have also been used to evolve optimal solvers [33].

We argue that the design of HPO algorithms can be seen as an instance of PBO. However, while there are many approaches that focus on individual algorithmic choices (e.g., the choice of a surrogate model for BO [34]), we are not aware of many cases where PBO is applied to designing HPO systems themselves. One exception is [35], who use SMACv3 [36] to automatically configure BO for HPO from a flexible search space of components. We take a similar approach here in that the algorithmic choices are exposed as hyperparameters that can be tuned. However, unlike [35], we do not configure an established HPO method (such as BO) with a predefined structure and associated control parameters (e.g., varying the surrogate model of BO). Instead, we introduce a *new* configurable algorithmic framework, which covers many different MF-HPO structures, including well-established principles for multifidelity handling (e.g., SH) as well as new approaches (e.g., equal batch size in all proposals).

In addition to designing well-performing algorithms, it is equally important to facilitate an understanding of the effects of all considered design choices. The field of *sensitivity analysis* (SA) comprises a multitude of methods to assess the importance of input factors on the output of a mathematical model [37]. Functional ANOVA (fANOVA) methods, which decompose the response of a (mathematical) model or function into lower-order components, are a widely studied method in the field of SA, dating back to [38]. This class of methods has also become popular in the field of ML to analyze the importance of hyperparameters [39].

Popular ways of analyzing effects of algorithmic effects in ML and algorithm configuration are *ablation studies* [40]. This involves measuring the performance when removing one or more of algorithmic subcomponents to understand the relative contribution of the ablated components to overall performance. There are different ways of performing an ablation analysis; probably the most common approach is *leave-one-component-out* (LOCO) ablation [41]. In the context of algorithm configuration, Fawcett and Hoos [40] proposed an ablation approach that links a source configuration (e.g., the default) to a target (e.g., the optimized configuration) through an ablation path.

Nevertheless, many existing works that propose or improve HPO or algorithm configuration systems do not analyze the algorithmic choices of an optimized system, and the ones that do perform relatively straightforward analyses. For example, Minton [21] compared the designs and their approach finds automatically to the designs expert humans generated. López-Ibáñez and Stützle [42] performed ANOVA and non-parametric Friedman tests to investigate in detail the effects that algorithmic choices, found through automatic configuration [26], have on the performance of multiobjective ant colony optimization algorithms. de Nobel et al. [43] presented a modular framework for CMA-ES variants on which they perform optimization; in particular, they investigate how the optimized configuration changes when the search space is enlarged by introducing new components.

III. METHODOLOGY

A. Supervised Machine Learning

Supervised ML typically deals with a dataset (which is, mathematically speaking, a tuple $\mathcal{D} = ((\mathbf{x}^{(i)}, y^{(i)})) \in (\mathcal{X} \times \mathcal{Y})^n$ of n observations, assumed to be drawn i.i.d. from a data-generating distribution $\mathbb{P}_{\mathcal{X}\mathcal{Y}}$. An ML model is a function $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}^g$ that assigns a prediction to a feature vector from \mathcal{X} .¹ \hat{f} is itself constructed by an *inducer* function \mathcal{I} , i.e., the model-fitting algorithm. The inducer $\mathcal{I} : (\mathcal{D}, \lambda) \mapsto \hat{f}$ uses training data \mathcal{D} and a vector of *hyperparameters* $\lambda \in \Lambda$ that govern its behavior. The overall goal of supervised ML is to derive a model \hat{f} from a data set \mathcal{D} so that \hat{f} predicts data sampled from $\mathbb{P}_{\mathcal{X}\mathcal{Y}}$ best. The quality of a prediction is measured as the discrepancy between predictions and ground truth. This is operationalized by the loss function $L : \mathcal{Y} \times \mathbb{R}^g \rightarrow \mathbb{R}_0^+$, which is to be minimized during model fitting. In contrast to the optimization problems that we will define in Sections III-B and III-C, we term this the “first-level” optimization problem.

The expectation of the loss value of predictions made for data samples drawn from $\mathbb{P}_{\mathcal{X}\mathcal{Y}}$ is the *generalization error*

$$\text{GE} := \mathbb{E}_{(\mathbf{x}, y) \sim \mathbb{P}_{\mathcal{X}\mathcal{Y}}} \left[L(y, \hat{f}(\mathbf{x})) \right] \quad (1)$$

which cannot be computed directly if $\mathbb{P}_{\mathcal{X}\mathcal{Y}}$ is not known beyond the available data \mathcal{D} . Therefore, one often uses so-called *resampling* techniques that fit models on N_{iter} subsamples $\mathcal{D}[J_j]$ and evaluate them on complements $\mathcal{D}[-J_j]$ of these

¹where g allows handling of multioutput regression, as well as multiclass classification with g classes by returning decision scores.

subsets to obtain an estimate of the generalization error

$$\widehat{\text{GE}}(\mathcal{I}, \lambda, \mathbf{J}) = \frac{1}{N_{\text{iter}}} \sum_{j=1}^{N_{\text{iter}}} L(y[-J_j], \mathcal{I}(\mathcal{D}[J_j], \lambda)(x[-J_j])). \quad (2)$$

Depending on the resampling method, the inducer \mathcal{I} , and the quantity of data in \mathcal{D} , estimating the generalization error $\widehat{\text{GE}}(\mathcal{I}, \lambda, \mathbf{J})$ can require large amounts of computational resources.

B. Hyperparameter Optimization

The goal of HPO is to identify a hyperparameter configuration that performs well in terms of the estimated generalization error in (2). Often, optimization only concerns a subspace of available hyperparameters because some hyperparameters might be set based on prior knowledge or due to other constraints. One would therefore split up the space of hyperparameters Λ into a subspace of hyperparameters Λ_S over which optimization takes place, and the remaining hyperparameters $\Lambda_C = \Lambda/\Lambda_S$ for which values λ_C are given exogenously. We define the HPO problem as

$$\lambda_S^* \in \underset{\lambda_S \in \Lambda_S}{\text{argmin}} c(\lambda_S) = \underset{\lambda_S \in \Lambda_S}{\text{argmin}} \widehat{\text{GE}}(\mathcal{I}, (\lambda_S, \lambda_C), \mathbf{J}). \quad (3)$$

Here, λ_S^* denotes a theoretical optimum, and $c(\lambda_S)$ is a shorthand for the estimated generalization error in (2). We refer to Problem 3 as the “second-level” optimization problem.

Hyperparameters can be either continuous, discrete, or categorical, and search spaces are often a mix of the different types. The search space may be hierarchical, i.e., some subordinate hyperparameters can only be set in a meaningful way if another parent hyperparameter takes a certain value. In particular, many AutoML frameworks perform optimization over a hierarchical hyperparameter space that represents the components of a complex ML pipeline [1].

Many HPO algorithms can be characterized by how they handle two different tradeoffs: 1) the exploration versus exploitation tradeoff refers to how much budget an optimizer spends on either trying to directly exploit the currently available knowledge base by evaluating very close to the currently best candidates (e.g., local search) or whether it explores the search space to gather new knowledge (e.g., RS) and 2) the inference versus search tradeoff refers to how much time and overhead is spent to induce a model from the currently available archive data in order to exploit past evaluations as much as possible. Other relevant aspects that HPO algorithms differ in are: *Parallelizability*, i.e., how many configurations a tuner can (reasonably) propose at the same time; *global versus local* behavior of the optimizer, i.e., if updates are always quite close to already evaluated configurations; *noise handling*, i.e., if the optimizer takes into account that the estimated generalization error is noisy; *search space complexity*, i.e., if and how hierarchical search spaces can be handled; *multifidelity*, i.e., if the optimizer uses cheaper evaluations to infer performance on the full data.

Multifidelity methods make use of the fact that the resampling procedure in (2) can be modified in multiple ways to make evaluation cheaper: one can 1) reduce the training sizes

$|J_j|$ via subsampling, as model evaluation complexity is often at least linear in training set size or 2) change some components in λ in a way that makes model fits cheaper. Examples of 2) are reducing the overall number of training cycles performed by a neural network fitting process or reducing the number of base learner fits in a bagging or boosting method. These modifications can both increase the variance of $\widehat{\text{GE}}$ and introduce an (often pessimistic) bias, as models trained on smaller datasets or with values of λ that make fitting cheaper often have worse generalization errors.

We introduce a *fidelity* parameter $r \in (0, 1]$ that influences the resource requirements of the evaluation of $\widehat{\text{GE}}$ and define

$$c(\lambda_S; r) := \widehat{\text{GE}}(\mathcal{I}, (\lambda_S, \lambda_C(r)), \mathbf{J}(r)). \quad (4)$$

With this definition, we make the choice that r should influence the evaluation cost of $\widehat{\text{GE}}$ only by modifying the resampling, $\mathbf{J}(r)$ or by modifying a hyperparameter $\lambda_C(r)$. Typically, r only affects one of these aspects at a time, and if it affects λ_C , it only affects a single hyperparameter dimension.

Note that we normally assume that a higher fidelity r returns a better model in terms of the estimate of the generalization error, and the best estimate is returned for $r = 1$. Therefore, r enters the expression in a way where it can influence performance but is not searched over. We define $c(\lambda_S) := c(\lambda_S; 1)$ as in [44], and the optimization problem remains as in (3).

This assumption may be violated in some scenarios, and model performance could worsen for a higher value of r (e.g., a neural network, which may overfit on a small dataset if trained for too many epochs). In this case, we define the optimization problem as $(\lambda_S^*, r^*) \in \underset{\lambda_S \in \Lambda_S, r \in (0, 1]}{\text{argmin}} c(\lambda_S; r)$.

The resource requirements of evaluating $c(\lambda; r)$ can have a complicated relationship with λ and r ; in practice, r is chosen in such a way that it has an overwhelming and linear influence on resource demand. The overall cost of optimization up to a given point in the optimization process is therefore assumed to be the cumulative sum of the values of r of all evaluations of $c(\lambda; r)$ up to that point. We can also interpret r as the fraction of the budget of a single full fidelity model evaluation that must be spent for evaluating $c(\lambda; r)$.

Given the definition of the HPO problem, we present an (MF-)HPO algorithm for a single, synchronous worker in its most generic form in Algorithm 1. Until a predetermined budget is exhausted, such an algorithm decides in every iteration 1) which configuration(s) λ_S to evaluate next and 2) which fidelity r to use for evaluation; nonmultifidelity algorithms set this to $r = 1$ as default. The algorithm makes use of an *archive* \mathcal{A} , a database recording previously proposed hyperparameter configurations and, if available, their evaluation results. This database can be shared among multiple worker processes that optimize concurrently.

The optimization process can be accelerated by making efficient use of parallel resources. We distinguish between *synchronous* and *asynchronous* scheduling. The former starts multiple evaluations synchronously at the same time and waits until all of these have finished. To be more precise, a number of $k > 1$ configurations are proposed in line 2 and evaluated

Algorithm 1 Generic HPO Algorithm

```

1: while budget is not exhausted do
2:   Propose  $(\lambda_S^{(i)}, r^{(i)})$ ,  $i = 1, \dots, k$ , based on archive  $\mathcal{A}$ 
3:   Write proposals into a shared archive  $\mathcal{A}$ 
4:   Estimate generalization error(s)  $c(\lambda_S^{(i)}; r^{(i)})$ 
5:   Write results into shared archive  $\mathcal{A}$ 
6: end while
7: Wait for workers to synchronize
8: Return best configuration in archive  $\mathcal{A}$ 

```

in parallel in line 4, all within the inner loop of Algorithm 1. Given K available parallel resources, it should be ensured that the number k of configurations scheduled in parallel is not significantly smaller than K and that the evaluation runtimes amongst these k configurations do not differ significantly in order to avoid unnecessarily idling single parallel resources. In contrast, for asynchronous scheduling, Algorithm 1 is run individually in K separate worker processes. Given a shared archive that is synchronized between the workers, every worker can independently schedule new configurations to evaluate.

C. Algorithm Design and Configuration

Our goal will be to design and configure a new HPO algorithm based on a superset of design choices included in previously published HPO methods. We are interested in finding a configuration (or making design choices) based on a set of training instances that works across a broad set of future problem instances. This problem is called *algorithm configuration* [5], [45]. It is quite similar to HPO; a major difference is that algorithm configuration optimizes the configuration of an arbitrary algorithm over a diverse set of often heterogeneous instances for optimal average performance, while HPO performs a per-instance configuration of an ML inducer for a single data set. We introduce the following notation for consistency with the relevant literature: γ denotes configuration parameters controlling our optimizer A , while λ denotes hyperparameters optimized by our optimizer, controlling our inducer \mathcal{I} . The algorithm configuration problem can be formally stated as follows. Given an algorithm $A : \Omega \times \Gamma \rightarrow \Lambda$ parameterized by $\gamma \in \Gamma$ and a distribution \mathbb{P}_Ω over problem instances Ω together with a cost metric ζ , we must find a parameter setting γ^* that minimizes the expected $\zeta(A)$ over \mathbb{P}_Ω

$$\gamma^* \in \underset{\gamma \in \Gamma}{\operatorname{argmin}} \mathbb{E}_{\omega \sim \mathbb{P}_\Omega} [\zeta(A(\omega, \gamma))]. \quad (5)$$

In our example, Γ corresponds to the space of possible components of our HPO method and Ω corresponds to a class of HPO problems (i.e., ML methods and datasets on which they are evaluated) for which their configuration should be optimal. Based on a training set of representative instances $\{\omega_i\}$ drawn from \mathbb{P}_Ω , a configuration γ^* that minimizes c across these instances should be chosen through optimization. When necessary, we refer to this process as the “third-level” optimization problem to distinguish it from the optimization performed by the HPO algorithm A , i.e., the second-level optimization.

Algorithm 2 SMASHY Algorithm

Configuration Parameters: batch size schedule $\mu(b)$, number of fidelity stages s , survival rate η_{surv} , fidelity rate η_{fid} , SAMPLE method (either SAMPLETOURNAMENT or SAMPLEPROGRESSIVE), *batch_method* (one of equal, SH, or HB), total budget B ; further configuration parameters of SAMPLE: $\mathcal{I}_{\text{surv}}$, $\mathbb{P}_\lambda(A)$, $\rho(t)$, $(N_s^0(t), N_s^1(t))$, n_{tm} .

State Variables: Expended budget fraction $t \leftarrow 0$, bracket counter $b \leftarrow 1$ (remains 1 for *batch_method* \in {equal, SH}), current fidelity $r \leftarrow 1$, batch of proposed configurations $C \leftarrow \emptyset$

```

1: while  $t < 1$  do
2:   if  $r = 1$  then  $\triangleright$  Generate new batch of configurations
3:      $r \leftarrow (\eta_{\text{fid}})^{b-s}$ 
4:      $C \leftarrow \text{SAMPLE}(\mathcal{A}, \mu(b), r; \mathcal{I}_{\text{surv}}, \mathbb{P}_\lambda(A),$ 
5:        $\rho(t), (N_s^0(t), N_s^1(t)), n_{\text{tm}})$ 
6:     if batch_method = HB then
7:        $b \leftarrow (b \bmod s) + 1$ 
8:     end if
9:   else  $\triangleright$  Progress fidelity
10:     $r \leftarrow r \cdot \eta_{\text{fid}}$ 
11:     $C \leftarrow \text{SELECT\_TOP}(C, |C|/\eta_{\text{surv}})$ 
12:    if batch_method = equal then
13:       $\tilde{\mu} \leftarrow \mu(b) - |C|$ 
14:       $C \leftarrow C \cup \text{SAMPLE}(\mathcal{A}, \tilde{\mu}, r; \mathcal{I}_{\text{surv}}, \mathbb{P}_\lambda(A),$ 
15:         $\rho(t), (N_s^0(t), N_s^1(t)), n_{\text{tm}})$ 
16:    end if
17:    end if
18:    Evaluate configuration(s)  $c(\lambda_S; r)$  for all  $\lambda_S \in C$ 
19:    Write results into shared archive  $\mathcal{A}$ 
20:     $t \leftarrow t + r \cdot |C|/B$   $\triangleright$  Update budget spent
21: end while

```

IV. FORMALIZING BROAD CLASS OF MF-HPO ALGORITHMS

We aim to find an HPO algorithm that performs particularly well in the multifidelity setting. To design an algorithm by optimization, we propose a framework and search space of HPO algorithm candidates that cover a large class of possible algorithms and focus on a subclass of algorithms similar to HB because of their favorable properties. This subclass focuses on multifidelity algorithms that use a predefined schedule of geometrically increasing fidelity evaluations containing algorithms like HB [14] and BOHB [16].

The basis of this framework is presented in Algorithm 2, which can be configured by combining algorithmic building blocks in novel ways. The main difference to Algorithm 1 is that the *Propose* part is specified more explicitly. At its core, Algorithm 2 consists of two parts: 1) sampling new configurations at low fidelities (lines 2–7) and 2) increasing the fidelity for existing configurations (lines 8–14). In contrast to Algorithm 1, Algorithm 2 makes use of state variables t , b , and r to account for optimization progress. However, these variables are only shown in Algorithm 2 for clarity and can, in principle, be inferred from the archive \mathcal{A} . As argued in Section III, every single worker instance of Algorithm 1 can, in principle, be scheduled asynchronously, but we do not consider this in this work.

TABLE I

RS, BO, SH, HB, AND BOHB AS INSTANCES OF ALGORITHM 2. η , ρ , AND N_s ARE CONFIGURATION PARAMETERS OF THE RESPECTIVE ALGORITHMS. “—” DENOTES THAT THE VALUE HAS NO INFLUENCE ON THE ALGORITHM IN THIS CONFIGURATION. *: BO AND BOHB USE INDUCERS THAT PRODUCE NONSTANDARD MODEL FUNCTIONS, WHICH DO NOT AIM TO PREDICT THE ACTUAL PERFORMANCE OF CONFIGURATIONS, AND INSTEAD CALCULATE THE VALUE OF AN ACQUISITION FUNCTION SUCH AS EI [4] (FOR BO) OR THE RATIO OF TWO KERNEL DENSITY ESTIMATOR (KDE) MODELS (FOR BOHB). †: IN A SMALL DEPARTURE FROM BOHB, ALGORITHM 2 USES THE KDE ESTIMATE OF GOOD POINTS FOR ALL SAMPLED POINTS, EVEN WHEN RANDOMLY INTERLEAVED. BOHB RANDOMLY INTERLEAVES FROM A UNIFORM DISTRIBUTION

Algorithm	$\mu(b)$	s	η_{surv}	η_{budget}	$\mathcal{I}_{f_{\text{sur}}}$	ρ	N_s	<i>batch_mode</i>	$\mathbb{P}_{\lambda}(\mathcal{A})$
RS	—	1	—	—	—	1	—	—	uniform
BO	1	1	—	—	e.g. GP+EI*	ρ	N_s	—	uniform
SH	μ	$\lfloor -\log_{\eta}(r_{\text{min}}) \rfloor + 1$	η	η	—	1	—	SH	uniform
HB	$\lceil s \cdot \frac{\eta^{s-b}}{s-b+1} \rceil$	$\lfloor -\log_{\eta}(r_{\text{min}}) \rfloor + 1$	η	η	—	1	—	HB	uniform
BOHB	$\lceil s \cdot \frac{\eta^{s-b}}{s-b+1} \rceil$	$\lfloor -\log_{\eta}(r_{\text{min}}) \rfloor + 1$	η	η	TPE*	ρ	N_s	HB	KDE†

In its first iteration, Algorithm 2 uses a SAMPLE-subroutine to initialize the initial batch C of μ solution candidates. The fidelity of the evaluation of the proposed configurations is refined iteratively; when all configurations in the batch have been evaluated with given fidelity r , the top $1/\eta_{\text{surv}}$ fraction of configurations is evaluated with a fidelity that is increased by a factor of η_{fid} . When the fidelity cannot be further increased for a batch because all of its configurations were evaluated at full fidelity $r = 1$, they are set aside, and a new batch of configurations is sampled.

The SAMPLE subroutine creates new configurations to be evaluated, possibly using information from the archive to propose points that are likely to perform well. We allow that any inducer $\mathcal{I}_{f_{\text{sur}}}$ that produces a surrogate model f_{sur} can be used for model-assisted sampling. The subroutine works by at first sampling a number of points from a given generating distribution $\mathbb{P}_{\lambda}(\mathcal{A})$. The performance of these points is then predicted using the surrogate model, and points with unfavorable predictions are discarded in a process we refer to as *filtering*. This process is repeated until the requested number μ of nondiscarded points is obtained. N_s and ρ have the same function as in [16] (see Section IV-A5), with the filter factor N_s controlling the number of sampled points needed for each of the μ points returned, and ρ controlling the fraction of points that are not filtered. Thus, the configuration space of sampling methods also includes purely random sampling, as in HB, by setting $\rho = 1$. The influence of the surrogate model on sampled candidates is larger when 1) the number of sampled configurations N_s is large or 2) the fraction ρ of candidates sampled at random is small. We present two slightly different SAMPLE algorithms: SAMPLETOURNAMENT (Algorithm 3) and SAMPLEPROGRESSIVE (Algorithm 4) based on this principle (see Appendix A in the supplementary material). Both allow to use different N_s values for different points they sample, parameterized by N_s^0 and N_s^1 .

While hyperparameters λ_S are proposed by one of the two SAMPLE methods, the fidelity hyperparameter r follows a fixed schedule similar to SH [19] and HB [14], with a few extensions. For one, the survivor factor η_{surv} can be a different value from the fidelity scaling factor η_{fid} . Furthermore, the algorithm allows three scheduling modes, controlled by *batch_method*: SH does SH. The HB mode evaluates brackets, as performed by HB. While $\mu(b)$ is, in principle, a free configuration parameter

for every value of b , we choose to set $\mu(b)$ so that total budget expenditure is approximately equal between all brackets. This follows the principle used in HB, but the dependency on η_{surv} and η_{fid} is more complex and determined dynamically. Finally, `equal_batch_method` uses equal batch sizes for every evaluation. Individuals that perform badly at low fidelity are removed, as in SH, but new individuals are sampled to fill up batches to the original size. Because new individuals are added to the batches at all fidelity steps, it is not necessary to use brackets with different initial fidelities, and therefore, only a single repeating bracket $b = 1$ is used. The `equal` method is an original contribution of this work and was designed to be similar to HB while using parallel resources more efficiently; the two batch scheduling methods are illustrated in Fig. 1.

If the exploration–exploitation tradeoff is not balanced properly, the optimization progress can either stagnate or function evaluations are wasted due to too much exploration of uninteresting regions of the search space. However, the relative importance of exploration and exploitation can change throughout the course of optimization, where exploration performed later during the optimization is not as useful as during the beginning. The given configuration space makes it possible to make the exploration–exploitation tradeoff dependent on optimization progress by providing the option to make $\rho(t)$ and $(N_s^0(t), N_s^1(t))$ dependent on the proportion of exhausted total budget at every configuration proposal step. It is likely that large values of $\rho(t)$ /small values of $N_s(t)$ perform better when t is small. Conversely, it is likely that small $\rho(t)$ /large $N_s(t)$ work well for large t .

A. Common MF-HPO Algorithms Covered by Algorithm 2

The following describes a few common HPO algorithms that can be instantiated within this framework; see Table I for specific configuration parameter settings within Algorithm 2 that correspond to these algorithms.

1) *Random Search*: Configurations λ_S are drawn (uniformly) at random, and every configuration is evaluated with full fidelity $r = 1$. Parallelization is straightforward, as configurations are drawn independently.

2) *Bayesian Optimization* [8]: The configuration that maximizes an acquisition function $a(\lambda)$ (e.g., expected improvement, EI [4]) is proposed and evaluated with the full fidelity $r = 1$. $a(\lambda)$ is based on a surrogate model trained on the

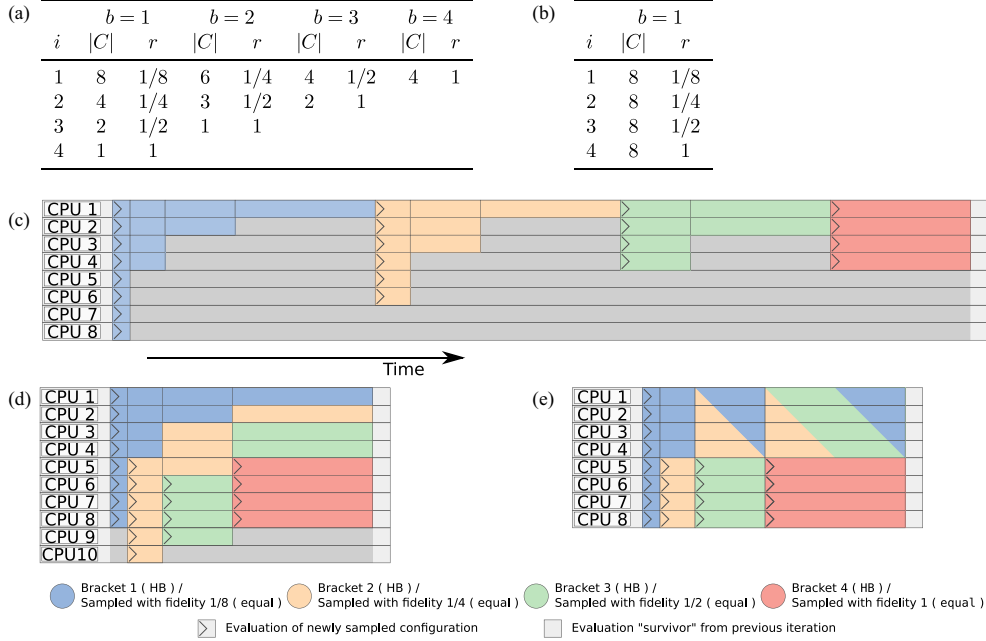


Fig. 1. Illustration of the different *batch_methods* used, corresponding to the values of $\eta_{\text{fid}} = \eta_{\text{surv}} = 2$, $s = 4$, and $\mu = 8$. The tables show the (a) HB method and (b) equal method. Shown are the number $|C|$ and fidelity value r of configurations being evaluated in the iterations i of the various brackets counted by b . Except for i , the variables are the same as in Algorithm 2. Subfigures (c)–(e) illustrate resource utilization by the batch methods, given availability of parallel resources. (c) Naively scheduling the configuration evaluations one batch after another can make use of available parallel resources but leaves many of them idle. (d) Hypothetical way of scheduling configuration evaluations of different brackets at the same time so that all configurations with the same r -value are scheduled together utilizes resources more efficiently, but the number of evaluations in each batch still varies. (e) Simpler equal batch scheduling method always evaluates the same number of configurations within each batch and, therefore, makes optimal use of available parallel resources.

archive \mathcal{A} . BO can be parallelized by either using methods that can propose multiple points at the same time using a single surrogate model or, alternatively, by fitting a surrogate model on the anticipated outcome of configurations that were proposed but not yet evaluated [11]. BO can be represented in Algorithm 2 by using an inducer $\mathcal{I}_{f_{\text{surr}}}$ that produces a function f_{surr} equal to the composition of model prediction and acquisition function. In its basic form, BO is not an MF algorithm and therefore always sets $r = 1$.

3) *Successive Halving* [19]: SH, also called sequential halving [46], is a simple multifidelity optimization algorithm that combines the random sampling of configurations with a fixed schedule for r . At the beginning, a batch of μ configurations is sampled randomly and evaluated with an initial fidelity $r_{\text{min}} < 1$. This is followed by repeated “halving” steps, where the top fraction η^{-1} of configurations is kept and evaluated after r is increased by a factor of η , until the maximum fidelity value is reached. The schedule is chosen to keep the total sum of all evaluated r constant in each batch. Both η_{surv} and η_{fid} in Algorithm 2 correspond to SH’s η -parameter.

4) *Hyperband* [14]: Similar to SH, HB uses a fixed schedule for the fidelity parameter r , but it augments SH by using multiple *brackets* b of SH runs starting at different $r_{\text{min}}(b)$ and with different $\mu(b)$. The number of brackets is

set to

$$s = \lfloor \log_{\eta}(1/r_{\text{min}}) \rfloor + 1 \quad (6)$$

which coincides with the number of fidelity steps that can be performed on a geometric scale on the interval $[r_{\text{min}}, 1]$. In bracket $b \in \{1, 2, \dots, s\}$, a number of $\mu(b)$ samples are initially sampled and evaluated with initial fidelity $r = \eta^{s-b}$. $\mu(b)$ is chosen such that each bracket needs an approximately similar amount of budget: $\mu(b) = \lceil s \cdot (\eta^{s-b}/s - b + 1) \rceil$.

5) *Bayesian Optimization Hyperband* [16]: Model-based methods outperform HB when a relatively large amount of budget is available and many objective function evaluations can be performed. BOHB was created to overcome this drawback. This method iterates through SH brackets like HB, but, instead of sampling new configurations randomly, it uses information from the archive to propose points that are likely to perform well. A total number of N_s configurations are proposed for evaluation; ρ are sampled at random, and the rest are chosen based on a surrogate model induced on the evaluated configurations in \mathcal{A} . The models used by BOHB are a pair of KDEs of the top and bottom configurations in \mathcal{A} , similar to the process in [47]. To implement BOHB in Algorithm 2, one, therefore, needs to use an inducer $\mathcal{I}_{f_{\text{surr}}}$ that

TABLE II
THREE BENCHMARK COLLECTIONS OF YAHPO GYM USED IN OUR BENCHMARK

Scenario	Target Metric	d	Hyperparameter Types				# Instances	# Training Set
			Cont.	Integer	Categ.	Hierarchical		
<i>lcbench</i> : HPO of a neural network	cross entropy loss	7	6	1	0	✗	35	8
<i>rbv2_super</i> : AutoML pipeline configuration	log loss	38	20	11	7	✓	89	30
<i>nb301</i> : Neural architecture search	validation accuracy	34	0	0	34	✓	1	—

produces a function that calculates the ratio of kernel densities, an unusual kind of regression model.

B. Limitations and Further MF-HPO Algorithms

The following lists notable HPO algorithms not currently covered by the optimization space of Algorithm 1. They were excluded because they differ in too substantial ways from the other algorithms considered here.

1) *FABOLAS* [48]: Fabolas is a continuous multifidelity BO method, where the conditional validation error is modeled as a Gaussian process using a complex kernel-capturing covariance with the training set fraction $r \in (0, 1]$ to allow for adaptive evaluation at different resource levels.

2) *Asynchronous Successive Halving* [15] and *Asynchronous Hyperband*: HB, as well as SH, have the drawback that batch sizes decrease throughout the stages of an SH run, preventing efficient utilization of parallel resources. ASHA is an effective method to parallelize SH by an asynchronous parallelization scheme. A shared archive across a number of different workers is maintained. Instead of waiting until all n configurations of a batch have been evaluated for fidelity r , every free worker queries the shared archive \mathcal{A} for “promotable” configurations (i.e., configurations that belong to the fraction of top η^{-1} configurations evaluated with the same fidelity). Asynchronous HB works similarly.

3) *Asynchronous BOHB* [17]: A-BOHB, an asynchronous extension of BOHB where configurations are sampled from a joint Gaussian Process, explicitly capturing correlations across fidelities. In contrast to ASHA and asynchronous versions of BOHB in the original BOHB publication [16], A-BOHB does not perform synchronization after each stage but instead uses a stopping rule [49] to asynchronously determine whether a configuration should continue to run or be terminated.

V. EXPERIMENTAL ANALYSIS

Given the formalization of the framework in Section IV, our goal is to find the best representative (out of this class of algorithms) by solving the third-level optimization problem in (5), and explain the role of specific algorithmic components in a benchmark-driven approach. We aim to answer the following research questions.

- RQ1: How does the optimal configuration of our MF-HPO framework differ between problem scenarios, i.e., do different problem scenarios benefit from different HPO algorithms?
- RQ2: How does our optimized MF-HPO algorithm compare to other established HPO implementations?
- RQ3: Does the successive-halving fidelity schedule have an advantage over the simpler equal-batch-size schedule?

RQ4: What is the effect of using multifidelity methods in general?

RQ5a: Does changing SAMPLE configuration parameters throughout the optimization process offer an advantage?

RQ5b: Does (more complicated) surrogate-assisted sampling in SAMPLE provide an advantage over using simple random sampling with surrogate filtering?

RQ6: What effect do different surrogate models (or using no model at all) have on performance?

RQ7: Does the equal-batch-size schedule give an advantage over established methods when parallel resources are available?

We rely on benchmark scenarios of the YAHPO Gym benchmark suite [50], each of which provides a number of related instances of optimization problems. The benchmark scenarios we have chosen cover three important application areas of AutoML: HPO of a neural network (*lcbench*), AutoML pipeline configuration (*rbv2_super*), and neural architecture search (*nb301*). These classes of problems do not only represent common and relevant tasks for researchers and practitioners in the field; as presented in Table II, they are also quite different with regards to: 1) the dimensionality of the search space; 2) hyperparameter types (categorical, integer, and continuous); and 3) whether there are hierarchical dependencies between hyperparameters. More details on the characteristics of the problem classes are given in Appendix B in the supplementary material. To avoid an optimistic bias in the analysis caused by over-adaptation to the random peculiarities of the particular instances used during configuration, we are using meta-holdout splits on the level of HPO problem instances (see Appendix D in the supplementary material). This means that for analyzing the performance of a configured candidate of Algorithm 2, we are evaluating this candidate by running it on instances that were not seen during configuration. Algorithm 2 is always run with a budget limit corresponding to $30 \cdot d$ full fidelity evaluations (where d is the dimension of the problem instance).

A. Algorithm Design via Configuration

First, we describe the experiments we conducted to configure Algorithm 2 via optimization.

We follow the PBO principle and configure Algorithm 2 by optimizing separately for different HPO scenarios, namely, for *lcbench* and *rbv2_super*, resulting in two optimized configurations $\gamma^{*lcbench}$ and γ^{*rbv2_super} , respectively. The *nb301* scenario is not used for configuration, but exclusively for subsequent analysis.

For the algorithm configuration of our framework (third level), the performance objective $\mathbb{E}_{\omega \sim \mathbb{P}_\Omega}[\zeta(A(\omega, \gamma))]$ for a

TABLE III
SUMMARY OF EXPERIMENT. SHOWN ARE THE VARIOUS OPTIMIZER CONFIGURATIONS γ THAT WERE OBTAINED FROM OPTIMIZATIONS WITH DIFFERENT CONSTRAINTS. “NAME”: THE NAME BY WHICH WE REFER TO THE CONFIGURATION IN THE TEXT. “RQ”: THE RESEARCH QUESTION THAT MAINLY RELATES TO THE CONFIGURATION. “OPTIMIZE”: WHETHER THE GIVEN CONFIGURATION WAS OBTAINED BY CONDUCTING A (POSSIBLY CONSTRAINED) OPTIMIZATION (\checkmark), OR BY SUBSTITUTING VALUES INTO THE GLOBAL OPTIMUM γ^*

Name	RQ	Optimize	Design Modification
γ^*	1, 2, 3	\checkmark	none (global optimization)
γ_1	4	\times	$\eta_{\text{fid}} \rightarrow \infty$
γ_2	5a	\checkmark	$n_{\text{trn}}(0) = n_{\text{trn}}(1)$, $N_s^0(0) = N_s^0(1)$, $N_s^1(0) = N_s^1(1)$, $\rho(0) = \rho(1)$
γ_3	5b	\checkmark	$\text{filter_method} \rightarrow \text{tournament}$, $n_{\text{trn}} \rightarrow 1$, $N_s^0(0) = N_s^0(1) = N_s^1(0) = N_s^1(1)$, $\rho(0) = \rho(1)$
γ_4	6	\times	$\text{batch_method} \rightarrow \text{equal}$, $\mathcal{I}_{\text{sur}} \rightarrow *$
γ_5	6	\times	$\text{batch_method} \rightarrow \text{equal}$, $\rho \rightarrow 0$
γ_6	6	\times	$\text{batch_method} \rightarrow \text{equal}$, $\rho \rightarrow 0$, $\mathbb{P}_\lambda(\mathcal{A}) \rightarrow \text{uniform}$
γ_7	7	\times	$\text{batch_method} \rightarrow \text{equal}$, $\mu \rightarrow 32$, quadruple budget

configuration γ in (5) is estimated by running Algorithm 2 (i.e., second-level optimization) configured by γ on a set of problem instances and taking the average of observed performances. For this, all problem instances included in the respective benchmark scenario that has not been held out for subsequent analysis are used. As a configuration for our framework, we use BO with the lower confidence bound acquisition function [51] with interleaved random configurations every three evaluations.² Configuration is repeated three times for each scenario, each running for 60 h, with different random seeds. To get the overall best configuration, the set of all evaluated configurations γ (i.e., the third-level optimization archive) is combined into a single data set for each scenario. To estimate the actual best configuration, a common *identification criterion* [52] is used: a surrogate model is fitted on the combined datasets and the optimum among the in-sample predictions of this model is used ($\gamma^{*lcbench}$ and γ^{*rbv2_super} , respectively). We also store the (surrogate-smoothed) optima of all three individual optimization runs and record the range of configuration parameter values to obtain an estimate of the uncertainty of the overall optimal configurations.

The search space used for the optimization of Algorithm 2 is shown in Table V in Appendix C in the supplementary material. While the batch size μ is constant in the equal *batch_method*, it changes for every bracket when *batch_method* is HB. The batch sizes $\mu(2)$, $\mu(3)$, ... are constructed from $\mu(1)$ dynamically as described in Section IV. The search space contains several surrogate learners: Random forests [53] (RF), K -nearest-neighbors with k set to 1 (KNN1), kernelized K -nearest-neighbors with “optimal” weighting [54] (KKN1), and the ratio of density predictions of good and bad points, similar to tree parzen estimators [47] without a hierarchical structure as in BOHB [16] (TPE). For the prefiltering sample distribution $\mathbb{P}_\lambda(\mathcal{A})$, we evaluate both uniform sampling (uniform), and sampling from the estimated density of good points as done in BOHB [16] (KDE). *filter_mb* determines whether the surrogate model makes predictions assuming the highest fidelity value r observed (TRUE), as opposed to assuming the fidelity of the points being sampled; in the framework of the SAMPLE Algorithms 3 and 4 in Appendix A in the supplementary material, this influences the behavior of \mathcal{I}_{sur} .

²Note that this optimizer used for third-level optimization is not an instance of Algorithm 2.

Note that the maximum number of fidelity steps per batch s is not part of the search space and instead inferred automatically from η_{fid} and the lower bound for r that is given as part of the optimization problem instance. As in HB, it is set to the largest number of stages that is possible given η_{fid} and the lower bound on r according to (6).

B. Algorithm Analysis

Our goal in this work is not only to determine configurations of Algorithm 2 that perform well on the respective benchmarking scenarios but also to determine what effect individual components have on performance. However, performing a complete SA would be prohibitively computationally expensive, as it would require evaluation of the objective (i.e., running Algorithm 2) in an experimental design of different configurations. Instead, we evaluate the performance of the candidate configurations found in Section V-A and alternative configurations—which are chosen in a way to allow for answering our research questions—on the benchmark test instances which were held out during configuration. A simple method to answer many of these questions is to take the optimized configuration of Algorithm 2 and swap components of it for simpler components (or removing them completely), thereby performing a one-factor-at-a-time analysis or an ablation study. However, the optimal values of some components may interact strongly with other components. We, therefore, auto-configure the framework several times under certain constraints dictated by our particular research question at hand. For example, to investigate the effect of varying n_{trn} and N_s over t , we run the optimization of Algorithm 2 with the constraint $n(0)$ to be equal to $n(1)$ and compare the resulting configuration to the overall optimum γ . Table III lists the different values of γ we generate under different constraints. For each value of γ , we run the, respectively, configured HPO algorithm on both the *lcbench* and the *rbv2_super* scenario, and (unless stated otherwise) once each for *batch_method* set to equal and HB. We refer to an optimized configuration that was obtained on the *lcbench* scenario with *batch_method* set to equal as $\gamma^{*lcbench}[\text{equal}]$, and to the overall optimum (i.e., the better of $\gamma^{*lcbench}[\text{equal}]$ and $\gamma^{*lcbench}[\text{HB}]$) as $\gamma^{*lcbench}$; similar for *rbv2_super*.

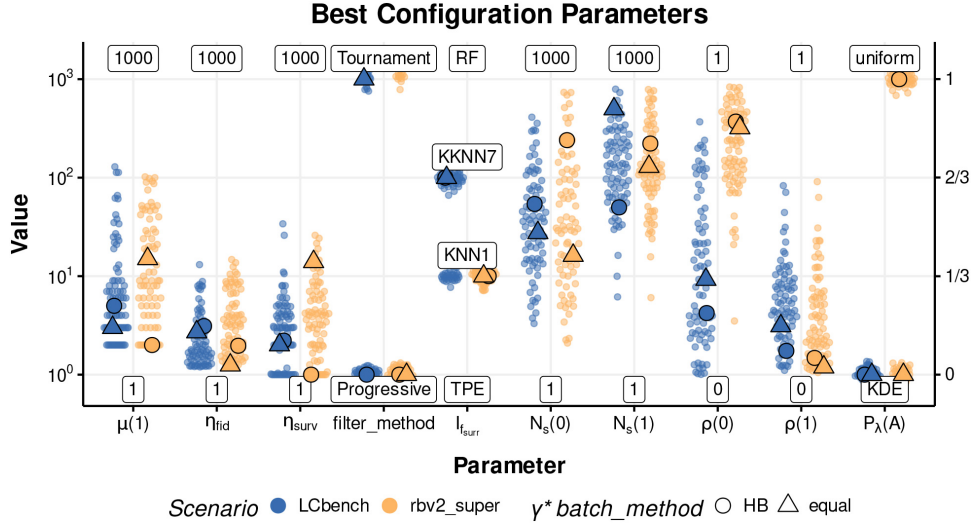


Fig. 2. Beeswarm plot of the best configurations according to the surrogate model over the meta-optimization archive of γ^* . Shown are the top 80 configuration points (according to the surrogate-model-predicted performance) that were evaluated during optimization. Levels of discrete parameters are shown. Most numeric parameters are on a log-scale (left axis), except for $\rho(0)$ and $\rho(1)$, which are on a linear scale (right axis). Instead of showing both $N_s^0(t)$ and $N_s^1(t)$, their geometric mean $N_s(t)$ is shown. The highlighted large points are γ^*_{HB} and γ^*_{EQUAL} , which were found on both benchmark scenarios.

Every evaluation of a framework configuration, i.e., a complete HPO run on a problem instance, is repeated 30 times (with different random seeds) to allow for statistical analysis.

The analysis of our research questions is based on the following tables and visualizations. Table VI in Appendix D in the supplementary material shows the configuration parameters that were selected for each benchmark scenario with various search space restrictions. We perform all optimization runs constrained to the fidelity scheduling `equal` and `HB`, respectively, and denote the resulting optimal configurations γ^*_{equal} and γ^*_{HB} . Fig. 2 shows the configuration values of the top 80 evaluated points according to their surrogate-predicted performance. The ranges covered by the bee swarms are again an indicator of approximate ranges of configuration values that can be expected to work well. Fig. 4 shows the final performance at 30- d full-budget evaluations for all optimization runs that were performed. The standard error shown is the estimated standard deviation of the mean of benchmark-instance-wise performance, representing uncertainty about the “true” performance mean if an infinite number of benchmark instances of the given class of problems were available.

We now describe in more detail how we operationalize each of the research question RQ1–RQ7 and report results.

RQ1: How does the optimal configuration differ between problem scenarios, i.e., do different problem scenarios benefit from different HPO algorithms?

Setup: We investigate the difference in the values that $\gamma^*_{\text{lcbench}}$ and $\gamma^*_{\text{rbv2_super}}$ take, and put this difference in perspective by comparing it to the uncertainty of these values.

To evaluate how well $\gamma^*_{\text{lcbench}}$ and $\gamma^*_{\text{rbv2_super}}$ generalize to other problem scenarios, we evaluate them on the respective instances of scenarios that they were not configured on.

Results: As can be seen in Table VI in the supplementary material and in Fig. 2, many of the selected components of the γ^* are relatively close to each other across the two scenarios on which they were optimized, relative to their uncertainty ranges. $\mathcal{I}_{f_{\text{sur}}}$ is chosen as `KNN1` on `rbv2_super`, but can also use `KKNN7` on `lcbench`, which in fact seems to be slightly preferred. This is interesting as KNN-based models are rarely considered in surrogate-based HPO; the typically preferred random forest model was not selected. $\mathbb{P}_\lambda(\mathcal{A})$ takes any of the two values for `rbv2_super`, but is chosen to be `KDE` in `lcbench`. Finally, $\rho(0)$ is close to 1 in the beginning on `rbv2_super`, and closer to 0 (although still greater than $\rho(1)$) for `lcbench`.

The degree to which the differences in γ^* influence the outcome can be observed in Fig. 4. The optimized results generalize well to test instances from the same scenario as they were configured on. Fig. 3 shows the optimization progress (on unseen test instances) of configurations if configured on the same scenario versus configurations that were configured on a different scenario. We see, for example, a clear advantage of the configurations that we obtained by optimizing directly on `lcbench` when we evaluate them on their respective held out test instances. We suspect that this difference in performance is mainly due to the different choices of surrogate model classes $\mathcal{I}_{f_{\text{sur}}}$ as well as the random interleave fraction ρ (cf. Fig. 2), and that specific settings for these two algorithmic components are needed for `lcbench` to reach optimal performance.

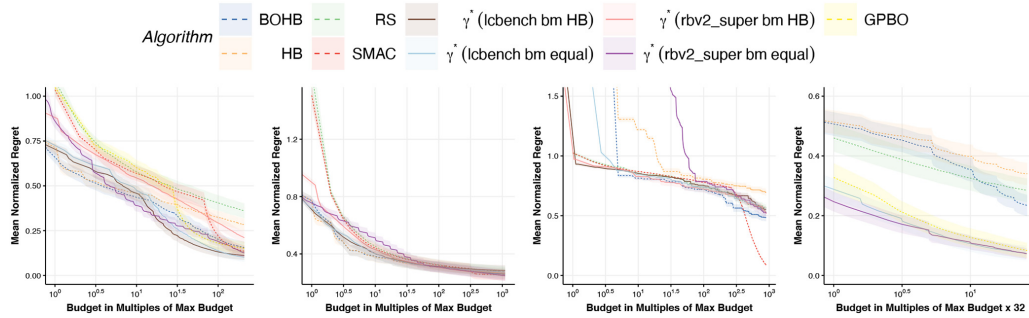


Fig. 3. Optimization progress (mean normalized regret) of serial evaluation on each benchmark scenario as well as $32\times$ parallel evaluation on *lcbench*. Different configurations of Algorithm 2 are executed on benchmark functions that have not been used for the meta-optimization itself, and the progress of these algorithm runs is shown. “ γ^* (*lcbench* bm equal)” is the configuration obtained from optimizing on *lcbench* with *batch_method*=equal, other labels are constructed similarly. Shown is the mean over 30 evaluations, averaged over all available test benchmark instances for each of the three scenarios. The uncertainty bands show the standard error over the test instances. Note the log-scale on the x-axis. Regret is calculated as the difference between the best evaluation performance so far and the overall best value found on each benchmark instance over all experiments; normalized such that 1 corresponds to the median of the performance of all randomly sampled full-fidelity evaluations. We plot performance values observed by the HPO algorithm which depend on evaluation fidelity. This is the reason for the initially “slow” convergence of algorithms that makes their first full-fidelity evaluation late. Note that μ of γ^* [equal] was set to 32 for the parallel evaluations, and HB and BOHB were only naïvely parallelized to simulate a synchronous “single optimizer, multiple workers” environment. See Fig. 6 in Appendix E in the supplementary material for a larger version.

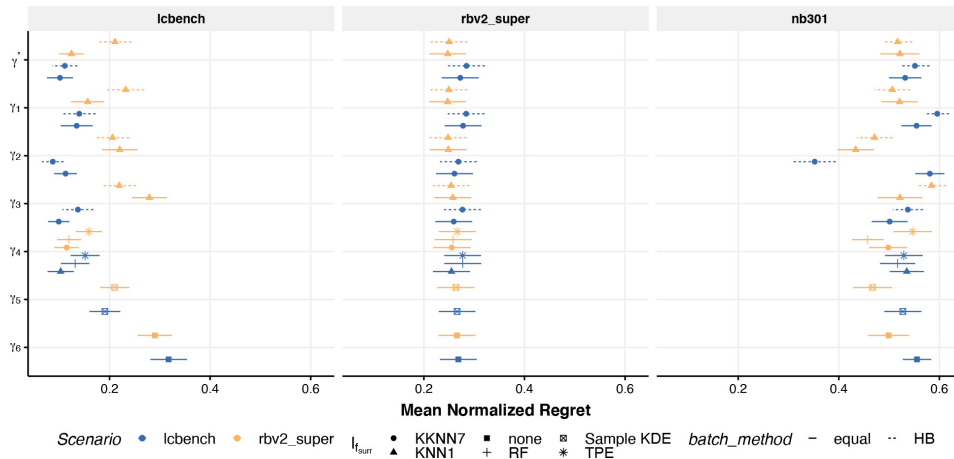


Fig. 4. Mean normalized regret of final performance on “test” benchmark instances for the configuration, shown in Table III. Shown is the mean over 30 evaluations, averaged over all available test benchmark instances for each of the three scenarios. The uncertainty bands show the standard error over instance means. Regret is calculated as the difference between the best evaluation performance so far and the overall best value on each benchmark instance over all experiments; normalized such that 1 corresponds to the median of the performance of all randomly sampled full-fidelity evaluations.

This is not the case for the *rbv2_super* scenario, where none of the different algorithms seem to clearly exploit the problem structure of *rbv2_super* better than others.

RQ2: How does the optimized algorithm compare to other established HPO implementations?

Setup: We evaluate several well-known HPO algorithms in their default configuration on the same benchmark instances: for BOHB [16], we use the implementation found in *HpBandSter*³ (version 0.7.4); for HB [14], we use

*mlr3hyperband*⁴ (version 0.1.2); and for SMAC [5], we use the SMACv3 package⁵ (version 1.0.1). We also construct a traditional Gaussian process-based BO (GPBO) [4] with *mlrMBO*⁶ (version 1.1.5). As GPBO works best with numerical search spaces, we only evaluate it on *lcbench*. Note that GPBO, SMAC, and RS are not multifidelity algorithms and therefore always evaluate points with maximum fidelity 1.

⁴<https://cran.r-project.org/package=mlr3hyperband>

⁵<https://github.com/automl/SMAC3>

⁶<https://cran.r-project.org/package=mlrMBO>

³<https://github.com/automl/HpBandSter>

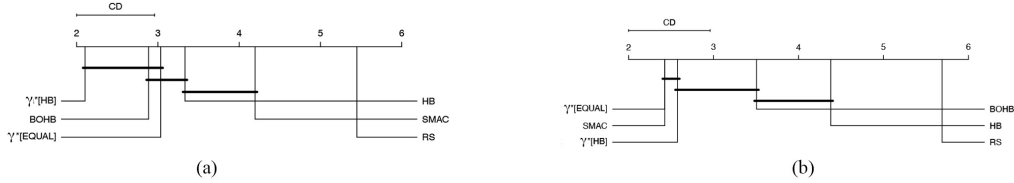


Fig. 5. Critical difference plot [55] comparing the performance of different algorithms across all instances and scenarios. For each of the three scenarios, the mean performance (across replications) for each of the six algorithms is computed ($\gamma^*[\text{HB}]$ is equal to $\gamma^{*lcbench}[\text{HB}]$ for instances of the *lcbench* scenario, and to $\gamma^{*rbv}[\text{HB}]$ for the *rbv2_super* scenario; same for $\gamma^*[\text{EQUAL}]$). The critical difference test is based on the ranks of the algorithms computed per scenario and instance. Lower ranks are better. Horizontal bold bars indicate that there is no significant difference between algorithms ($\alpha = 1\%$). GPBO, which was not evaluated on all scenarios, is not included. (a) Intermediate optimization budget of 100 full evaluations. (b) Full evaluation budget (final performance).

Results: The performance curves for the mean normalized regret are shown in Fig. 3, and the final performance values at $30 \cdot d$ full-fidelity evaluations are shown in Fig. 4. A critical difference plot and test can be seen in Fig. 5(b). The behavior of RS, HB, BOHB, and SMAC is not surprising; initially, RS and SMAC perform the same, as SMAC evaluates an initial random design. After this, the performance of SMAC improves quickly. HB and BOHB initially both perform better than RS or SMAC because of their multifidelity evaluations, but there is little difference between them. After a while, BOHB starts to outperform HB because of its surrogate-based sampling, which aligns with the observations in [16]. Therefore, BOHB performs well for most budgets, often being the best optimizer for a budget of one as well as for 100 full-fidelity evaluations. Given its multifidelity characteristics, HB is a good choice for low budgets, while SMAC is well suited for larger optimization budgets. Our framework is very competitive on both *lcbench* and *rbv2_super*, but is outperformed by SMAC on *nb301*. We assume that this is because Algorithm 2 was not explicitly optimized for the *nb301* scenario.

Although our framework was only optimized for performance at $30 \cdot d$ evaluations, it is also competitive with BOHB after fewer evaluations, as seen in Fig. 5(b).

RQ3: Does the successive-halving fidelity schedule have an advantage over the (simpler) equal-batch-size schedule?

Setup: It is likely that the type of fidelity scheduling used interacts with other configuration parameters. Therefore, we investigate the difference of resulting optimal configurations $\gamma^*[\text{equal}]$ and $\gamma^*[\text{HB}]$.

Results: In both scenarios, the batch method HB is ultimately selected for the optimum γ^* , although Fig. 5(a) and (b) shows that the difference to batch size `equal` is not statistically significant at $\alpha = 1\%$. We observe that the `equal` fidelity scheduling mode has several advantages: it is much simpler than HB as it does not need to keep track of SH brackets and does not need to adapt $\mu(b)$ to make the expended budget at each bracket approximately equal. As another benefit, it allows for easy parallel scheduling of evaluations (see also Fig. 1). This is because it always schedules the same number of function evaluations at a time, which can therefore be run synchronously.

RQ4: What is the effect of using multifidelity methods in general?

Setup: We evaluate the performance of a modified γ^* where the number of fidelity stages s is set to 1, thus ensuring that configurations are only evaluated with maximum fidelity 1.⁷

Results: Our results show the superiority of MF-HPO methods compared to HPO methods that do not make use of lower-fidelity approximations. Fig. 5(a) suggests that multifidelity methods are significantly better than their nonmultifidelity counterparts if optimization is stopped at an intermediate overall budget corresponding to 100 full-fidelity evaluations. To be more precise, we see that BOHB as well as both optimized variants $\gamma^*[\text{equal}]$ and $\gamma^*[\text{HB}]$ (optimized for the respective scenario, respectively) significantly outperform SMAC under this strict budget constraint. In line with [14], HB significantly outperforms RS for this budget. On the other hand, Fig. 5(b) provides evidence that multifidelity methods can achieve performance on the same level as state-of-the-art methods that do not make use of low-fidelity approximations (e.g., SMAC) for larger budgets. We conclude that a properly designed multifidelity mechanism provides substantial improvements of anytime performance without affecting performance for larger budgets negatively. In our opinion, the gain in anytime performance justifies the additional algorithmic complexity that is introduced by multifidelity methods.

RQ5a and RQ5b: Does changing SAMPLE configuration parameters throughout the optimization process offer an advantage? Does (more complicated) surrogate-assisted sampling in SAMPLE provide an advantage over using simple random sampling with surrogate filtering?

Setup: To investigate RQ5a (i.e., the effect of the dependence of ρ , n_{tm} and the N_s configuration parameters on t), we performed an optimization where this t -dependence was removed. As these parameters are interpolated between the values at $t = 0$ and $t = 1$, this corresponds to restricting the search space to where these values are equal, as shown for γ_2 in Table III. In addition to this, we ran another optimization where we further restricted N_s^0 and N_s^1 to be equal, n_{tm} to be 1, and only the `tournament filter_method` be used for RQ5b. The performance of the resulting configurations gives an indication of the performance that is lost for the gain in simplicity.

⁷Because s is not part of the search space Γ and is instead given by 6, this is achieved by setting η_{fid} to ∞ .

Results: The observations made for γ_2 (forbidding change over time) and γ_3 (forbidding change over time and within each batch) are slightly contradictory. In particular, the *nb301* performance of $\gamma_2^{\text{cbench}}_{\text{[HB]}}$ is a visible outlier with regards to optimization performance. There is no obvious explanation from inspecting the configuration parameters of $\gamma_2^{\text{cbench}}_{\text{[HB]}}$, but it is possible that it is an accidental “good fit” of configuration parameters to the specific landscape of *nb301*.

On *lcbench* and *rbv2_super*, the impact of restricting the search space is smaller and within the uncertainty of the performance of a single configuration. However, we note that both changing configuration parameters over time and within each batch sample introduce significant complexity to the algorithm; thus we prefer the restricted optimization results over γ^* .

RQ6: What effect do different surrogate models (or using no model at all) have on performance?

Setup: We evaluate the overall result $\gamma^*_{\text{[equal]}}$ with \mathcal{I}_{sur} set to each of the inducers in the original search space (see Table V in the supplementary material). Furthermore, $\gamma^*_{\text{[equal]}}$ is evaluated with ρ set to 1 (i.e., all points are sampled randomly from a distribution that may be nonuniform), and finally, with $\rho = 1$ and $\mathbb{P}_\lambda(\mathcal{A}) = \text{uniform}$ (i.e., all points are sampled completely uniformly at random).

Results: Surprisingly, the simple k -nearest-neighbors algorithm seems to be chosen consistently by the algorithm configuration for both *lcbench* and *rbv2_super* (see Fig. 2), either with a value of $k = 1$ or $k = 7$. This result is in line with what we already speculated for RQ1. Our ablation experiments suggest that the performance of the optimizer is on average best when using this surrogate learner, even though the differences do not seem to be significant. KNN1 is therefore a reasonable, and simpler, alternative to more complex surrogate learners like the TPE-based method proposed for the original BOHB algorithm.

RQ7: Does the equal-batch-size schedule give an advantage over established methods when parallel resources are available?

Setup: The optimization of ML methods that are expensive to evaluate is often done in parallel; we evaluate the performance of our method and other methods in a (simulated) parallel setting. We evaluate $\gamma^*_{\text{[equal]}}$ with μ set to 32 and with an optimization budget of $30 \cdot 4 \cdot d$, where d is the dimensionality of the optimization problem. We compare it to GPBO with qLCB [10] for 32 parallel evaluations and simulate parallel execution of RS by running $30 \cdot 4 \cdot d$ random evaluations. Both BOHB and SMAC offer parallelized versions, but the YAHPO Gym benchmark package does not yet provide support for asynchronous parallel evaluations [50]. However, since HB and BOHB propose evaluations in batches, we compared HB and BOHB by accounting for submitted batches in increments of 32, essentially simulating a single HB/BOHB optimizer sending evaluations to 32 parallel workers and waiting for their completion synchronously.

Results: Fig. 3 shows that our algorithm is competitive with GPBO—a state-of-the-art synchronously parallel optimization algorithm—when evaluated with 32 parallel resources. This result also shows the main advantage that the `equal` fidelity

schedule has over scheduling like HB, as synchronously parallelizing HB or BOHB puts them at a great disadvantage over even RS. For HB and BOHB, it is necessary to use asynchronously parallelized methods [15], [17] or use an archive shared between multiple workers [16] to obtain competitive results. However, synchronous objective evaluations are much easier to implement in many environments than asynchronous communication between workers, making the advantage of the simplicity of the `equal` schedule even more pronounced.

C. Reproducibility and Open Science

The implementation of the framework in Algorithm 2 and reproducible scripts for the algorithm configuration and analysis are available in public repositories.⁸ All data that were generated by our analyses are available as well.

VI. CONCLUSION

We presented a principled approach and framework to benchmark-driven algorithm design and applied it to generic MF-HPO. We formalized the search space of multifidelity hyperparameter optimizers and created a rich and configurable optimization framework. Given the search space, we used BO for meta-optimization of our framework on two different problem scenarios within the field of AutoML and evaluated the result on held out test problems and an entirely held out test scenario. We evaluated the configured optimizers and compared to BOHB, HB, SMAC, and a simple RS as reference. We performed an extensive analysis of the effect of different algorithmic components on performance, while also considering the additional algorithmic complexity they introduce. Our configured framework showed equal and in some cases superior performance to widely used HPO algorithms.

The additional algorithmic complexity introduced by multifidelity evaluations provides substantial benefits. However, based on our experiments, we argue that design choices made by established multifidelity optimizers like BOHB can be replaced by simpler choices: For example, the (more complex) SH schedule is not significantly better than a schedule using equal batch sizes, which allows for more efficient parallelization.

A KDE-based sampling of points to propose, whether filtered by a surrogate model or not, was consistently chosen by our framework. This detail, which is not usually presented as the main feature of BOHB, seems to have an unexpectedly large impact. On the other hand, our optimization results suggest that a surprisingly simple surrogate learner (knn, $k = 1$) can perform even better.

Some components of our search space with large algorithmic complexity have not shown much benefits. Optimization on *rbv2_super* did choose time-varying random interleaving, and overall, more aggressive filtering late during an optimization run ($N_s(1) > N_s(0)$) was slightly favored, but the results did not consistently outperform a configuration obtained from a restricted optimization that excluded time-varying configuration parameters.

⁸<https://github.com/mlr-org/smashy>,

https://github.com/compstat-lmu/paper_2021_benchmarking_special_issue

Our analysis of the set of best observed performances during optimization indicates that there is a large agreement between benchmark scenarios about what the optimal γ^* configuration should be, with parameters that control (model-based) sampling and the surrogate model being the notable exception. This suggests that there may be a set of configuration parameters that are either generally good for many ML problems, or have little impact on performance and can therefore be set to the simplest value. However, some configuration parameters should be adapted to the properties of the particular given optimization problem. The meta-optimization framework presented in this work can be used in future work to investigate the relationship between features of optimization problems and related optimal configurations.

Other fruitful directions for future work include the more in-depth evaluation of asynchronous evaluations; asynchronous methods are important nowadays where parallel resources are plentiful, but current widely used surrogate-based benchmarks do not allow for easy asynchronous evaluations. Suggested methods, such as waiting with a sleep-timer for an appropriate amount [16], are impractical for meta-optimization.

REFERENCES

- [1] B. Bischl et al., "Hyperparameter optimization: Foundations, algorithms, best practices and open challenges," 2021, *arXiv:2107.05847*.
- [2] L. Kothhoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kothhoff, and J. Vanschoren, Eds. Cham, Switzerland: Springer Int., 2019, pp. 81–95.
- [3] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
- [4] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Global Optim.*, vol. 13, no. 4, pp. 455–492, 1998.
- [5] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed. Berlin, Germany: Springer, 2011, pp. 507–523.
- [6] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw Bayesian optimization," 2014, *arXiv:1406.3896*.
- [7] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.
- [8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst. Vol. 2*, 2012, pp. 2951–2959.
- [9] R. Turner et al., "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," in *Proc. NeurIPS Competition Demonstration Track*, vol. 133. Vancouver, BC, Canada, Dec. 2020, pp. 3–26.
- [10] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Parallel algorithm configuration," in *Learning and Intelligent Optimization*, Y. Hamadi and M. Schoenauer, Eds. Berlin, Germany: Springer, 2012, pp. 55–70.
- [11] B. Bischl, S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs, "MO-MBO: Multiobjective infill for parallel model-based optimization," in *Proc. 8th Int. Conf. Learn. Intell. Optim.*, Gainesville, FL, USA, Feb. 2014, pp. 173–186.
- [12] J. González, Z. Dai, P. Hennig, and N. D. Lawrence, "Batch Bayesian optimization via local penalization," in *Proc. 19th Int. Conf. Artif. Intell. Stat. (AISTATS)*, vol. 51. Cadiz, Spain, May 2016, pp. 648–657.
- [13] C. Chevalier and D. Ginsbourger, "Fast computation of the multi-points expected improvement with applications in batch selection," in *Learning and Intelligent Optimization*. Berlin, Germany: Springer, 2013, pp. 59–69.
- [14] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, pp. 1–52, Jan. 2017.
- [15] L. Li et al., "A system for massively parallel hyperparameter tuning," in *Proc. Int. Conf. Mach. Learn. Syst. (MLSys)*, Austin, TX, USA, Mar. 2020, pp. 230–246.
- [16] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 80. Stockholm, Sweden, Jul. 2018, pp. 1436–1445.
- [17] L. C. Tiao, A. Klein, C. Archambeau, and M. W. Seeger, "Model-based asynchronous hyperparameter optimization," 2020, *arXiv:2003.10865*.
- [18] D. Sculley et al., "Hidden technical debt in machine learning systems," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Montreal, QC, Canada, Dec. 2015, pp. 2503–2511.
- [19] K. G. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proc. 19th Int. Conf. Artif. Intell. Stat. (AISTATS)*, vol. 51. Cadiz, Spain, May 2016, pp. 240–248.
- [20] H. H. Hoos, "Programming by optimization," *Commun. Assoc. Comput. Mach.*, vol. 55, no. 2, pp. 70–80, Feb. 2012.
- [21] S. Minton, "Automatically configuring constraint satisfaction programs: A case study," *Constraints*, vol. 1, pp. 7–43, Sep. 1996.
- [22] S. J. Westfold and D. R. Smith, "Synthesis of efficient constraint satisfaction programs," *Knowl. Eng. Rev.*, vol. 16, no. 1, pp. 69–84, 2001.
- [23] D. Balasubramaniam, L. de Silva, C. A. Jefferson, L. Kothhoff, I. Miguel, and P. Nightingale, "Dominion: An architecture-driven approach to generating efficient constraint solvers," in *Proc. 9th Workshop IEEE/IFIP Conf. Softw. Archit.*, Jun. 2011, pp. 228–231.
- [24] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown, "SATenstein: Automatically building local search SAT solvers from components," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, San Francisco, CA, USA, 2009, pp. 517–524.
- [25] J.-N. Monette, Y. Deville, and P. van Hentenryck, "Aeon: Synthesizing scheduling algorithms from high-level models," in *Aeons: Automating Cyber-Infrastructure*. Boston, MA, USA: Springer, 2009, pp. 43–59.
- [26] M. López-Ibáñez and T. Stützle, "The automatic design of multiobjective ant colony optimization algorithms," *IEEE Trans. Evol. Comput.*, vol. 16, no. 6, pp. 861–875, Dec. 2012.
- [27] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle, "Automatic component-wise design of multiobjective evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 403–417, Jun. 2016.
- [28] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated F-race: An overview," in *Experimental Methods for the Analysis of Optimization Algorithms*. Berlin, Germany: Springer, 2010, pp. 311–336.
- [29] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Oper. Res. Perspect.*, vol. 3, pp. 43–58, Jan. 2016.
- [30] N. Dang, L. P. Cáceres, P. D. Causmaecker, and T. Stützle, "Configuring irace using surrogate configuration benchmarks," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Berlin, Germany, Jul. 2017, pp. 243–250.
- [31] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *J. Artif. Intell. Res.*, vol. 36, pp. 267–306, Sep. 2009.
- [32] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artif. Intell. Rev.*, vol. 11, nos. 1–5, pp. 193–225, 1997. [Online]. Available: <https://doi.org/10.1023/A:1006556606079>
- [33] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck, "Evolving the structure of evolution strategies," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, 2016, pp. 1–8.
- [34] G. Malkomes and R. Garnett, "Automating Bayesian optimization with Bayesian optimization," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 5984–5994.
- [35] M. Lindauer, M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter, "Towards assessing the impact of Bayesian optimization's own hyperparameters," 2019, *arXiv:1908.06674*.
- [36] M. Lindauer et al., "SMAC3: A versatile Bayesian optimization package for hyperparameter optimization," 2021, *arXiv:2109.09831*.
- [37] A. Saltelli, "Sensitivity analysis for importance assessment," *Risk Anal.*, vol. 22, no. 3, pp. 579–590, 2002.
- [38] W. Hoeffding, "A class of statistics with asymptotically normal distribution," *Ann. Math. Stat.*, vol. 19, no. 3, pp. 293–325, 1948.
- [39] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32. Beijing, China, Jun. 2014, pp. 754–762.
- [40] C. Fawcett and H. H. Hoos, "Analysing differences between algorithm configurations through ablation," *J. Heuristics*, vol. 22, no. 4, pp. 431–458, 2016.

- [41] S. Sheikholeslami, M. Meister, T. Wang, A. H. Payberah, V. Vlassov, and J. Dowling, "AutoAblation: Automated parallel ablation studies for deep learning," in *Proc. 1st Workshop Mach. Learn. (EuroMLSys@EuroSys)*, Edinburgh, U.K., Apr. 2021, pp. 55–61.
- [42] M. López-Ibáñez and T. Stützle, "An experimental analysis of design choices of multi-objective ant colony optimization algorithms," *Swarm Intell.*, vol. 6, pp. 207–232, Jul. 2012.
- [43] J. de Nobel, D. Vermetten, H. Wang, C. Doerr, and T. Bäck, "Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules," in *Proc. Genet. Evol. Comput. Conf. Companion*, 2021, pp. 1375–1384.
- [44] A. Klein, L. C. Tiao, T. Lienart, C. Archambeau, and M. Seeger, "Model-based asynchronous hyperparameter and neural architecture search," 2020, *arXiv:2003.10865*.
- [45] M. Birattari, *Tuning Metaheuristics—A Machine Learning Perspective* (Studies in Computational Intelligence), vol. 197. Berlin, Germany: Springer, 2009.
- [46] Z. Karnin, T. Koren, and O. Somekh, "Almost optimal exploration in multi-armed bandits," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1238–1246.
- [47] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyperparameter optimization," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 24, 2011, pp. 2546–2554.
- [48] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian optimization of machine learning hyperparameters on large datasets," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2017, pp. 528–536.
- [49] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2017, pp. 1487–1495.
- [50] F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl, "YAHPO gym—Design criteria and a new multifidelity benchmark for hyperparameter optimization," 2021, *arXiv:2109.03670*.
- [51] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *J. Global Optim.*, vol. 21, no. 4, pp. 345–383, 2001.
- [52] H. Jalali, I. Van Nieuwenhuysse, and V. Picheny, "Comparison of kriging-based algorithms for simulation optimization with heterogeneous noise," *Eur. J. Oper. Res.*, vol. 261, no. 1, pp. 279–301, 2017.
- [53] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
- [54] R. J. Samworth, "Optimal weighted nearest neighbour classifiers," *Ann. Statist.*, vol. 40, no. 5, pp. 2733–2763, Oct. 2012.
- [55] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.



Julia Moosbauer is currently pursuing the Doctoral degree with the Chair for Statistical Learning and Data Science, Ludwig-Maximilians-Universität München, Munich, Germany.

She is a member of the Munich Center for Machine Learning. Her research focus lies in the interface between automated and explainable machine learning with the goal to increase transparency and trust into automated machine learning systems. She is also interested in hyperparameter optimization algorithms (in particular, Bayesian optimization), algorithm configuration, multiobjective optimization, and (sequential) experimental design.

Martin Binder is currently pursuing the Doctoral degree with the Chair for Statistical Learning and Data Science, Ludwig-Maximilians-Universität München, Munich, Germany.

He is a member of the Munich Center for Machine Learning. He is mostly working on black-box optimization methods for automatic machine learning and hyperparameter optimization, with a focus on multifidelity optimization. He also works on deep learning, specifically self-supervised learning, for genome sequence classification and analysis.



Lennart Schneider is currently pursuing the Doctoral degree with the Chair of Statistical Learning and Data Science, Ludwig-Maximilians-Universität München, Munich, Germany.

His research focuses on automated machine learning, hyperparameter optimization, multiobjective optimization, and neural architecture search.



Florian Pfisterer received the master's degree in statistics from the Ludwig Maximilian University of Munich, Munich, Germany, in 2018, where he is currently pursuing the Ph.D. degree with the Statistical Learning and Data Science Group, Department of Statistics.

His research interests are in the field of automated machine learning, multiobjective optimization, and algorithmic fairness.



Marc Becker received the master's degree in geoinformatics from Friedrich-Schiller University Jena, Jena, Germany, in 2020.

Since 2020, he has been a Research Engineer with the Ludwig Maximilian University of Munich, Munich, Germany, and a Main Developer of the mlr3 optimization packages.



Michel Lang received the Ph.D. degree in statistics, Dortmund Technical University, Dortmund, Germany, in 2015.

He is a former member of the Munich Center for Machine Learning. He is currently a Scientific Manager of the Research Center Trustworthy Data Science and Security, Dortmund, Germany. He is the author of many popular R packages for machine learning and parallelization, e.g., mlr3 or batchtools. His research areas include machine learning, optimization, and software development.



Lars Kotthoff received the Doctoral degree in computer science from the University of St Andrews, St Andrews, U.K., in 2012.

He is an Assistant Professor of Computer Science with the University of Wyoming, Laramie, WY, USA. His research focuses on data-driven combinatorial optimization, automated machine learning, and applying machine learning in other disciplines, for example, materials science.



Bernd Bischl studied computer science, artificial intelligence, and data sciences in Hamburg, Germany; Edinburgh, U.K.; and Dortmund, Germany. He received the Ph.D. degree in statistics from Dortmund Technical University, Dortmund, in 2013 with a thesis on "Model and Algorithm Selection in Statistical Learning and Optimization."

He holds the Chair of Statistical Learning and Data Science with the Department of Statistics, Ludwig-Maximilians-Universität München, Munich, Germany, and he is the Co-Director of the Munich Center for Machine Learning, one of Germany's national competence centers for ML. He is a member of ELLIS, and a Faculty Member of ELLIS Munich, an active developer of several R-packages, leads the "mlr" (Machine Learning in R) Engineering Group and is the Co-Founder of the Science Platform "OpenML" for open and reproducible ML. Furthermore, he leads the Munich branch of the Fraunhofer ADA Lovelace Center for Analytics, Data, and Applications, i.e., a new type of research infrastructure to support businesses in Bavaria, especially in the SME sector. His research interests include AutoML, model selection, interpretable ML, as well as the development of statistical software.

5.4. Mutation is all you need

Contributing article:

L. Schneider, F. Pfisterer, M. Binder, and B. Bischl. Mutation is all you need. In *8th ICML Workshop on Automated Machine Learning, 2021*. <https://openreview.net/forum?id=KFCxuHWNc4>.

Copyright information:

This article is licensed under the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).

Author contributions:

Lennart Schneider is the first author of this manuscript, contributing the following. The project originated as a follow-up to Lennart Schneider’s Master’s Thesis (accessible at https://epub.uni-muenchen.de/77443/1/MA_Schneider_Lennart.pdf), building upon the NAS-Bench-301 Ablation Study section in the Appendix. The manuscript includes several substantial and novel contributions beyond those presented in the Master’s Thesis. Compared to the Master’s Thesis, this manuscript primarily focuses on the impact of the acquisition function optimizer. It provides a comprehensive comparison of mutation against small- and large-budget RS, supported by additional analyses and visualizations of results (specifically Figures 2 and 3). Furthermore, this manuscript introduces additional contributions through additional baselines, including Local Search, BANANAS with $k = 10$ batch proposal, and all BANANAS variants using the posterior predictive mean as the acquisition function. Text passages of the manuscript were newly written and streamlined to emphasize the critical role of the acquisition function optimizer in the performance of BANANAS. All analyses and visualizations presented in the manuscript were re-run and newly created with the inclusion of the new baselines. Lennart Schneider conceptualized the manuscript, performed data curation, conducted the formal analysis, and executed all analyses and created all visualizations of benchmark results. He also developed the code for benchmark experiments and analyses building upon the existing code base established during his Master’s Thesis. Additionally, Lennart Schneider drafted the initial manuscript. Florian Pfisterer, Martin Binder, and Bernd Bischl supervised and provided guidance. All authors collaboratively contributed to the revision process.

Supplementary material available at:

- arXiv version: <https://arxiv.org/abs/2107.07343>

Mutation is all you need

Lennart Schneider
Florian Pfisterer
Martin Binder
Bernd Bischl

LENNART.SCHNEIDER@STAT.UNI-MUENCHEN.DE
FLORIAN.PFISTERER@STAT.UNI-MUENCHEN.DE
MARTIN.BINDER@STAT.UNI-MUENCHEN.DE
BERND.BISCHL@STAT.UNI-MUENCHEN.DE

Department of Statistics, LMU Munich, Germany

Abstract

Neural architecture search (NAS) promises to make deep learning accessible to non-experts by automating architecture engineering of deep neural networks. BANANAS is one state-of-the-art NAS method that is embedded within the Bayesian optimization framework. Recent experimental findings have demonstrated the strong performance of BANANAS on the NAS-Bench-101 benchmark being determined by its path encoding and not its choice of surrogate model. We present experimental results suggesting that the performance of BANANAS on the NAS-Bench-301 benchmark is determined by its acquisition function optimizer, which minimally mutates the incumbent.

1. Introduction

Neural architecture search (NAS) methods can be categorized along three dimensions (Elsken et al., 2019a): search space, search strategy, and performance estimation strategy. Focusing on search strategy, popular methods are given by Bayesian optimization (BO, e.g., Bergstra et al. 2013; Domhan et al. 2015; Mendoza et al. 2016; Kandasamy et al. 2018; White et al. 2019), evolutionary methods (e.g., Miller et al. 1989; Liu et al. 2017; Real et al. 2017, 2019; Elsken et al. 2019b), reinforcement learning (RL, e.g., Zoph and Le 2017; Zoph et al. 2018), and gradient-based algorithms (e.g., Liu et al. 2019; Pham et al. 2018).

Within the BO framework, BANANAS (White et al., 2019) has emerged as one state-of-the-art algorithm (White et al., 2019; Siems et al., 2020; Guerrero-Viu et al., 2021; White et al., 2021). The two main components of BANANAS are a (truncated) path encoding, where architectures represented as directed acyclic graphs (DAG) are encoded based on the possible paths through that graph, and an ensemble of feed-forward neural networks as surrogate model. Recently, White et al. (2021) investigated the performance of different surrogate models in the context of BO-based NAS and concluded that the strong performance of BANANAS on the NAS-Bench-101 benchmark (Ying et al., 2019) is determined by its path encoding and not its choice of surrogate model. Results suggest that path encoding leads to a performance boost on smaller search spaces (such as the one of NAS-Bench-101) but does not scale well on larger search spaces such as DARTS (Liu et al., 2019).

We hypothesize that for larger search spaces, the strong performance of BANANAS stems from its choice of acquisition function optimizer in the sense that local optimization of architectures is most important and other components have less impact on performance. To investigate this hypothesis, we vary the main BANANAS components, namely architecture representation, surrogate model, acquisition function and acquisition function optimizer in a

factorial manner and examine the performance difference on the NAS-Bench-301 benchmark (Siems et al., 2020)¹.

2. BANANAS

BANANAS (White et al., 2019) uses a (truncated) path encoding, combined with an ensemble of feed-forward neural networks as surrogate model, to predict the performance of architectures. Cell-based search spaces such as DARTS can be encoded by representing cells as DAGs, with nodes as vertices and connections with operations between them as edges. For every *path*, i.e., every possible ordering of vertices, a binary feature is generated, indicating whether the DAG contains all directed edges along this path. If architectures are created by sampling edges in the DAG subject to a maximum edge constraint (i.e., limiting the number of edges), most possible paths have a low probability of occurring (White et al., 2019; Ying et al., 2019). Therefore, BANANAS truncates the least-likely paths, resulting in a relatively informative encoding that scales linearly with the size of the cell.

Let \mathcal{A} denote the search space of architectures and $\{f_m\}_{m=1}^M$ denote an ensemble of M feed-forward neural networks (NN)², where $f_m : \mathcal{A} \rightarrow \mathbb{R}$. BANANAS uses independent Thompson sampling (ITS, Thompson 1933; White et al. 2019) as acquisition function:

$$\alpha_{\text{ITS}}(x) = \tilde{f}_x(x), \quad \tilde{f}_x(x) \sim \mathcal{N}(\hat{f}, \hat{\sigma}^2), \quad (1)$$

where $\hat{f} = \frac{1}{M} \sum_{m=1}^M f_m(x)$ and $\hat{\sigma} = \sqrt{\frac{\sum_{m=1}^M (f_m(x) - \hat{f})^2}{M-1}}$. $\alpha_{\text{ITS}}(\cdot)$ is then optimized using the following mutation algorithm (**Mut**): The best performing architecture so far is selected and mutated in 100 different ways by changing a single operation or edge randomly and the architecture yielding the largest acquisition value is proposed as the next candidate for evaluation.

3. Experiments

To investigate the effectiveness of different components of BANANAS on NAS-Bench-301, we conducted a series of experiments where we replaced some of them with what we consider more “standard” choices. A simpler configuration could use a random forest (**RF**, Breiman 2001; notably used successfully in SMAC, Hutter et al. 2011) as a surrogate model which can either be fitted to path encodings (**Path**) or natural tabular representations (**Tabular**) of the architectures as provided in NAS-Bench-301 in the form of a `ConfigSpace` (see the `ConfigSpace` library, Lindauer et al. 2019). In the tabular encoding, architectures are represented by enumerating all nodes and potential edges and introducing categorical hyperparameters for each operation along each potential edge, where the nodes serving as input of each intermediate node are again defined as categorical hyperparameters and operations on a certain edge can only be specified if this edge is actually present in the DAG

1. NAS-Bench-301 uses architectures of the DARTS search space trained and evaluated on CIFAR-10 (Krizhevsky, 2009)

2. White et al. (2019) use $M = 5$ sequential fully-connected networks with 10 layers of width 20 by default, initialized with different random weights and trained using permuted training sets, the Adam optimizer with a learning rate of 0.01, and mean absolute error (MAE) loss

(Siems et al., 2020). Note that another possible architecture representation is given by adjacency matrix encoding (Ying et al., 2019; White et al., 2020a), which was not considered by us. Looking at the acquisition function, the expected improvement (EI) is a well-known alternative:

$$\alpha_{\text{EI}}(x) = E_y[\max(y - y_{\text{max}}, 0)], \quad (2)$$

given in Jones et al. (1998), where in our context y_{max} is the best validation accuracy observed so far and y is the surrogate prediction of architecture x . As a very simple alternative, one could also only be interested in the posterior mean prediction (**Const. Mean**) as acquisition function, which does not take the surrogate model uncertainty estimates into account. Finally, looking at acquisition function optimizers, a popular choice is given by random search (RS): Drawing a large number of architectures uniformly at random (e.g., by sampling from the ConfigSpace) and selecting the architecture with the largest acquisition value. Our RS method samples 1000 architectures in each BO iteration.

3.1 Different BANANAS Configurations on NAS-Bench-301

Choices for the architecture encodings, surrogate candidates, acquisition functions, and acquisition function optimizers were crossed in a full factorial manner (where possible), resulting in overall 18 different algorithms. BANANAS, local search (LS) and random search (as NAS method, **Random**) were used as implemented in **naszilla** (White et al., 2020a). In LS (White et al., 2020b), all neighbors (e.g., all architectures differing in one operation or edge) of an incumbent are evaluated and the incumbent is replaced if a better architecture has been found and the process is repeated until no better architecture can be found (i.e., a local optimum is reached) or another termination criterion is met. Regarding the reference BANANAS implementation, two configurations were used differing in the frequency of updating their ensemble of feed-forward networks ($k = 1$, i.e., after every iteration, or $k = 10$, see White et al. 2019). The initial design for all methods consisted of ten architectures that were sampled uniformly at random (note that LS and **Random** do not rely on an initial design and simply start from zero evaluations). All methods were run for 100 iterations (architecture evaluations) and all runs were replicated 20 times. Results are shown in Figure 1, where the validation accuracy is plotted against the batch number. Note that in each facet, the reference **naszilla** implementations of BANANAS, LS, and **Random** are provided and by design, **Paths + NN + ITS + Mut** is a (re-)implementation of the BANANAS ($k = 1$) configuration. In general, using **Mut** as acquisition function optimizer always results in a strong performance boost compared to using RS. Notably, BANANAS’ ensemble of feed-forward neural networks, together with path encoding only performs well if combined with **Mut** and is otherwise outperformed by **Random**. Moreover, the very simple configuration of **Tabular + RF + EI + Mut** performs similarly to the reference BANANAS implementation. Finally, neglecting all uncertainty in the predictions by opting for the **Const. Mean** acquisition function results in very good performance when combined with **Tabular + RF + Mut**. Performing a one-way ANOVA on the top seven algorithms indicated no significant difference in final performance, $F(6, 133) = 1.026, p = 0.411$. Table 1 presents results of a four-way ANOVA on the final performance of the 18 algorithms outlined above with respect to the factors architecture encoding, surrogate candidate, acquisition

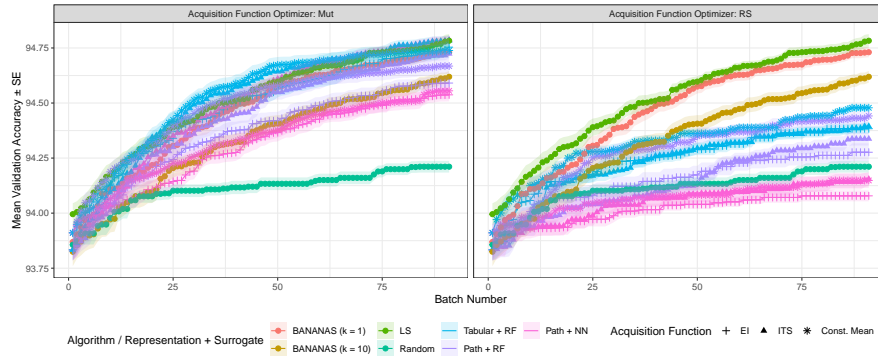


Figure 1: Different BANANAS configurations on NAS-Bench-301. Mean validation accuracy with standard error bands, higher is better. Color: optimization method and surrogate model. Facet: acquisition function optimizer, where applicable. Point shape: acquisition function, where applicable. The ITS acquisition function and `Mut` acquisition function optimizer is used for BANANAS methods, and `LS` and `Random` do not use an acquisition function; their accuracy is therefore shown in both facets of the graph.

function, and acquisition function optimizer. The acquisition function optimizer is by far the most important determinant of final performance.

	Sum Sq	Df	F value	Pr(>F)
Architecture Encoding	0.41	1	19.57	0.0000
Surrogate Candidate	1.01	1	48.31	0.0000
Acquisition Function	0.56	2	13.49	0.0000
Acq. F. Optimizer	13.18	1	632.43	0.0000
Residuals	7.38	354		

Table 1: Results of a four-way ANOVA on the factors architecture encoding, surrogate candidate, acquisition function, and acquisition function optimizer. Type II sums of squares.

3.2 Examining the Effect of the Acquisition Function Optimizer

To investigate the performance difference with respect to the acquisition function optimizers, another experiment was conducted. Based on the `Tabular + RF + EI` configuration three different acquisition function optimizers were compared: Random search with 100000 architectures drawn uniformly at random in each BO iteration (`RS+`), random search as described above (`RS`) and `Mut` as described above. Ten architectures were sampled uniformly at random and used as the initial design points for all replications. All methods were run for 100 iterations (architecture evaluations) and all runs were replicated 20 times. Results are given in Figure 2A. As can be seen, `Mut` strongly outperforms even the `RS+` optimizer.

We collected additional data in the `RS+` runs shown in Figure 2A. In each BO iteration of these runs, we also performed acquisition function optimization using the other two methods (`RS` and `Mut`) and investigated the properties of the proposed architectures. While the op-

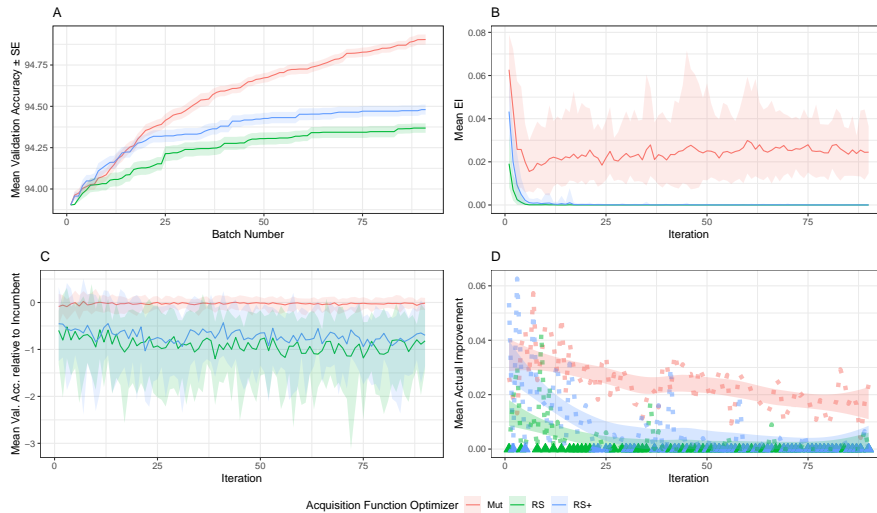


Figure 2: Tabular + RF + EI with different acquisition function optimizers on NAS-Bench-301. A: Validation accuracy. B: EI. C: Validation accuracy relative to the incumbent. D: Actual improvement. Ribbons in B and C represent 2.5% and 97.5% quantiles. In D, LOESS smoothing was performed and triangles indicate no improvement.

timization itself proceeded with the architectures proposed by RS+, the collected data gives information about the quality of architecture proposals done by the other methods. The data collected was the EI of each proposed architecture, according to the surrogate model (Figure 2B), the actual validation accuracy of each proposed architecture (when evaluated), minus the validation accuracy of the incumbent during that iteration (Figure 2C), and that same quantity, conditional on the proposed architecture giving higher validation accuracy than the incumbent (“actual improvement”, Figure 2D).

Mut results in both higher EI and actual improvement, i.e., Mut solves the inner optimization problem better than the other optimizers and the actual improvement is comparably large. Note that the difference between the validation accuracy of proposed architectures and incumbent is mostly negative due to a fixed iteration seldom resulting in actual improvement. Looking at Figure 2D, we observe that following the proposals by RS+ and RS results in many iterations with no improvement (as indicated by triangles).

In a final experiment, focus was given to the accuracy of the surrogate model when predicting the validation accuracy of architectures depending on the edit distance to the incumbent. Based on the Tabular + RF + EI + Mut configuration, the BO loop was run for 50 iterations (architecture evaluations); the construction of the initial design remained the same and all runs were replicated 100 times. For edit distances ranging from 1 to 8, 100 test architectures were constructed each by mutating a fixed number of parameters (operations or edges) of the incumbent. For these test architectures, Kendall’s τ with respect to the predicted and true validation accuracy (after evaluation) is given in Figure 3A.

Additionally, the true validation accuracy is plotted against the edit distance (Figure 3B), with the gray point representing the incumbent. In Figure 3C, the expected improvement and the actual improvement is plotted. While the true validation accuracy decreases when increasing the edit distance, Kendall’s τ increases, suggesting that the surrogate model is not capable of precise performance prediction for high performing architectures close to the incumbent. This finding goes in line with results of White et al. (2021) that model based NAS methods perform bad when predicting the performance of neighbors of high performing architectures when the search space is large. Moreover, the expected improvement is relatively unaffected by the edit distance, although the actual improvement is largest for close architectures. This may indicate that thorough optimization of the acquisition function is not needed, instead simply considering neighboring architectures as candidates may be sufficient.

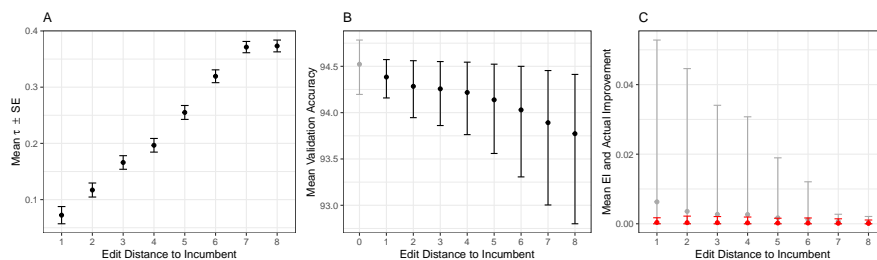


Figure 3: **Tabular + RF + EI + Mut** on NAS-Bench-301. A: Kendall’s τ of the predicted and true validation accuracy of test architectures constructed to have different edit distances to the incumbent. B: True validation accuracy of these test architectures. Validation accuracy of the incumbent is given in gray. C: Expected Improvement (red) and actual improvement (gray) of these test architectures. Bars in B and C represent 2.5% and 97.5% quantiles.

4. Discussion

We have presented empirical results suggesting that the performance of BANANAS on large cell-based search spaces such as DARTS is predominantly determined by its choice of acquisition function optimizer that is effectively performing a randomized local search. Other components such as the architecture encoding, surrogate model and acquisition function have a comparably small effect on the performance, and exchanging most components of BANANAS with more “standard” choices results in a method that is not significantly worse. Local search, which uses no surrogate model at all, does in fact perform equally well (at least on the NAS-Bench-301 benchmark), giving more evidence that the local nature of BANANAS’ mutation acquisition function optimization contributes mainly to its success. Minimally mutating the incumbent allows for solving the inner acquisition function optimization problem better than random search variants with large budget, although the surrogate model suffers from imprecise surrogate predictions for architectures close in edit distance to the incumbent. Future work on BO methods for NAS should therefore also focus on algorithms for solving the inner acquisition function optimization problem.

References

- M. Becker, J. Richter, M. Lang, B. Bischl, and M. Binder. *bbotk: Black-Box Optimization Toolkit*, 2021. <https://bbotk.ml-org.com>, <https://github.com/mlr-org/bbotk>.
- H. Bengtsson. A unifying framework for parallel and distributed processing in R using futures. arXiv:2008.00553 [cs.DG], 2020.
- J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, pages 115–123, 2013.
- M. Binder, F. Pfisterer, L. Schneider, B. Bischl, M. Lang, and S. Dandl. *mlr3pipelines: Preprocessing Operators and Pipelines for 'mlr3'*, 2020. URL <https://CRAN.R-project.org/package=mlr3pipelines>. R package version 0.3.0.
- L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, page 3460–3468, 2015.
- M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2021. URL <https://CRAN.R-project.org/package=data.table>. R package version 1.14.0.
- T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019a.
- T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via Lamarckian evolution. In *Proceedings of the International Conference on Learning Representations*, 2019b.
- J. Guerrero-Viu, S. Hauns, S. Izquierdo, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M Lindauer, and F. Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. arXiv:2105.01015 [cs.LG], 2021.
- F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523, 2011.
- D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.
- K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.
- A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

- M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019.
- M. Lindauer and F. Hutter. Best practices for scientific research on neural architecture search. arXiv:1909.02453 [cs.LG], 2019.
- M. Lindauer, K. Eggenesperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter. BOAH: A tool suite for multi-fidelity Bayesian optimization & analysis of hyperparameters. arXiv:1908.06756 [cs.LG], 2019.
- H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the International Conference on Learning Representations*, 2017.
- H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.
- H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In *ICML Workshop on Automatic Machine Learning*, 2016.
- G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.
- H. Pham, M. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4095–4104, 2018.
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL <https://www.R-project.org/>.
- E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, page 2902–2911, 2017.
- E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019.
- J. Richter, M. Becker, M. Lang, B. Bischl, M. Binder, and J. Moosbauer. *mlr3mbo: Flexible Bayesian Optimization in R*, 2021. <https://mlr3mbo.ml-org.com>, <https://github.com/mlr-org/mlr3mbo>.
- J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. arXiv:2008.09777 [cs.LG], 2020.

- W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.
- K. Ushey, JJ. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2020. URL <https://CRAN.R-project.org/package=reticulate>. R package version 1.18.
- C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. arXiv:1910.11858 [cs.LG], 2019.
- C. White, W. Neiswanger, S. Nolen, and Y. Savani. A study on encodings for neural architecture search. In *Proceedings of the 34th Conference on Neural Information Processing Systems*, 2020a.
- C. White, S. Nolen, and Y. Savani. Local search is state of the art for neural architecture search benchmarks. In *ICML Workshop on Automatic Machine Learning*, 2020b.
- C. White, A. Zela, B. Ru, Y. Liu, and F. Hutter. How powerful are performance predictors in neural architecture search? arXiv:2104.01177 [cs.LG], 2021.
- M. N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017.
- C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7105–7114, 2019.
- B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2017.
- B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

Appendix A. Computational Details

The BO algorithms were implemented in R (R Core Team, 2020) within the `mlr3` (Lang et al., 2019) ecosystem relying on `mlr3mbo` (version 0.0.0.9999; Richter et al. 2021) and `bbotk` (version 0.3.0.9999; Becker et al. 2021). Random forests were used as implemented in the `mlr3extralearners` package wrapping `ranger::ranger` (version 0.12.1; Wright and Ziegler 2017) with `num.trees` set to 500, `se.method` set to "jack", and `respect.unordered.factors` set to "order". Missing values were encoded with a new level "missing" via a preprocessing pipeline built using `mlr3pipelines` (version 0.3.0; Binder et al. 2020).

Python 3.8.7 was used via the `reticulate` package (version 1.18; Ushey et al. 2020) within R. For NAS-Bench-301, `nasbench301` version 0.2 (Siems et al., 2020) was used relying on the `xgb_v1.0` surrogate model for the validation accuracy. The feed-forward ensemble of neural networks and path encoding as used by BANANAS was directly adopted as implemented in `naszilla` (version 1.0; White et al. 2020a). BANANAS, local search and random search (as NAS methods) were run using `naszilla` employing the same `nasbench301` setup as described above under Python 3.6.12 (due to different module requirements).

All computations were performed on 2 Intel[®] Xeon[®] E5-2650 v2 @ 2.60GHz CPUs each with 16 threads using R 4.0.3 under Ubuntu 20.04.1 LTS. Parallelization in R was done via the `future` (Bengtsson, 2020) and `future.apply` (Bengtsson, 2020) packages (version 1.21.0 and 1.7.0) on top of the internal parallelization of the `data.table` (Dowle and Srinivasan, 2021) package (version 1.14.0).

Appendix B. NAS Best Practices Checklist

Here, we answer to applicable questions of the NAS best practices checklist (version 1.0), see Lindauer and Hutter (2019).

- as NAS benchmark, NAS-Bench-301 (`nasbench301`) version 0.2 was used relying on the `xgb_v1.0` surrogate model (deterministic) for the validation accuracy
- all computations were run on the same hardware (2 Intel[®] Xeon[®] E5-2650 v2 @ 2.60GHz CPUs)
- all results reported are based on ablation studies
- the same evaluation protocol was used for all methods
- performance was compared with respect to the number of architecture evaluations
- random search was included as a NAS method
- multiple runs (20 or 100) were conducted; reproducibility with respect to algorithms implemented in R is given due to an initial random seed being set; regarding `naszilla`, no seed can be explicitly set

5.5. Neural Networks as Black-Box Benchmark Functions Optimized for Exploratory Landscape Features

Contributing article:

R. P. Prager*, K. Dietrich*, L. Schneider, L. Schäpermeier, B. Bischl, P. Kerschke, H. Trautmann, and O. Mersmann. Neural networks as black-box benchmark functions optimized for exploratory landscape features. In *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 129–139, 2023. <https://dl.acm.org/doi/10.1145/3594805.3607136>.

Copyright information:

This article is licensed to ACM (<https://www.acm.org/publications/policies/publication-rights-and-licensing-policy>). Copyright is held by the authors.

Author contributions:

Lennart Schneider served as a co-author of this manuscript, contributing the following. The project idea was conceptualized by Raphael Patrick Prager, Konstantin Dietrich, Lennart Schneider, Lennart Schäpermeier, Pascal Kerschke, and Heike Trautmann. Methodological approaches, including the development and refinement of how to generate the point clouds, optimizing objective values via CMA-ES, and how to train surrogate models on them, were designed by Raphael Patrick Prager, Konstantin Dietrich, Lennart Schneider, and Lennart Schäpermeier. Data collection and curation were carried out by Raphael Patrick Prager and Konstantin Dietrich, who also conducted the formal analyses. Raphael Patrick Prager and Konstantin Dietrich further analyzed all collected data and designed the benchmark study to assess optimizer performance on newly constructed benchmark functions, contrasting this with established synthetic benchmarking functions. Raphael Patrick Prager and Konstantin Dietrich wrote all code used for the collection of data and the analyses of benchmarking results. Visualizations were created jointly by Raphael Patrick Prager and Konstantin Dietrich. Supervision was conducted by Heike Trautmann, Pascal Kerschke, Bernd Bischl, and Olaf Mersmann. The initial draft of the manuscript was written collectively by Raphael Patrick Prager, Konstantin Dietrich, Lennart Schneider, and Lennart Schäpermeier. Raphael Patrick Prager, Konstantin Dietrich, Lennart Schneider, Lennart Schäpermeier, Heike Trautmann, and Pascal Kerschke contributed collaboratively to the revision process.

Supplementary material available at:

- Code and data: https://github.com/Reiyan/ela_nn_function_generation



Neural Networks as Black-Box Benchmark Functions Optimized for Exploratory Landscape Features

Raphael Patrick Prager*
University of Münster
Münster, NRW, Germany
raphael.prager@wi.uni-muenster.de

Konstantin Dietrich*
TU Dresden & Center for Scalable
Data Analytics and Artificial
Intelligence (ScaDS.AI)
Dresden, Saxony, Germany
konstantin.dietrich@tu-dresden.de

Lennart Schneider
LMU Munich & Munich Center for
Machine Learning (MCML)
Munich, Bavaria, Germany
lennart.schneider@stat.uni-
muenchen.de

Lennart Schäpermeier
TU Dresden & Center for Scalable
Data Analytics and Artificial
Intelligence (ScaDS.AI)
Dresden, Saxony, Germany
lennart.schaepemeier@tu-
dresden.de

Bernd Bischl
LMU Munich & Munich Center for
Machine Learning (MCML)
Munich, Bavaria, Germany
bernd.bischl@stat.uni-muenchen.de

Pascal Kerschke
TU Dresden & Center for Scalable
Data Analytics and Artificial
Intelligence (ScaDS.AI)
Dresden, Saxony, Germany
pascal.kerschke@tu-dresden.de

Heike Trautmann
University of Münster
Münster, NRW, Germany
University of Twente
Enschede, Overijssel, Netherlands
trautmann@wi.uni-muenster.de

Olaf Mersmann
TH Köln
Gummersbach, NRW, Germany
olaf.mersmann@th-koeln.de

ABSTRACT

Artificial benchmark functions are commonly used in optimization research because of their ability to rapidly evaluate potential solutions, making them a preferred substitute for real-world problems. However, these benchmark functions have faced criticism for their limited resemblance to real-world problems. In response, recent research has focused on automatically generating new benchmark functions for areas where established test suites are inadequate. These approaches have limitations, such as the difficulty of generating new benchmark functions that exhibit exploratory landscape analysis (ELA) features beyond those of existing benchmarks.

The objective of this work is to develop a method for generating benchmark functions for single-objective continuous optimization with user-specified structural properties. Specifically, we aim to demonstrate a proof of concept for a method that uses an ELA feature vector to specify these properties in advance. To achieve this, we begin by generating a random sample of decision space variables and objective values. We then adjust the objective values using CMA-ES until the corresponding features of our new problem

match the predefined ELA features within a specified threshold. By iteratively transforming the landscape in this way, we ensure that the resulting function exhibits the desired properties. To create the final function, we use the resulting point cloud as training data for a simple neural network that produces a function exhibiting the target ELA features. We demonstrate the effectiveness of this approach by replicating the existing functions of the well-known BBOB suite and creating new functions with ELA feature values that are not present in BBOB.

CCS CONCEPTS

• **Theory of computation** → **Design and analysis of algorithms**; **Continuous optimization**;

KEYWORDS

Exploratory Landscape Analysis, Benchmarking, Instance Generator, Black-Box Continuous Optimization, Neural Networks

ACM Reference Format:

Raphael Patrick Prager, Konstantin Dietrich, Lennart Schneider, Lennart Schäpermeier, Bernd Bischl, Pascal Kerschke, Heike Trautmann, and Olaf Mersmann. 2023. Neural Networks as Black-Box Benchmark Functions Optimized for Exploratory Landscape Features. In *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA '23)*, August 30–September 1, 2023, Potsdam, Germany. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3594805.3607136>

1 INTRODUCTION

The development of optimization algorithms creates a natural need for test problems to assess their search behaviour, robustness and

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

FOGA '23, August 30–September 1, 2023, Potsdam, Germany

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0202-0/23/08...\$15.00

<https://doi.org/10.1145/3594805.3607136>

overall performance. The choice of test problems is far from trivial and will determine the course of development. Ideally, we would like to guide this process by confronting the algorithm with challenging and representative problems that resemble problems we encounter in real-world settings. An impeding factor is that those real-world problems are often very expensive to evaluate and subject to companies' proprietary information. Standardized artificial benchmark suites, like the Black-Box Optimization Benchmark (BBOB) suite [6], the annually changing CEC benchmark (e.g. [37]), and to some degree Nevergrad [31], strive to alleviate this issue by offering a wide range of different problem instances. These suites contain a set of hand-picked artificial problems that are chosen to cover a range of different problem properties such as a varying degree of multi-modality, presence of global and funnel structures, and so forth. While this approach is not without merit, it begs the question whether these suites are sufficiently diverse and resemble prominent real-world problems as a whole. Research endeavours such as [21] and [34] show that this might not always be the case. This in turn may bias the development and evaluation of algorithms where they achieve competitive results on artificial benchmarks but are effectively never used in practice. As algorithm selection and configuration become increasingly automated [17, 18, 29], there is a rising incentive to further increase the heterogeneity of solvers from which a model can choose from. Given these pitfalls concerning artificial benchmarks in general, it is not surprising that several approaches have been developed to address these issues [3, 4, 19, 25].

One of the earliest works to build a tune-able landscape generator was presented by Gallagher and Yuan [4]. The generator uses a set of Gaussian functions and a small number of parameters that can be linked to a problem's geometric properties. This allowed them to achieve new insights on the behavior of some estimation of distribution algorithms, indicating the usefulness and need for a feature driven problem generator.

Another approach has been developed by Lang and Engelbrecht [19]. In their work, they construct a novel benchmark suite by systematically evaluating existing benchmark functions originating from a multitude of different suites. They used exploratory landscape analysis (ELA) [24] to test how diverse each of these functions really is w.r.t. a subset of ELA features. Using self-organized feature maps, they construct a space spanned by the ELA features. They aim to maximize the coverage of this space by selecting the most appropriate functions out of their pool of benchmark functions for each cell map. The only drawback of their approach is that they are not proposing a mechanism to generate entirely new functions.

Muñoz and Smith-Miles [25] presented another approach. In a similar fashion to [19], they use ELA features to evaluate the diversity of a single benchmark (BBOB). This allowed them to characterize every function using an eight-dimensional feature vector. By projecting the vectors into a two-dimensional instance space, they could identify uncovered areas and thereby identify target ELA feature vectors. New functions that exhibit the respective target ELA vector properties are then generated making use of genetic programming. They achieve good results in being able to interpolate within and even extrapolate beyond the convex hull that the BBOB suite spans in the instance space.

Nevertheless, as recently pointed out by Dietrich and Mersmann [3], there are some downsides to the results of Muñoz and Smith-Miles [25]. For one, there is a lack of knowledge about the global optima of genetically programmed functions. But the high computational cost of genetic programming weighs more severely. While Dietrich and Mersmann [3] were able to get rid of both these downsides by using affine recombinations of the BBOB functions as new benchmark problems, this approach could only interpolate within the convex hull of the ELA feature space within BBOB.

The main contribution of this paper is that we address this shortcoming by proposing a new, neural-network based, method for generating novel problem instances with an arbitrary property combination w.r.t. the chosen ELA features. At the same time, we retain the auspicious aspects of [25] and [3]. In other words, our devised approach is not only able to interpolate but also to extrapolate beyond the problem space and thereby can potentially generate truly novel problem instances in less time.

This paper can be compartmentalized into two distinct sets of experiments. The first experiments focus on validating and demonstrating that our method works reasonably well in principle. We accomplish this by trying to emulate certain benchmark functions for which we sample and compute a so-called target ELA feature vector. We start by creating a random sample in the decision space and random objective values corresponding to each observation in our sample. We optimize these objective values until they exhibit the desired ELA values which are determined by the target ELA feature vector. The resulting point cloud contains our anchor points to generate a new benchmark problem. To construct the new problem we then make use of a simple neural network which is trained on this optimized point cloud. In order to show that the resulting functions mimic the existing ones well, we compare the behavior of optimization algorithms on both sets. This gives us the opportunity to investigate where our devised approach excels and where it encounters issues by comparing the emulated landscape with the original one as well as the algorithm rankings between these two. The second set of experiments highlights the potential of our devised approach to create entirely novel functions which are not represented in our selected benchmark suite.

The remainder of this paper is structured as follows. In Section 2, we give a general overview of ELA as well as a justification for the selected features and some technical details for their calculation. Section 3 provides a full account of our devised approach where the construction of the aforementioned point cloud is subject of Subsection 3.1, and the surrogate models are discussed in Subsection 3.2. In Section 4, we validate our approach by imitating functions from the BBOB suite, discussing the results from the landscape perspective in Subsection 4.1 and from an algorithm performance perspective in Subsection 4.2. Our general workflow is then evaluated by generating functions for ELA feature vectors which are not part of the chosen benchmark suite in Subsection 4.3. Finally, we conclude our paper in Section 5 and provide an outlook on future research opportunities based on our findings.

2 EXPLORATORY LANDSCAPE ANALYSIS

While the hardness and properties of an optimization problem, given enough samples, may be characterized visually for up to two-dimensional problems, visualizations of higher-dimensional problems are generally infeasible. Thus, other mechanisms are required to identify problem properties, such as the degree of multi-modality or the presence of global and funnel structures [16, 23, 24]. These properties ultimately define the hardness of a problem. The aforementioned mechanisms manifest themselves in a set of numerical features, which, in the single-objective continuous optimization domain, are consolidated under the term exploratory landscape analysis (ELA) [24].

In essence, ELA is a collection of heterogeneous methods to extract quantitative information about a black-box optimization problem. The majority of ELA features can be computed based on a fixed sample of randomly generated points from the search space, along with their respective (evaluated) objective values. Different sampling procedures have been investigated over the years [32]. The sampling size, on the other hand, often depends on the scope of the analysis. In automated algorithm selection [14], the sampling size must be competitive and, therefore, small, whereas theoretical undertakings can be more lavish (e.g., [3, 11, 33]). Regardless of the scope, the sampling size is typically scaled with the dimensionality d of the problem instance. In this work, we use a sampling size of $250 \cdot d$, which balances between a sufficient coverage of the search space together with keeping the generation of new problem instances computationally less intensive. This size is also recommended by [33] to correctly classify all BBOB problems. For our work, we choose Latin hypercube sampling as the sampling strategy. Furthermore, for any given sample we normalize the objective values to the range of $[0, 1]$ via their respective sample minimum and maximum. This has two desirable effects. Firstly, it recently has been shown that not all ELA features are invariant to linear transformations of the objective space [30]. The proposed method to deal with this issue is to normalize the objective values. Secondly, a lower and upper bound of zero and one alleviates certain problems for our neural networks which act as surrogate models in the consecutive section. This allows us for instance to utilize a sigmoid activation function in the output layer which naturally maps all values in the interval $[0, 1]$.

Up to this point, we only discussed details pertaining to the ELA sample. But ultimately, problem hardness is subject to the properties of the fitness landscape making the selection of suitable ELA features another crucial aspect of this work. We adapt a semi-structured approach. Meaning, we utilize the findings of [33] and [35] and iteratively remove and add certain hand-picked features to improve the landscape properties of our generated functions. The final chosen ELA features are listed below and computed with the Python package `pf1acco`^{*}:

ela_meta.lin_simple.adj_r2 Adjusted coefficient of determination of the linear regression model without variable interactions [24].

ela_meta.lin_w_interact.adj_r2 Adjusted coefficient of determination of the linear regression model with variable interactions [24].

ela_meta.quad_simple.adj_r2 Adjusted coefficient of determination of the quadratic regression model without variable interactions [24].

ela_meta.quad_w_interact.adj_r2 Adjusted coefficient of determination of the quadratic regression model with variable interactions [24].

ela_distr.skewness Skewness of the sample's objective values [24].

nbc.nb_fitness.cor The correlation between the fitness values of the search points and their indegree in the nearest-better point graph [15].

nbc.nn_nb.sd_ratio Ratio of the standard deviation of all nearest neighbor distances to the standard deviation of all nearest better distances [15].

fitness_distance.fitness_std Standard deviation of the sample's objective values [12].

3 BLACK-BOX FUNCTION GENERATION

The generation of novel problem instances consists of three steps. Conceptually, we want to (1) **identify a target vector of ELA features**, which occupies sparse regions of an existing benchmark to enhance its diversity. Meaning, this target vector should be constituted of vastly different ELA feature values compared to the values of ELA features of any given benchmark suite. Note that the exact identification of this target vector is not the focus of this work. However, for five novel ELA vectors we demonstrate this generation procedure is able to generate functions with landscape structures arguably different to BBOB. In addition, we compute these target ELA feature vectors on the problem instances of BBOB. This gives us the opportunity to discern how successful our developed approach in general is by comparing our generated function to the existing BBOB instance which has served as a target. Given this target ELA vector, we (2) **generate a sample of points** (i.e., a point cloud) through an optimization process that exhibits the desired ELA vector values. Once the similarity of the values is satisfactory, we (3) **build a surrogate model** based on the found point cloud.

The entire process is depicted in Figure 1. In the following, we will discuss the second and third components in more detail.

3.1 Optimization of Point Clouds

Given a target ELA feature vector \mathbf{a} (which is constituted of the aforementioned 8 features) that we want to emulate, we first create a random sample $X \in \mathbb{R}^{n \times d}$ in the decision space and generate its objective values $\mathbf{y} \in \mathbb{R}^n$ randomly in the interval $[0, 1]$. We refer to the tuple (X, \mathbf{y}) as the *point cloud*. For this current iteration of the point cloud, we then compute the corresponding ELA feature vector $\mathbf{b} = h(X, \mathbf{y})$, where h is the set of functions required to calculate the ELA features of interest based on the point cloud (X, \mathbf{y}) . Within our optimization procedure, we then *only* adjust the objective values \mathbf{y} such that the distance between the ELA feature vectors \mathbf{b} and \mathbf{a} is minimal. Hence, we can formally define the function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ to generate an adequate point cloud, where we strive to find a $\mathbf{y}^* \in \arg \min_{\mathbf{y}} f(\mathbf{y})$ such that the distance between the two ELA feature vectors \mathbf{a} and \mathbf{b} is minimal:

$$f(\mathbf{y}) = \|h(X, \mathbf{y}) - \mathbf{a}\|_2 \quad (1)$$

^{*}<https://github.com/reiyan/pf1acco>

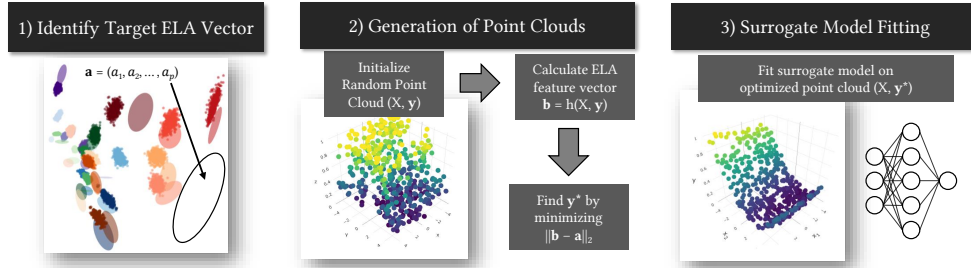


Figure 1: High-level process description of our approach to generate arbitrary black-box problems. In 1), an ELA target vector is identified in sparse regions of an existing benchmark suite. In 2), we generate a random point cloud, calculate the respective ELA features and minimize between the former and the latter by changing y . In 3), we use the optimized point cloud as the basis for fitting our surrogate model.

The global optimum in such a case is known to be zero since distances between two objects w.l.o.g. are in the interval $[0, \infty)$. The dimensionality of this problem is governed by the sample size n of the point cloud. In our experiments, we use a sample size of $250 \cdot d$, which scales linearly with the dimensionality d of the problem we want to create. Meaning if we want to create a two-dimensional problem instance, the dimensionality of our minimization problem is $250 \cdot 2 = 500$. It is apparent that these problems can become extremely high-dimensional. Here, the work of [38] provides valuable insight into this matter. The authors show that the conventional CMA-ES [7] still achieves competitive results even for higher-dimensional problems. While our problem dimension exceeds their limit of up to 320, we argue that their findings are still relevant for and can be extrapolated to our case, i.e., we use the CMA-ES to solve the optimization problem given in Equation 1. Each attempt to generate a point cloud is allocated a budget of $200 \cdot n$, where n is simultaneously the dimensionality of our optimization problem of Equation 1 and by extension the size of our point cloud. The required CPU time is dependent (1) on the time it requires to calculate $f(y)$ (i.e., ELA feature calculation) for each candidate solution and (2) on the time necessary to update the covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$. Generating a single $2d$ point cloud takes up to 4 CPU hours whereas a single $3d$ point cloud demands almost 3 CPU days.

Until now, as hinted at in Section 2, we only normalize the objective values of any given sample by applying min-max normalization. While ELA features value ranges become less prone to having unusually small or large values, it does not account for the different scales *between* distinct ELA features. This biases the search trajectory of the CMA-ES and places more importance on certain ELA features without justifiable reason. Therefore, we experimentally determined the minima and maxima of each ELA feature where no theoretical lower or upper bound can be determined (e.g. the upper bound of any R^2 value is 1) and use these values to apply min-max normalization to our ELA features during the optimization process.

3.2 Surrogate Model Fitting

To construct a novel function based on a given optimized point cloud (X, y^*) , we use neural networks (NNs) as surrogate model $m: \mathbb{R}^{n \times d} \rightarrow \mathbb{R}$, $m(x) = \hat{y}$ fitted on $\mathcal{D} = \{x^{(i)}, y^{*(i)}\}_{i=1, \dots, n}$, where $x^{(i)}$ is the i -th row of X and $y^{*(i)}$ the i -th element of y^* . We use a simple feed forward architecture consisting of one hidden layer with 512 units and a tanh activation function. As a final output we use a sigmoid activation function (due to objective values being normalized via min-max normalization and we want the predictions of the model m to be on the same scale).

On the one hand, our choice of surrogate model is motivated based on theoretical properties, as feed forward NNs are known to be universal function approximators, i.e., already a feed forward NN with a single hidden layer can approximate any continuous function to any desired accuracy, given arbitrary width (number of hidden units) [10]. On the other hand, NNs have desirable practical properties:

- (i) They scale well with the number of data points (which will become relevant for higher dimensions).
- (ii) They are easy to deploy and integrate in benchmarking suites.
- (iii) They induce very little latency overhead during evaluation, i.e., a forward pass.

All in all, this makes NNs very attractive as surrogate models in our scenario compared to other regression models such as generalized additive models, splines, support vector machines, tree-based models or Gaussian Processes. Still, one has to be aware that NNs have a strong inductive bias to learn smooth functions (see, e.g., [5]) – especially when using tanh and sigmoid activation functions – which can only be influenced to a certain degree by architectural design choices.

Our explicit training procedure looks like the following: As our goal is to perfectly interpolate \mathcal{D} , we perform gradient descent using the mean squared error as a loss function and AdamW as optimizer with default parameters and a learning rate of 0.001

for 5000 epochs. The objective here is to perfectly interpolate the training data and essentially overfit drastically. Only under these circumstances we can guarantee that the resulting model m will exhibit the same (or reasonably similar) target ELA feature vector \mathbf{a} when evaluated at the anchor points X , i.e., $\|h(X, \hat{y}) - \mathbf{a}\|_2$ is sufficiently small[†]. Note that we do not employ mini-batches but use a single batch of all available data - again due to our goal to heavily overfit the training data. We implement our NNs in PyTorch [27]. We provide the required code to replicate our experiments on GitHub[‡].

4 RESULTS

4.1 Landscape Perspective

To validate our devised methodology, we use ELA feature vectors generated based on the functions of the Black-Box Optimization Benchmark (BBOB) suite [6] as targets. The BBOB suite consists of 24 different functions, conveying distinct challenges to numeric optimization algorithms. Each function (FID) belongs to one of five so-called function groups, which share the same major characteristics, and is instantiated using a set of transformations such as rotations and shifts, yielding different function instances with fundamentally the same properties. In this study, we consider only the first instance of each FID in the dimensions $d = \{2, 3\}$, where we denote a problem instance as $p_i := (\text{FID}, d)$. This amounts to 48 distinct problem instances we endeavour to recreate. For each p_i , we run the Python implementation of CMA-ES called `pycma` with a budget of $200n$ where n is the dimensionality of the point cloud optimization problem and not the dimension d of the function we are trying to generate. We perform ten replications of which all are able to minimize Equation 1 similarly. For every problem p_i , we then select the point cloud that achieved the best objective value to provide our neural networks with the best possible point cloud to fit. All the results presented below, therefore, refer to the fits of the best point clouds.

Exemplary point clouds, where the approximation of the target ELA feature vector works sufficiently well and a counterexample, are given in Figure 2. The left-hand side depicts the case for BBOB function 3. Subfigure 2(a) depicts a random sample of the original BBOB function 3 whereas the adjacent subplot visualizes our point cloud after optimization. We can discern that the two landscapes are rotated, yet in essence they structurally remain the same. The right-hand side showcases an approximation attempt which did not terminate satisfactorily. While some aspects of the original BBOB function (FID 14) are present in our point cloud, the general landscape is far too quadratic and noisy. We believe that the observed phenomenon is attributable to a low resolution of the point cloud.

A full account for every problem instance p_i is given in Figure 3. As pointed out in Section 3.1, each ELA feature is normalized to fall into the interval $[0, 1]$. Hence, we can determine the maximum distance between any two ELA feature vectors $\in \mathbb{R}^8$ is $\sqrt{8}$. This helps to contextualize the reported results because these will be in the interval of $[0, \sqrt{8}]$. The vast majority of cells is colored white

which indicates the CMA-ES was able minimize the distance between the two ELA feature vectors up to a precision of 10^{-7} . These results are only tarnished by the functions 6, 9, 14 and 15 for $d = 2$, and for $d = 3$ this list is extended by functions 4, 12, 17 and 20. However, we cannot observe any statistical association which lets us divine the cause of these unsatisfactory instances. Yet, we have to put these values into context, meaning despite their perceptible coloring their values pale in comparison only and not when viewed in isolation. We deem them still sufficient to be used as training data for our surrogate model.

To judge the accuracy of these point clouds further, we create 100 samples for each problem instance p_i (using Latin hypercube sampling) and compute the respective ELA feature vectors. These are aggregated into a single vector $\bar{\mathbf{a}}_i$ via the arithmetic mean. While we make only use of a single dedicated NN architecture for all 48 problem instances, we train a NN individually for a single problem and each p_i receives its own NN model. In general, the training procedure in each individual attempt introduces a stochastic component. Hence, we trained five NNs for each problem instance where every NN is identical except their initial weights. Surprisingly, all five NNs interpolate between samples provided by the point cloud in similar fashion. Ultimately, this means that the resulting fitness landscape of our newly generated function is not subject to the training procedure of our NNs, i.e., the initialization of weights and biases. Similar to creating our target ELA vectors, we generate 100 ELA feature vectors $\mathbf{b}_i^{(j)}$ for each of the five NNs, therefore $j \in \{1, 2, \dots, 500\}$.

We report our results using the same metric of Equation 1. Meaning, the Euclidean distance between the target ELA vector $\bar{\mathbf{a}}_i$ of p_i and the 500 ELA feature vectors $\mathbf{b}_i^{(j)}$ of our constructed function. The distribution of Euclidean distances for each p_i can be found in Figure 4.

A cursory assessment already reveals that our devised approach works very well for the majority of the 48 considered problem instances. Nevertheless, the results for $d = 2$ are about an order of magnitude better than the results for $d = 3$. The cause for this relatively poor performance can partially be traced back to ill-suited point clouds found by CMA-ES. This is the case for functions 4, 6, 9, 12, 14, 17, and 20. Overall, we anticipate that premature termination of network training due to insufficient number of epochs is another contributing factor to this observation.

In the following, we provide a more in-depth view into the fitness landscape of a few selected functions. These are functions in which we excel but also functions which did not work out as well. In addition, we present exemplary uni-modal and multi-modal problem instances. These functions are visualized in Figure 5. We contrast the contour plot of the original BBOB problem instance to its corresponding approximation of our approach. This is accompanied by a parallel coordinate plot of the ELA features where the target ELA features are given in black and the values of our five surrogate models are colored.

The first row shows the Sphere function (FID 1), which is arguably one of the easiest optimization problems there is. This is also the case for the approximation of its landscape. The only notable difference between the two is that our function is not as artificially smooth as the BBOB function.

[†]Here, \hat{y} is the vector of predictions of the surrogate model consisting of predictions $\hat{y} = m(\mathbf{x})$ for each point $\mathbf{x} \in X$.

[‡]https://github.com/Reiyan/ela_nn_function_generation

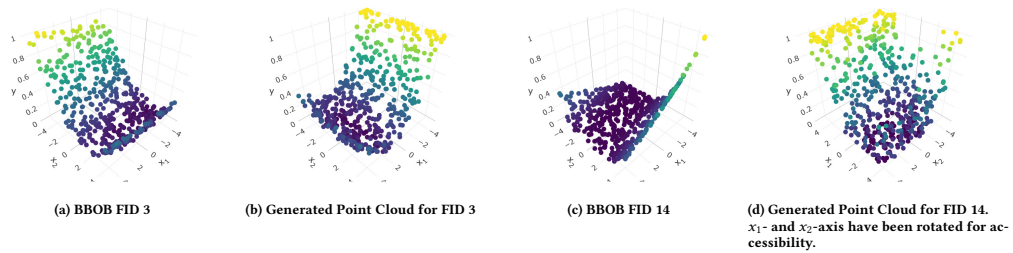


Figure 2: Generated point clouds optimized by CMA-ES in contrast to a sample generated from the respective BBOB function directly. Objective values of the respective functions have been scaled to $[0, 1]$ via min-max normalization.

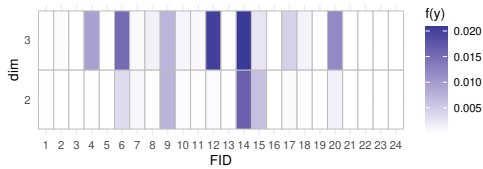


Figure 3: Heatmap of 48 distances in terms of $f(y)$ between a target ELA feature vector and the ELA feature vector calculated on the point cloud found by CMA-ES. The theoretical boundaries of these objective values are in the interval $[0, \sqrt{8}]$.

BBOB Function 2 displays a similar case. In terms of the captured landscape properties, we are able to emulate the separability of this problem well and also the quadratic structure as a whole. In addition, the generated landscape is rotated but we argue that this fundamentally no issue, since this does not change the characteristics of this problem.

Another function we deem worthy to report individual results on, is the rotated Rosenbrock function (FID 9). This function requires a solver to follow a long path to the global optimum while also accounting for changes in the search direction. Our approximation of it mimics this to a certain degree where we can observe a convergence into the corners of the search space. Yet, we cannot model the strength of this turn as strongly as it is present in the BBOB function.

We also want to critically discuss the inadequacies of our developed approach. This pertains to FID 14. This problem instance exhibits the worst performance in terms of our similarity measure s_i and also in terms of $f(y)$ when optimizing the point cloud. In this case, we can notice that overfitting our NNs leads to a more rugged landscape with drastically increasing and decreasing objective values of the landscape. This is, on the other hand, a property we require to model BBOB functions which are multi-modal and exhibit a mediocre to high conditioning of their respective landscape as will be shown in the following.

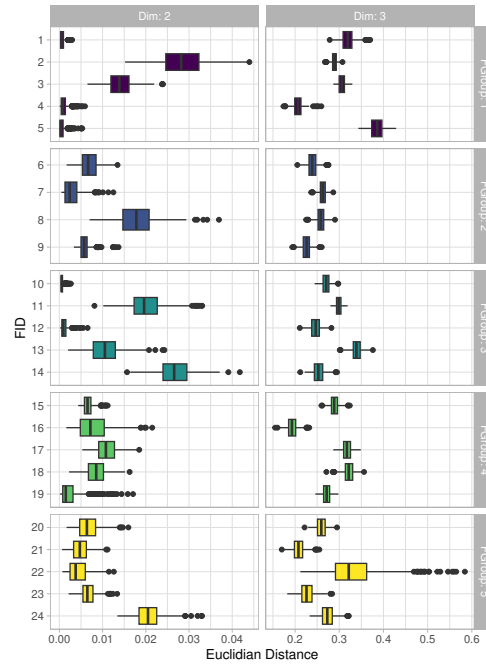


Figure 4: Boxplots of the Euclidean distances between the target ELA vector \bar{a}_i and each associated problem instance $b_i^{(j)}$.

Up until now, we only discussed the uni-modal functions with varying degrees of separability and conditioning. The last subplot of Figure 5 depicts the BBOB Function 21. This function is highly multi-modal with no global structure. The basin sizes of attraction

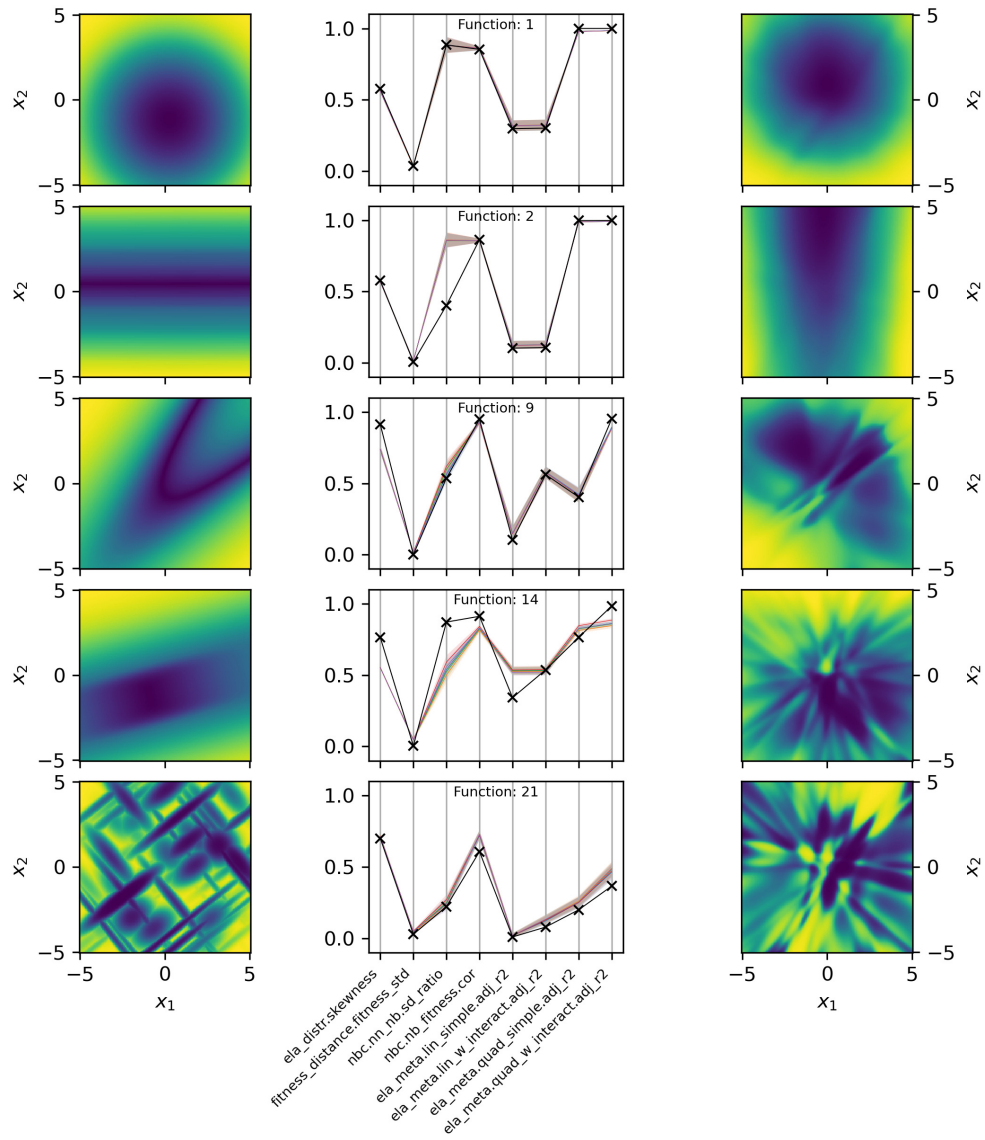


Figure 5: The left column shows contour plots of five selected BBOB functions. The affiliated eight dimensional target ELA vectors are represented by the black lines in the parallel plots of the middle column. The five colored lines show the mean ELA feature vector of the five trained neural networks. The matching colored area around the lines covers the tenth to ninetieth percentile of ELA feature values achieved during sampling. The right column represents the contour plots of one of the trained neural networks.

are aligned in random directions with different convex shapes. Our approximation is able to capture these properties in general. A meaningful difference is that our basins of attraction more often run parallel to the axes of the search space in contrast to the original BBOB function.

4.2 Algorithm Perspective

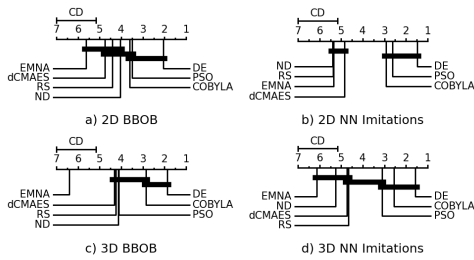


Figure 6: Performance rankings based on the performance of each optimizer with budget $10\,000 \cdot d$ for each dimension separately between BBOB and our imitated versions. Ties are identified by a non-parametric Nemenyi test and indicated by the horizontal lines. Lower rank is better.

It is imprudent to solely rely on the analysis of landscape properties to investigate the similarities between our imitated functions and their original counterparts. A different analysis avenue is provided by observing the behaviour of algorithms on each of these two sets and whether they exhibit a similar performance on the original as well as on the imitation. Hence, we conducted a comprehensive benchmark on BBOB making use of the Nevergrad framework [31]. Nevergrad provides a wide range of gradient-free optimization algorithms but what makes it most useful for our scenario is its interface for implementing and testing custom benchmark functions. As solvers we selected the following seven complementary algorithms:

- estimation of multivariate normal algorithm (EMNA) [20],
- Nelder-Mead algorithm (ND) [26],
- differential evolution (DE) [36],
- particle swarm optimization (PSO) [13],
- diagonal covariance matrix adaptation evolution strategy (dCMA-ES) [8],
- constrained optimization by linear approximation (Cobyla) [28],
- and random search (RS) [1].

Each of these algorithms is given a budget of $10\,000 \cdot d$ function evaluations and is applied 10 times on every single problem in the dimension two and three.

Figure 6 shows critical distance (CD) plots [2] for the BBOB and NN problems. We compute rankings based on the performance of each optimizer (median of 10 replications) on each of the BBOB and NN problems. A non-parametric Friedman test with Nemenyi

post-hoc test is performed to identify ties in the optimizer rankings, which are indicated by the horizontal lines. We rely on the implementation in the Python package autorank [9].

In Figure 6, we observe a comparable behavior in the algorithms examined. Specifically, the top three performing algorithms are DE, PSO, and COBYLA. In the $2d$ scenario, there is no statistically significant difference between these three. However, there are slight variations in the ranking of the remaining four algorithms. In our $2d$ imitations of BBOB, these variances are indistinguishable, whereas in the original BBOB suite, other algorithms sometimes tie with the three best performers in terms of their overall performance indicated by the horizontal line. Overall, our imitations of BBOB divide the seven algorithms more strongly into two groups. The $3d$ case provides a more convoluted case. Here, the best performing algorithms on BBOB are DE and COBYLA (which are statistically tied) whereas on our imitation suite this is extended by PSO. These permutations in rankings are to be expected and we think that overall also from an algorithmic perspective these two sets of functions are similar.

4.3 Generation of Novel Benchmark Functions

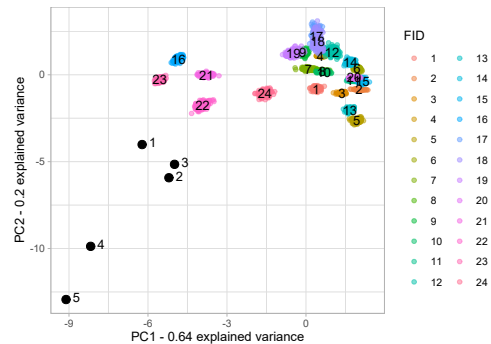


Figure 7: Projection of the ELA feature vectors of BBOB and newly generated functions (represented by a black dot) into a Cartesian plane via PCA.

While the imitation of existing benchmark functions allows us to dissect our approach in isolation and to identify possible shortcomings, we ultimately propose this method to create problem instances which are not covered by an existing benchmark suite. At first glance the creation of an ELA feature vector which is not represented by any problem instance of a given benchmark suite seems simple. This, however, is not the case since interactions between ELA features exist. For example, it is not possible to create an ELA feature vector which has R^2 of 1 for a linear model and simultaneously for a quadratic model (assuming non-zero model coefficients). Thereby, the identification of feasible ELA feature vectors - which are also different to ELA feature vectors of a benchmark - poses a separate research question which is not addressed in this work.

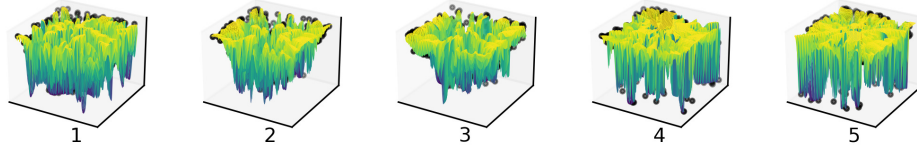


Figure 8: Landscape of the 5 newly created $2d$ Functions where the black points represent the point cloud optimized by the CMA-ES

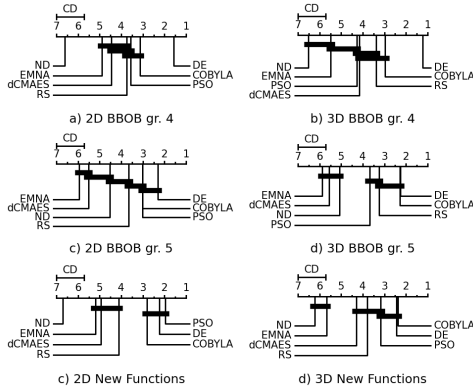


Figure 9: Performance rankings based on the performance of each optimizer with budget $10\,000 \cdot d$ for each dimension separately between BBOB function groups 4 and 5 in contrast to the 5 newly created functions. Ties are identified by a non-parametric Nemenyi test and indicated by the horizontal lines.

Rather, we handpicked five exemplary and feasible ELA feature vectors which have a considerable distance to the existing ELA feature vectors of BBOB to showcase the potential of our method. A projection of these five exemplary ELA feature vectors into a two dimensional plane (using PCA) is depicted in Figure 7.

The generation of the respective point clouds as well as the training of our NN surrogate models is identical to our previous undertaking. The optimization process to generate the point clouds terminates with objective values ranging from $2.37e-06$ to $1.26e-02$ and an average of $3.01e-03$ for $2d$ and $3d$ problems. We deem this sufficiently well approximated to build our surrogate models upon. Furthermore, we apply the same algorithm benchmark procedure outlined in Subsection 4.2.

Landscape Perspective. The landscape of the newly generated $2d$ functions is depicted in Figure 8. All five functions are highly multi-modal without any or very weak apparent global structure. The conditioning of these functions is also extremely high, i.e., small

changes in the decision space generally lead to drastic changes in the objective space. The first three landscapes tend to have the majority of their local and global optima located more in the center of the search space, whereas the latter two follow no observable pattern and have even highly competitive local optima right at the vertices of the box-constraints. Especially the fourth and fifth problem share commonalities with the Katsuura function (FID 23) of BBOB. However, our functions do not exhibit the same regular pattern for local optima which presumably makes it more difficult. This irregular pattern is uncommon for BBOB in general.

Algorithm Perspective. These aforementioned landscape properties make the newly generated functions most similar to the BBOB function group 4 or 5. Hence, we present the algorithm rankings in form of CD-plots for these groups specifically as the CD plots for the remaining function groups do not share any resemblance at all with the CD plots of the newly generated functions.

When assessing Figure 9, there seems to be no conclusive evidence, that our newly generated problems are substantially different in terms of algorithm rankings. ND performs the worst on the new generated functions which is also the case for BBOB group 4. The top performing algorithms are mainly PSO, DE, COBYLA and occasionally even RS. While we would have appreciated a more indicative finding from the algorithm perspective, we are satisfied with the structural differences in the constructed landscapes compared to BBOB.

5 CONCLUSION

In this work, we propose a novel approach for creating new benchmark problems for single-objective continuous black-box optimization. Our approach is mainly based on creating a so-called point cloud which exhibits the desired ELA features and thereby the desired fitness landscape properties. This point cloud is used to train a neural network which acts as a surrogate model and newly created benchmark function. Our approach solves the previously unaddressed issues of [3] by providing the capability to not only interpolate in the ELA feature space of benchmark functions but also extrapolate beyond the convex hull of this region. Furthermore, NNs are fully differentiable in theory (not only with respect to the weights and biases, but also with respect to the input variables) and we hypothesize that we can determine all optima comparably efficiently. This, however, needs to be fully explored. We evaluated our approach on a set of BBOB functions where the goal was to

recreate them as closely as possible. Our approach generates problem instances of identical nature. This is supported by a systematic benchmark study of different optimizers which exhibit an overall similar performance ranking on the original BBOB functions and their respective imitations.

In addition, we showcase that it is possible to create functions based on ELA feature vectors which are not represented by any BBOB function. The resulting functions offer interesting irregular structures and landscape property combinations which are not present in the current BBOB suite.

Yet, our devised approach is only the first foray into relatively uncharted territory. An inundating number of research opportunities still remains such as, e.g., improving the scalability to create functions of higher dimensionality. This can be done by employing more CPU-efficient algorithms such as the LM-CMA-ES [22]. Furthermore, a systematic evaluation of different sampling sizes and strategies may conclude that a smaller sample size still produces sufficient results. This would effectively reduce the dimensionality of our point cloud optimization and presumably the complexity of this optimization task.

While we evaluated a multitude of different NN architectures, we still believe that there is potential for improvement in this area. Testing different architectures or investigating the viability of smaller point clouds is only a small excerpt of possible refinements. We could also employ our approach to model real-world problems where a small sample of data exist. While this amount of data may be not sufficient enough to build a surrogate model on it alone, we can use it to calculate ELA features. Thereafter, our procedure can be applied to generate a surrogate function comparatively cheaply. While solving this surrogate function does not lead to the identification of the global optimum of the real-world problem, it can serve as a proxy to benchmark and design algorithms more tailored to real-world problems.

ACKNOWLEDGMENTS

Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8).

REFERENCES

- [1] R. L. Anderson. 1953. Recent Advances in Finding Best Operating Conditions. *J. Amer. Statist. Assoc.* 48, 264 (1953), 789–798. <https://doi.org/10.1080/01621459.1953.10501200>
- [2] Janez Demsar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.
- [3] Konstantin Dietrich and Olaf Mersmann. 2022. Increasing the Diversity of Benchmark Function Sets Through Affine Recombination. In *Parallel Problem Solving from Nature – PPSN XVII: 17th International Conference, PPSN 2022, Dortmund, Germany, September 10–14, 2022, Proceedings, Part I* (Dortmund, Germany). Springer-Verlag, Berlin, Heidelberg, 590–602. https://doi.org/10.1007/978-3-031-14714-2_41
- [4] M. Gallagher and Bo Yuan. 2006. A general-purpose tunable landscape generator. *IEEE Transactions on Evolutionary Computation* 10 (2006), 590–603. Issue 5. <https://doi.org/10.1109/TEVC.2005.863>
- [5] L. Grinsztajn, E. Oyallon, and G. Varoquaux. 2022. Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- [6] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Research Report RR-6829. INRIA. <https://hal.inria.fr/inria-00362633>
- [7] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. 2003. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evol. Comput.* 11, 1 (mar 2003), 1–18. <https://doi.org/10.1162/10636560321828970>
- [8] N. Hansen and A. Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9 (6 2001), 159–195. Issue 2. <https://doi.org/10.1162/106365601750190398>
- [9] Steffen Herbold. 2020. Autorank: A Python package for automated ranking of classifiers. *Journal of Open Source Software* 5, 48 (2020), 2173. <https://doi.org/10.21105/joss.02173>
- [10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer Feed-forward Networks Are Universal Approximators. *Neural Networks* 2, 5 (1989), 359–366.
- [11] Anja Janković and Carola Doerr. 2019. Adaptive Landscape Analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Prague, Czech Republic) (GECCO '19). Association for Computing Machinery, New York, NY, USA, 2032–2035. <https://doi.org/10.1145/3319619.3326905>
- [12] Terry Jones and Stephanie Forrest. 1995. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 184–192.
- [13] J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. 4 (1995), 1942–1948 vol.4. <https://doi.org/10.1109/ICNN.1995.488968>
- [14] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation* 27, 1 (2019), 3 – 45. https://doi.org/10.1162/evco_a_00242
- [15] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2015. Detecting Funnel Structures by Means of Exploratory Landscape Analysis. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain) (GECCO '15). Association for Computing Machinery, New York, NY, USA, 265–272. <https://doi.org/10.1145/2739480.2754642>
- [16] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2016. Low-Budget Exploratory Landscape Analysis on Multiple Peaks Models. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (Denver, Colorado, USA) (GECCO '16). Association for Computing Machinery, New York, NY, USA, 229–236. <https://doi.org/10.1145/2908812.2908845>
- [17] Pascal Kerschke and Heike Trautmann. 2019. Automated Algorithm Selection on Continuous Black-Box Problems By Combining Exploratory Landscape Analysis and Machine Learning. *Evolutionary Computation* 27, 1 (2019), 99 – 127. https://doi.org/10.1162/evco_a_00236
- [18] Ana Kostovska, Diederick Vermetten, Sašo Džeroski, Carola Doerr, Peter Korosec, and Tome Eftimov. 2022. The Importance of Landscape Features for Performance Prediction of Modular CMA-ES Variants. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (Boston, MA, USA), Jonathan E. Fieldsend and Markus Wagner (Eds.). ACM, 648 – 656. <https://doi.org/10.1145/3512290.3528832>
- [19] Ryan Dieter Lang and Andries Petrus Engelbrecht. 2021. An Exploratory Landscape Analysis-Based Benchmark Suite. *Algorithms* 14, 3 (2021), 78.
- [20] Pedro Larrañaga and Jose A Lazano. 2002. *Estimation of Distribution Algorithms*. Vol. 2. Springer Science & Business Media. <https://doi.org/10.1007/978-1-4615-1539-5>
- [21] Fu Xing Long, Bas van Stein, Moritz Frenzel, Peter Krause, Markus Gitterle, and Thomas Bäck. 2022. Learning the Characteristics of Engineering Optimization Problems with Applications in Automotive Crash (GECCO '22). Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3512290.3528712>
- [22] Ilya Loshchilov. 2014. A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (Vancouver, BC, Canada) (GECCO '14). Association for Computing Machinery, New York, NY, USA, 397–404. <https://doi.org/10.1145/2576768.2598294>
- [23] Katherine Mary Malan and Andries Petrus Engelbrecht. 2009. Quantifying Ruggedness of Continuous Landscapes Using Entropy. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1440 – 1447. <https://doi.org/10.1109/CEC.2009.4983112>
- [24] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory Landscape Analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Dublin, Ireland) (GECCO '11). Association for Computing Machinery, New York, NY, USA, 829–836. <https://doi.org/10.1145/2001576.2001690>
- [25] Mario Andrés Muñoz and Kate Smith-Miles. 2020. Generating New Space-Filling Test Instances for Continuous Black-Box Optimization. *Evolutionary Computation* 28 (2020), 379–404. Issue 3. https://doi.org/10.1162/evco_a_00262
- [26] J. A. Nelder and R. Mead. 1965. A Simplex Method for Function Minimization. *Comput. J.* 7 (1 1965), 308–313. Issue 4. <https://doi.org/10.1093/COMJNL/7.4.308>
- [27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan

- Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [28] M. J. D. Powell. 1994. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*. Springer Netherlands, Dordrecht, 51–67. https://doi.org/10.1007/978-94-015-8330-5_4
- [29] Raphael Patrick Prager, Moritz Vinzent Seiler, Heike Trautmann, and Pascal Kerschke. 2022. Automated Algorithm Selection in Single-Objective Continuous Optimization: A Comparative Study of Deep Learning and Landscape Analysis Methods. In *Proceedings of the 17th International Conference on Parallel Problem Solving from Nature (PPSN XVII)* (Dortmund, Germany), Günter Rudolph, Anna V. Kononova, Hernán E. Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tušar (Eds.). Springer, 3 – 17. https://doi.org/10.1007/978-3-031-14714-2_1
- [30] Raphael Patrick Prager and Heike Trautmann. 2023. Nullifying the Inherent Bias of Non-Invariant Exploratory Landscape Analysis Features. In *Applications of Evolutionary Computation*, João Correia, Stephen Smith, and Raneem Qaddoura (Eds.). Springer International Publishing, Cham.
- [31] J. Rapin and O. Teytaud. 2018. Nevergrad - A Gradient-Free Optimization Platform. <https://GitHub.com/FacebookResearch/Nevergrad>.
- [32] Quentin Renau, Carola Doerr, Johann Dreo, and Benjamin Doerr. 2020. Exploratory Landscape Analysis is Strongly Sensitive to the Sampling Strategy. *Proceedings of the 16th International Conference on Parallel Problem Solving from Nature (PPSN 2020)* 12270 LNCS (2020), 139–153. https://doi.org/10.1007/978-3-030-58115-2_10
- [33] Quentin Renau, Johann Dreo, Carola Doerr, and Benjamin Doerr. 2021. Towards Explainable Exploratory Landscape Analysis: Extreme Feature Selection for Classifying BBOB Functions. *Proceedings of the 24th International Conference, EvoApplications 2021* 12694 LNCS (2021), 17–33. <https://doi.org/10.48550/arxiv.2102.00736>
- [34] Lennart Schneider, Lennart Schäpermeier, Raphael Patrick Prager, Bernd Bischl, Heike Trautmann, and Pascal Kerschke. 2022. HPO × ELA: Investigating Hyperparameter Optimization Landscapes by Means of Exploratory Landscape Analysis. In *Parallel Problem Solving from Nature – PPSN XVII*, Günter Rudolph, Anna V. Kononova, Hernán Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tušar (Eds.). Springer International Publishing, Cham, 575–589.
- [35] Moritz Vinzent Seiler, Raphael Patrick Prager, Pascal Kerschke, and Heike Trautmann. 2022. A Collection of Deep Learning-Based Feature-Free Approaches for Characterizing Single-Objective Continuous Fitness Landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference (Boston, Massachusetts) (GECCO '22)*, Association for Computing Machinery, New York, NY, USA, 657–665. <https://doi.org/10.1145/3512290.3528834>
- [36] Rainer Storn and Kenneth Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359. <https://doi.org/10.1023/A:1008202821328>
- [37] Ke Tang, Xin Yao, Ponnuthurai Nagaratnam Suganthan, Cara MacNish, Ying-Ping Chen, Chih-Ming Chen, and Zhenyu Yang. 2007. Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. *Nature inspired computation and applications laboratory, USTC, China* 24 (2007), 1–18.
- [38] Konstantinos Varelas. 2019. Benchmarking Large Scale Variants of CMA-ES and L-BFGS-B on the Bbob-Largescale Testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (Prague, Czech Republic) (GECCO '19)*, Association for Computing Machinery, New York, NY, USA, 1937–1945. <https://doi.org/10.1145/3319619.3326893>

Part III.

Conclusion

6. Concluding Remarks

In this thesis, we have advanced HPO along three key dimensions: its foundations, multi-objective HPO and quality diversity HPO, and benchmarking. We have demonstrated both theoretically and empirically that reshuffling resampling splits during HPO can improve generalization performance. Unlike the standard approach of keeping train and validation splits fixed, reshuffling introduces variability that can be beneficial for optimization. We further analyzed the landscape properties of HPO problems, comparing them to traditional synthetic black-box problems, and found that HPO landscapes often exhibit low multimodality and wide plateaus. To efficiently optimize both model performance and interpretability quantified through feature sparsity, interaction sparsity, and the sparsity of non-monotone features, we developed a novel multi-objective HPO algorithm. Furthermore, we explored how the black-box optimization field of quality diversity optimization can be used for NAS by simultaneously optimizing model performance and resource efficiency under multiple constraints in a single optimization run. For efficient, reproducible and scalable HPO benchmarking, we introduced YAHPO Gym, a large-scale surrogate benchmarking suite that sets a new standard for efficient benchmarking in HPO. To demonstrate its utility, we used YAHPO Gym to automatically configure a multi-fidelity HPO algorithm, aligning with the principles of programming by optimization. Through ablation studies, we found that many simple algorithmic components often perform as well as, or even outperform more complicated alternatives. Additionally, we introduced quality diversity benchmark problems derived from HPO, bridging a gap between two research communities. Finally, we developed a systematic approach to generating black-box functions that exhibit desired optimization landscape properties.

At the same time, it is important to take a step back and critically reflect on both our contributions and recent developments in HPO and AutoML. With respect to benchmarking, the fields of HPO, NAS, and AutoML are in a substantially better state compared to when this thesis was first started. The emergence of benchmarking tools such as HPOBench (Eggenesperger et al., 2021), HPO-B (Pineda Arango et al., 2021), YAHPO Gym, Syne Tune (Salinas et al., 2022), and various NAS benchmark suites (Ying et al., 2019; Dong and Yang, 2020; Li et al., 2021; Zela et al., 2022; Bansal et al., 2022; Mehta et al., 2022), along with the AMLB benchmark for evaluating full AutoML pipelines (Gijsbers et al., 2024), has allowed for more accessible, reproducible, and efficient benchmarking. However, despite these advancements, researchers still tend to cherry-pick benchmarks from different suites, define their own evaluation protocols, and set arbitrary optimization budgets. This shows that efforts to establish standardized benchmarking suites and protocols have not been universally adopted by the research community. Interestingly, while past concerns focused on the need for more realistic, diverse and practically relevant HPO and NAS benchmarks, the opposite may now be emerging. The sheer number of available benchmarking suites and benchmarks has grown so large that it is becoming impractical to exhaustively evaluate algorithms across all of them. This raises the following question: How can we define a representative and relevant subset of benchmarks that remain comprehensive yet small enough? Foundational work in this direction in the context of traditional black-box benchmarking, such as

instance hardness analysis (Smith-Miles, 2025), can provide valuable insights for the HPO community. Ultimately, the selection of benchmark instances by the research community not only reflects but also shapes the direction of algorithmic development. Recent trends in both HPO and NAS have shown a growing focus on improving validation errors, often relying on tabular or surrogate benchmarks that fail to capture the stochastic nature of generalization error estimation. While this approach is reasonable for comparing optimization algorithms (Feurer and Hutter, 2019), it overlooks the fundamental goal of HPO and NAS: to identify models that generalize well to unseen data, i.e., to an outer test set. Additionally, treating the objective function as purely deterministic can lead to misleading conclusions about algorithmic progress that may not translate to real-world improvements in HPO. For example, in our empirical investigation of resamplings and reshuffling resampling splits, we also demonstrated that the strongest improvements in generalization performance, particularly on small to medium-sized datasets, stem from obtaining a more precise resampling estimate of the generalization error using repeated cross-validation instead of a simple holdout. Large-scale surrogate or tabular benchmarking suites typically do not allow for such variations in evaluation protocols, as they primarily provide deterministic validation performance estimates obtained via a single resampling. Future HPO benchmarks should explicitly aim to bridge this gap, ensuring benchmarks to better reflect the complexities of real-world HPO and NAS rather than remaining narrowly focused on validation error alone.

With respect to HPO in the context of multiple objectives, we focused on multi-objective optimization of performance and interpretability, as well as quality diversity optimization for well-performing architectures that satisfy different hardware constraints. Based on these contributions and the current state of the related literature, we conclude that HPO and AutoML with multiple objectives remain rather niche topics. On the one hand, numerous works emphasize the need for AutoML solutions to be more customizable and adaptive to user-specific requirements, including objectives such as model simplicity or interpretability (Bischl et al., 2023; Karl et al., 2023; Baratchi et al., 2024). The contributions in this thesis can be seen as initial steps toward the broader goal of multi-objective and more flexible AutoML. On the other hand, to the best of our knowledge, no existing AutoML tool currently supports multi-objective optimization as an integral part of its pipeline. Furthermore, from a usability perspective, it remains unclear whether multi-objective HPO or AutoML would provide a clear benefit, as users may struggle with selecting a solution from the Pareto set, especially when faced with more than two competing objectives, where trade-offs become complex to communicate and interpret. In this aspect, exploring the potential of preferential optimization techniques (González et al., 2017), which rely on direct user feedback through pairwise comparisons of solutions, could be a promising direction for improving usability and decision support in multi-objective HPO.

Looking at a deeper understanding of HPO and its optimization characteristics we note that follow-up research emerged, which has explored the landscape properties of HPO and NAS (Huang and Li, 2024), as well as reinforcement learning problems (Mohan et al., 2023). In line with the findings presented in this thesis, these studies show that many HPO and NAS landscapes are nearly unimodal, with few local optima, whereas reinforcement learning problems might be more challenging. Interestingly, Huang and Li (2024) report that the landscape of the validation error does not always generalize to the landscape of the test error on an outer test set. This further supports the argument that the HPO and AutoML community should shift focus away from merely achieving state-of-the-art validation errors on benchmarks and instead prioritize developing robust solutions that generalize well. Additionally, we must critically reflect on the “difficulty” of many HPO problems. Besides our investigation of the landscape properties of HPO problems,

ECDF plots from benchmarks such as HPOBench and YAHPO Gym suggest that often many well-performing configurations exist, and sophisticated HPO algorithms can efficiently identify them even under tight computational budgets. In this sense, vanilla HPO may have reached its asymptote of effectiveness, or, worded more optimistically, fulfilled its original goal. However, a notable gap remains between scientific and algorithmic advancements in HPO and their adoption in practice. For instance, [Sun et al. \(2023\)](#), based on semi-structured interviews with AutoML users, highlight that a lack of customization remains a key limitation of current AutoML systems. They emphasize the need for integrating domain knowledge into AutoML workflows to enhance usability and effectiveness. In the context of HPO, the inclusion of prior knowledge and user preferences are gaining attention, with notable contributions from [Hvarfner et al. \(2022b\)](#) and [Mallik et al. \(2023\)](#). Related, [Kannengießer et al. \(2025\)](#) report that many ML practitioners still rely on manual tuning, which they find highly effective for enhancing model understanding while requiring minimal effort in complex settings. In combination, these insights show the need for AutoML solutions that re-center the user in the optimization process, enabling more flexible, domain-adaptive, and interpretable model selection. For a more detailed discussion of this topic, we refer to [Lindauer et al. \(2024\)](#).

Finally, looking at the position of HPO in the broader field of ML, we have to acknowledge that HPO might have reached the peak of its popularity and potentially also relevance. Figure 6.1 visualizes the normalized interest in the search term “hyperparameter optimization” from January 2016 to March 2025 according to Google Trends. We observe a steady rise in interest over the years, peaking around 2022, followed by a slight decline and subsequent stabilization. This shows that, while interest in HPO has experienced significant growth, it no longer exhibits a strong upward trajectory. However, rather than indicating a decline, this stabilization likely shows that HPO has matured as a field within ML. Taking into perspective recent advancements in ML and the strong performance of large pre-trained foundation models, this might not be surprising. In the field of natural language processing the dominant paradigm has long shifted to the usage of large retained language models ([Vaswani et al., 2017](#); [Radford et al., 2018](#); [Devlin et al., 2019](#); [Brown et al., 2020](#); [Ding et al., 2023](#)). Similarly, in the field of computer vision, there is a trend to develop large all-purpose models ([Kirillov et al., 2023](#); [Oquab et al., 2024](#)) that can be fine-tuned ([Xin et al., 2024](#)) to a downstream task at hand. This naturally results in less relevance of the fields of HPO and NAS, as models are no longer trained from scratch except during the extremely costly pre-training step,

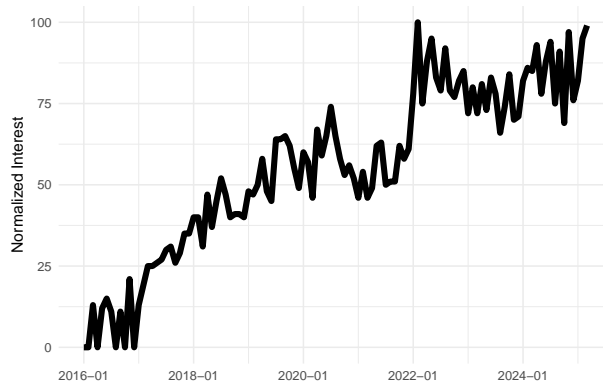


Figure 6.1.: Normalized interest in the search term “hyperparameter optimization” from January 2016 to March 2025 according to Google Trends.

where making use of HPO and AutoML techniques remains a challenge (Tornede et al., 2024). A similar trend is emerging in tabular supervised learning, where pre-training strategies have demonstrated strong performance. The introduction of TabPFN (Hollmann et al., 2023, 2025), a transformer-based model for tabular data (Müller et al., 2022) pre-trained on vast amounts of synthetic data, has shown exceptional results, particularly for small datasets. In such cases, HPO and NAS are no longer strictly necessary to achieve state-of-the-art performance, at least when good predictive performance is the primary objective.

Despite these shifts in the general ML landscape, HPO remains a relevant tool for developing well-performing and customized models, especially in areas where large-scale pretraining is not feasible or resource constraints call for efficient model selection or custom user metrics need to be addressed. The methods, insights and benchmarks introduced in this thesis have contributed to a more principled, reproducible, and efficient HPO pipeline, ensuring that future research can build upon a solid foundation. Moreover, while HPO may have reached maturity as a field, its integration with broader AutoML, interpretability and user-centered optimization paradigms presents exciting new directions. The growing need for customizable and domain-adaptive solutions shows the importance of multi-objective HPO, potentially preferential optimization and human-in-the-loop approaches, all of which remain largely underexplored. Rather than signaling an end, the stabilization of interest in HPO suggests a transition toward practical adoption and refinement. Ideally, the focus of HPO will shift to real-world impact. As ML continues to evolve, HPO will likely remain a key component in optimizing new learning algorithms and architectures, guiding efficient model deployment, and ensuring that predictive models are not only well-performing but also interpretable, efficient and accessible. With this perspective, we conclude this thesis with optimism, hoping that the advancements made here will serve as stepping stones for the next generation of HPO research.

Part IV.

References, Excluded Publications and Contributing Publications

References

- T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.
- D. Asenjo, J. D. Stevenson, D. J. Wales, and D. Frenkel. Visualizing basins of attraction for different minimization algorithms. *The Journal of Physical Chemistry B*, 117(42):12717–12723, 2013.
- A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1769–1776, 2005.
- M. Austern and W. Zhou. Asymptotics of cross-validation, 2020. URL <https://arxiv.org/abs/2001.11111>.
- N. Awad, N. Mallik, and F. Hutter. DEHB: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2147–2153, 2021.
- T. Bai, Y. Li, Y. Shen, X. Zhang, W. Zhang, and B. Cui. Transfer learning for Bayesian optimization: A survey, 2023. URL <https://arxiv.org/abs/2302.05927>.
- L. Baier, F. Jöhren, and S. Seebacher. Challenges in the deployment and operation of machine learning in practice. In *ECIS*, volume 1, 2019.
- B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *The Fifth International Conference on Learning Representations*, 2017.
- M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A framework for efficient monte-carlo Bayesian optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21524–21538, 2020.
- A. Bansal, D. Stoll, M. Janowski, A. Zela, and F. Hutter. JAHS-Bench-201: A foundation for research on joint architecture and hyperparameter search. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 38788–38802, 2022.
- M. Baratchi, C. Wang, S. Limmer, J. N. van Rijn, H. H. Hoos, T. Bäck, and M. Olhofer. Automated machine learning: Past, present and future. *Artificial Intelligence Review*, 57(5):122, 2024.
- T. Bartz-Beielstein, C. Doerr, D. van den Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. López-Ibañez, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise. Benchmarking in optimization: Best practice and open issues, 2020. URL <https://arxiv.org/abs/2007.03488>.

- S. Bates, T. Hastie, and R. Tibshirani. Cross-validation: What does it estimate and how well does it do it? *Journal of the American Statistical Association*, 119(546):1434–1445, 2024.
- S. Batra, B. Tjanaka, M. C. Fontaine, A. Petrenko, S. Nikolaidis, and G. Sukhatme. Proximal policy gradient arborescence for quality diversity reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.
- P. Bayle, A. Bayle, L. Janson, and L. Mackey. Cross-validation confidence intervals for test error. In H. H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16339–16350, 2020.
- S. Belakaria, A. Deshwal, and J. R. Doppa. Max-value entropy search for multi-objective Bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019.
- S. Belakaria, A. Deshwal, and J. R. Doppa. Multi-fidelity multi-objective Bayesian optimization: An output space entropy search approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10035–10043, 2020.
- Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16, 2003.
- H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang. Hardware-aware neural architecture search: Survey and taxonomy. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4322–4329, 2021.
- J. Bennett and S. Lanning. The netflix prize. In *Proceedings of the KDD Cup and Workshop*, 2007.
- J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24, 2011.
- J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 115–123, 2013.
- A. Bibal and B. Frénay. Interpretability of machine learning models and representations: An introduction. In *24th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 77–82, 2016.
- Bilal, M. Pant, H. Zaheer, L. Garcia-Hernandez, and A. Abraham. Differential evolution: A review of more than two decades of research. *Engineering Applications of Artificial Intelligence*, 90:103479, 2020.
- B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pages 313–320, 2012a.

References

- B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary Computation*, 20(2):249–275, 2012b.
- B. Bischl, S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs. MOI-MBO: Multiobjective infill for parallel model-based optimization. In P. Pardalos, M. Resende, C. Vogiatzis, and J. Walteros, editors, *Learning and Intelligent Optimization*, pages 173–186, 2014.
- B. Bischl, G. Casalicchio, M. Feurer, P. Gijsbers, F. Hutter, M. Lang, R. Gomes Mantovani, J. N. van Rijn, and J. Vanschoren. OpenML benchmarking suites. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, and M. Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.
- S. Borra and A. Di Ciaccio. Measuring the prediction error. A comparison of cross-validation, bootstrap and covariance penalty methods. *Comput. Stat. Data Anal.*, 54(12):2976–2989, 2010.
- L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning, 2018. URL <https://arxiv.org/abs/1606.04838>.
- A.-L. Boulesteix, C. Strobl, T. Augustin, and M. Daumer. Evaluating microarray-based classifiers: An overview. *Cancer Informatics*, 6:CIN.S408, 2008.
- X. Bouthillier, P. Delaunay, M. Bronzi, A. Trofimov, B. Nichyporuk, J. Szeto, N. Mohammadi Sepahvand, E. Raff, K. Madan, V. Voleti, S. Ebrahimi Kahou, V. Michalski, T. Arbel, C. Pal, G. Varoquaux, and P. Vincent. Accounting for variance in machine learning benchmarks. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 747–769, 2021.
- F. H. Branin. Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, 16(5):504–522, 1972.
- L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.
- L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman and Hall, New York, 1984.
- T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.
- S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

- A. D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(88):2879–2904, 2011.
- A. Butterfield, G. E. Ngondi, and A. Kerr. *A Dictionary of Computer Science*. Oxford University Press, Oxford, 7 edition, 2016.
- R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.
- H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *The Seventh International Conference on Learning Representations*, 2019.
- G. C. Cawley and N. L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(70):2079–2107, 2010.
- T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.
- T. Chen, E. Fox, and C. Guestrin. Stochastic gradient hamiltonian Monte Carlo. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 1683–1691, 2014.
- C. Chevalier and D. Ginsbourger. Fast computation of the multi-points expected improvement with applications in batch selection. In G. Nicosia and P. Pardalos, editors, *Learning and Intelligent Optimization*, pages 59–69, 2013.
- K. W. Church. Emerging trends: A tribute to Charles Wayne. *Natural Language Engineering*, 24(1):155–160, 2018.
- S. Corbett-Davies, J. D. Gaebler, H. Nilforoshan, R. Shroff, and S. Goel. The measure and mismeasure of fairness. *Journal of Machine Learning Research*, 24(312):1–117, 2023.
- C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- A. I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. R. Griffiths, A. M. Maraval, H. Jianye, J. Wang, J. Peters, and H. Bou-Ammar. HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349, 2022.
- A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.
- N. De Freitas, A. J. Smola, and M. Zoghi. Exponential regret bounds for Gaussian process bandits with deterministic observations. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, pages 955–962, 2012.
- K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

References

- T. Desautels, A. Krause, and J. W. Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research*, 15(119):4053–4103, 2014.
- J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.
- N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, H.-T. Zheng, J. Chen, Y. Liu, J. Tang, J. Li, and M. Sun. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.
- T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 3460–3468, 2015.
- X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *The Eighth International Conference on Learning Representations*, 2020.
- A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *The Ninth International Conference on Learning Representations*, 2021.
- R. Egelé, I. Guyon, V. Vishwanath, and P. Balaprakash. Asynchronous decentralized Bayesian optimization for large scale hyperparameter optimization. In *2023 IEEE 19th International Conference on e-Science (e-Science)*, pages 1–10, 2023.
- K. Eggenberger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, volume 10, pages 1–5, 2013.
- K. Eggenberger, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), 2015.
- K. Eggenberger, M. Lindauer, and F. Hutter. Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 64:861–893, 2019.
- K. Eggenberger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019a.
- T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via Lamarckian evolution. In *The Seventh International Conference on Learning Representations*, 2019b.

- M. T. M. Emmerich and A. H. Deutz. A tutorial on multiobjective optimization: Fundamentals and evolutionary methods. *Natural Computing*, 17:585–609, 2018.
- M. T. M. Emmerich, K. C. Giannakoglou, and B. Naujoks. Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, 2006.
- M. T. M. Emmerich, A. H. Deutz, and J. W. Klinkenberg. Hypervolume-based expected improvement: Monotonicity properties and exact computation. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2147–2154, 2011.
- M. T. M. Emmerich, K. Yang, A. Deutz, H. Wang, and C. M. Fonseca. A multicriteria generalization of Bayesian global optimization. *Advances in Stochastic and Deterministic Global Optimization*, pages 229–242, 2016.
- S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1437–1446, 2018.
- C. Fawcett and H. H. Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22:431–458, 2016.
- M. Feurer and F. Hutter. Hyperparameter optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 3–33. Springer International Publishing, Cham, 2019.
- M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, 2015a.
- M. Feurer, J. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015b.
- M. C. Fontaine and S. Nikolaidis. Differentiable quality diversity. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 10040–10052, 2021.
- M. C. Fontaine, R. Liu, A. Khalifa, J. Modi, J. Togelius, A. K. Hoover, and S. Nikolaidis. Illuminating mario scenes in the latent space of a generative adversarial network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5922–5930, 2021.
- L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1165–1173, 2017.
- L. Franceschi, P. Frasconi, S. Salzo, R. Grazi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1568–1577, 2018.
- L. Franceschi, M. Donini, V. Perrone, A. Klein, C. Archambeau, M. Seeger, M. Pontil, and P. Frasconi. Hyperparameter optimization in machine learning, 2024. URL <https://arxiv.org/abs/2410.22854>.

References

- P. Frazier. A tutorial on Bayesian optimization, 2018. URL <https://arxiv.org/abs/1807.02811>.
- P. Frazier, W. Powell, and S. Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS Journal on Computing*, 21(4):599–613, 2009.
- A. A. Freitas. Automated machine learning for studying the trade-off between predictive accuracy and interpretability. In A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, editors, *Machine Learning and Knowledge Extraction*, pages 48–66, 2019.
- A. Gaier, A. Asteroth, and J.-B. Mouret. Data-efficient exploration, optimization, and modeling of diverse designs through surrogate-assisted illumination. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 99–106, 2017.
- J. R. Gardner, M. Kusner, X. Zhixiang, K. Weinberger, and J. Cunningham. Bayesian optimization with inequality constraints. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 937–945, 2014.
- R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.
- E. C. Garrido-Merchán and D. Hernández-Lobato. Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. *Neurocomputing*, 380:20–35, 2020.
- M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 250–259, 2014.
- P. Gijbbers, M. L. P. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. AMLB: An AutoML benchmark. *Journal of Machine Learning Research*, 25(101):1–65, 2024.
- D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization. In Y. Tenne and C.-K. Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, pages 131–162, 2010.
- D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495, 2017.
- J. Gonzalez, Z. Dai, P. Hennig, and N. D. Lawrence. Batch Bayesian optimization via local penalization. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pages 648–657, 2016.
- J. González, Z. Dai, A. Damianou, and N. D. Lawrence. Preferential Bayesian optimization. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1282–1291, 2017.
- I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- R. B. Gramacy and H. K. H. Lee. Optimization under unknown constraints. *Bayesian Statistics*, 9:229–256, 2011.

- I. Guyon, A. Saffari, G. Dror, and G. Cawley. Model selection: Beyond the Bayesian/frequentist divide. *Journal of Machine Learning Research*, 11(3):61–87, 2010.
- I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 ChaLearn AutoML challenge. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.
- L. Hancox-Li. Robustness in machine learning explanations: Does it matter? In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 640–647, 2020.
- N. Hansen. The CMA Evolution Strategy: A tutorial, 2023. URL <https://arxiv.org/abs/1604.00772>.
- N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, 1996.
- N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Research Report RR-6829, INRIA, 2009. URL <https://inria.hal.science/inria-00362633>.
- N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36:114–144, 2021.
- T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2 edition, 2009.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(57):1809–1837, 2012.
- D. Hernández-Lobato, J. M. Hernández-Lobato, A. Shah, and R. P. Adams. Predictive entropy search for multi-objective Bayesian optimization. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1492–1501, 2016a.
- J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, 2014.
- J. M. Hernández-Lobato, M. A. Gelbart, M. Hoffman, R. P. Adams, and Z. Ghahramani. Predictive entropy search for Bayesian optimization with unknown constraints. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1699–1707, 2015.
- J. M. Hernández-Lobato, M. A. Gelbart, R. P. Adams, M. W. Hoffman, and Z. Ghahramani. A general framework for constrained Bayesian optimization using information-based search. *Journal of Machine Learning Research*, 17(160):1–53, 2016b.

References

- M. Herrmann, F. J. D. Lange, K. Eggenesperger, G. Casalicchio, M. Wever, M. Feurer, D. Rügamer, E. Hüllermeier, A.-L. Boulesteix, and B. Bischl. Position: Why we must rethink empirical research in machine learning. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 18228–18247, 2024.
- N. Hollmann, S. Müller, K. Eggenesperger, and F. Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023.
- N. Hollmann, S. Müller, L. Purucker, A. Krishnakumar, M. Körfer, S. B. Hoo, R. T. Schirrmeister, and F. Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.
- G. Hooker. Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3):709–732, 2007.
- H. H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012a.
- H. H. Hoos. Automated algorithm configuration and parameter tuning. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, pages 37–71. Springer, Berlin, Heidelberg, 2012b.
- D. Horn, T. Wagner, D. Biermann, C. Weihs, and B. Bischl. Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 64–78, 2015.
- A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.
- M. Huang and K. Li. On the hyperparameter loss landscapes of machine learning models: An exploratory study, 2024. URL <https://arxiv.org/abs/2311.14014>.
- F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 1152–1157, 2007.
- F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.
- F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, 2011.
- F. Hutter, H. H. Hoos, and K. Leyton-Brown. Parallel algorithm configuration. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, pages 55–70, 2012.
- F. Hutter, H. H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 754–762, 2014.
- F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing, Cham, 2019.

- C. Hvarfner, F. Hutter, and L. Nardi. Joint entropy search for maximally-informed Bayesian optimization. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 11494–11506, 2022a.
- C. Hvarfner, D. Stoll, A. Souza, M. Lindauer, F. Hutter, and L. Nardi. π BO: Augmenting acquisition functions with user beliefs for Bayesian optimization. In *The Tenth International Conference on Learning Representations*, 2022b.
- M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks, 2017. URL <https://arxiv.org/abs/1711.09846>.
- K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pages 240–248, 2016.
- R. Jenatton, C. Archambeau, J. González, and M. Seeger. Bayesian optimization with tree-structured dependencies. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1655–1664, 2017.
- S. Jeong and S. Obayashi. Efficient global optimization (EGO) for multi-objective problem and data mining. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2138–2145, 2005.
- D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79:157–181, 1993.
- D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.
- A. Kadra, M. Janowski, M. Wistuba, and J. Grabocka. Scaling laws for hyperparameter optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 47527–47553, 2023.
- K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity Bayesian optimisation with continuous approximations. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1799–1808, 2017.
- K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.
- K. Kandasamy, K. R. Vysyaraju, W. Neiswanger, B. Paria, C. R. Collins, J. Schneider, B. Póczos, and E. P. Xing. Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with Dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.
- N. Kannengießner, N. Hasebrook, F. Morsbach, M.-A. Zöllner, J. Franke, M. Lindauer, F. Hutter, and A. Sunyaev. Practitioner motives to select hyperparameter optimization methods, 2025. URL <https://arxiv.org/abs/2203.01717>.

References

- F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. Multi-objective hyperparameter optimization in machine learning – An overview. *ACM Trans. Evol. Learn. Optim.*, 3(4), 2023.
- Z. Karnin, T. Koren, and O. Somekh. Almost optimal exploration in multi-armed bandits. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1238–1246, 2013.
- A. J. Keane. Statistical improvement criteria for use in multiobjective design optimization. *AIAA Journal*, 44(4):879–891, 2006.
- J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.
- P. Kent and J. Branke. BOP-Elites, a Bayesian optimisation algorithm for quality-diversity search, 2020. URL <https://arxiv.org/abs/2005.04320>.
- P. Kent, A. Gaier, J.-B. Mouret, and J. Branke. BOP-Elites, a Bayesian optimisation approach to quality diversity search with black-box descriptor functions, 2023. URL <https://arxiv.org/abs/2307.09326>.
- P. Kerschke and H. Trautmann. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, 27(1):99–127, 2019.
- P. Kerschke, M. Preuss, C. Hernández, O. Schütze, J.-Q. Sun, C. Grimme, G. Rudolph, B. Bischl, and H. Trautmann. Cell mapping techniques for exploratory landscape analysis. In A.-A. Tantar, E. Tantar, J.-Q. Sun, W. Zhang, Q. Ding, O. Schütze, M. T. M. Emmerich, P. Legrand, P. Del Moral, and C. A. Coello Coello, editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, pages 115–131, 2014.
- P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann. Detecting funnel structures by means of exploratory landscape analysis. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 265–272, 2015.
- J. Kim and S. Choi. On uncertainty estimation by tree-based surrogate models in sequential model-based optimization. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151, pages 4359–4375, 2022.
- D. P. Kingma and L. J. Ba. Adam: A method for stochastic optimization. In *The Third International Conference on Learning Representations*, 2015.
- A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything, 2023. URL <https://arxiv.org/abs/2304.02643>.
- A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536, 2017a.

- A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *The Fifth International Conference on Learning Representations*, 2017b.
- A. Klein, L. C. Tiao, T. Lienart, C. Archambeau, and M. Seeger. Model-based asynchronous hyperparameter and neural architecture search, 2020. URL <https://arxiv.org/abs/2003.10865>.
- J. Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.
- B. J. Koch and D. Peterson. From protoscience to epistemic monoculture: How benchmarking set the stage for the deep learning revolution, 2024. URL <https://arxiv.org/abs/2404.06647>.
- B. J. Koch, E. Denton, A. Hanna, and J. G. Foster. Reduced, reused and recycled: The life of a dataset in machine learning research. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical Report*, 2009. URL <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, 2012.
- H. J. Kushner. A new method of locating the maximum point of an arbitrary multi-peak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.
- B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017.
- Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits, 1998. URL <http://yann.lecun.com/exdb/mnist/>.
- J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011a.
- J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 211–218, 2011b.
- B. Letham, B. Karrer, G. Ottoni, and E. Bakshy. Constrained Bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519, 2019.
- J.-C. Lévesque. *Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces*. PhD thesis, Université Laval, 2018.
- C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, and Y. Lin. HW-NAS-Bench: Hardware-aware neural architecture search benchmark, 2021.

References

- L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18 (185):1–52, 2018.
- L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, 2020.
- R. Li, M. T. M. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. C. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21 (1):29–64, 2013.
- X. Li, Y. Wang, and R. Ruiz. A survey on sparse learning models for feature selection. *IEEE Transactions on Cybernetics*, 52(3):1642–1660, 2022.
- Y. L. Li, T. G. J. Rudner, and A. G. Wilson. A study of Bayesian neural network surrogates for Bayesian optimization. In *The Sixth International Conference on Learning Representations*, 2024.
- M. Lindauer and F. Hutter. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18, 2020.
- M. Lindauer, M. Feurer, K. Eggenberger, A. Biedenkapp, and F. Hutter. Towards assessing the impact of Bayesian optimization’s own hyperparameters. In *IJCAI Workshop on Data Science Meets Optimisation*, 2019.
- M. Lindauer, K. Eggenberger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.
- M. Lindauer, F. Karl, A. Klier, J. Moosbauer, A. Tornede, A. C. Mueller, F. Hutter, M. Feurer, and B. Bischl. Position: A call to action for a human-centered AutoML paradigm. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 30566–30584, 2024.
- H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *The Sixth International Conference on Learning Representations*, 2018.
- H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *The Seventh International Conference on Learning Representations*, 2019.
- Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti. NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 35–51, 2020.
- D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2113–2122, 2015.

- N. Mallik, E. Bergman, C. Hvarfner, D. Stoll, M. Janowski, M. Lindauer, L. Nardi, and F. Hutter. PriorBand: Practical hyperparameter optimization in the age of deep learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 7377–7391, 2023.
- M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2): 239–245, 1979.
- McKinsey & Company. AI adoption advances, but foundational barriers remain, 2018. URL <https://www.mckinsey.com/featured-insights/artificial-intelligence/ai-adoption-advances-but-foundational-barriers-remain>.
- Y. Mehta, C. White, A. Zela, A. Krishnakumar, G. Zabergja, S. Moradian, M. Safari, K. Yu, and F. Hutter. NAS-Bench-Suite: NAS evaluation is (now) surprisingly easy. In *The Tenth International Conference on Learning Representations*, 2022.
- J. Mellor, J. Turner, A. Storkey, and E. J. Crowley. Neural architecture search without training. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 7588–7598, 2021.
- O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 829–836, 2011.
- T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- J. Moćkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques: IFIP Technical Conference*, 1974.
- A. Mohan, C. Benjamins, K. Wienecke, A. Dockhorn, and M. Lindauer. AutoRL hyperparameter landscapes. In A. Faust, R. Garnett, C. White, F. Hutter, and J. R. Gardner, editors, *Proceedings of the Second International Conference on Automated Machine Learning*, volume 224, pages 13/1–27, 2023.
- C. Molnar, G. Casalicchio, and B. Bischl. Quantifying model complexity via functional decomposition for better post-hoc interpretability. In P. Cellier and K. Driessens, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 193–204, 2020.
- J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites, 2015. URL <https://arxiv.org/abs/1504.04909>.
- M. A. Muñoz, M. Kirley, and S. K. Halgamuge. Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, 19(1):74–87, 2015.
- S. Müller, N. Hollmann, S. P. Arango, J. Grabocka, and F. Hutter. Transformers can do Bayesian inference. In *The Tenth International Conference on Learning Representations*, 2022.
- C. Nadeau and Y. Bengio. Inference for the generalization error. In S. S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, 1999.
- R. M. Neal. *Bayesian Learning for Neural Networks*. Springer, New York, 1996.

References

- M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.
- W. Orr and E. B. Kang. AI as a sport: On the competitive epistemologies of benchmarking. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*, pages 1875–1884, 2024.
- A. Paleyes, R.-G. Urma, and N. D. Lawrence. Challenges in deploying machine learning: A survey of case studies. *ACM Computing Surveys*, 55(6):1–29, 2022.
- F. Pedregosa. Hyperparameter optimization with approximate gradient. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 737–746, 2016.
- D. Perez-Liebana, C. Guerrero-Romero, A. Dockhorn, L. Xu, J. Hurtado, and D. Jeurissen. Generating diverse and competitive play-styles for strategy games. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021.
- V. Perrone, I. Shcherbatyi, R. Jenatton, C. Archambeau, and M. Seeger. Constrained Bayesian optimization with max-value entropy search. In *NeurIPS Workshop on Meta-Learning*, volume 3, 2019.
- H. Pham, M. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 4095–4104, 2018.
- V. Picheny, R. B. Gramacy, S. Wild, and S. Le Digabel. Bayesian optimization under mixed constraints with a slack-variable augmented Lagrangian. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, 2016.
- S. Pineda Arango, H. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted \mathcal{S} -metric selection. In G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, editors, *Parallel Problem Solving from Nature – PPSN X*, pages 784–794, 2008.
- M. Preuss. *Multimodal Optimization by Means of Evolutionary Algorithms*. Springer, Cham, 2015.
- P. Probst, A.-L. Boulesteix, and B. Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.
- Y. Pushak and H. H. Hoos. Algorithm configuration landscapes: More benign than expected? In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV*, pages 271–283, 2018.

- A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training, 2018.
- D. Raji, E. Denton, E. M. Bender, A. Hanna, and A. Paullada. AI and the everything in the whole wide world benchmark. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.
- H. Rakotoarison, S. Adriaensen, N. Mallik, S. Garibov, E. Bergman, and F. Hutter. In-context freeze-thaw Bayesian optimization for hyperparameter optimization. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 41982–42008, 2024.
- E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2902–2911, 2017.
- E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, 2019.
- B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do CIFAR-10 classifiers generalize to CIFAR-10?, 2018. URL <https://arxiv.org/abs/1806.00451>.
- B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do ImageNet classifiers generalize to ImageNet? In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 5389–5400, 2019.
- H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.
- H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- B. Ru, A. Alvi, V. Nguyen, M. A. Osborne, and S. Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 8276–8285, 2020.
- D. Salinas, V. Perrone, O. Cruchant, and C. Archambeau. A multi-objective perspective on jointly tuning hardware and hyperparameters. In *ICLR Workshop on Neural Architecture Search*, volume 2, 2021.
- D. Salinas, M. Seeger, A. Klein, V. Perrone, M. Wistuba, and C. Archambeau. Syne Tune: A library for large scale hyperparameter tuning and reproducible research. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188, pages 16/1–23, 2022.
- A. Saltelli. Sensitivity analysis for importance assessment. *Risk Analysis*, 22(3):579–590, 2002.
- E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. A survey of methods for automated algorithm configuration. *Journal of Artificial Intelligence Research*, 75:425–487, 2022.

References

- D. Schlangen. Targeting the benchmark: On methodology in current natural language processing research. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 670–674, 2021.
- R. Schmucker, M. Donini, V. Perrone, and C. Archambeau. Multi-objective multi-fidelity hyperparameter optimization with application to fairness. In *NeurIPS Workshop on Meta-Learning*, volume 4, 2020.
- S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran. Operationalizing machine learning: An interview study, 2022. URL <https://arxiv.org/abs/2209.09125>.
- D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Timothy Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
- D. Simon. *Evolutionary Optimization Algorithms*. Wiley, New Jersey, 1 edition, 2013.
- K. Smith-Miles. Understanding instance hardness for optimisation algorithms: Methodologies, open challenges and post-quantum implications. *Applied Mathematical Modelling*, 142:115965, 2025.
- J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, 2012.
- J. Snoek, K. Swersky, R. Zemel, and R. P. Adams. Input warping for Bayesian optimization of non-stationary functions. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 1674–1682, 2014.
- J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. P. Adams. Scalable Bayesian optimization using deep neural networks. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2171–2180, 2015.
- I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802, 1967.
- J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, 2016.
- N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 1015–1022, 2010.
- G. Stiglic, P. Kocbek, N. Fijacko, M. Zitnik, K. Verbert, and L. Cilar. Interpretability of machine learning-based prediction models in healthcare. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(5):e1379, 2020.

- Y. Sun, Q. Song, X. Gui, F. Ma, and T. Wang. AutoML in the wild: Obstacles, workarounds, and expectations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023.
- M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.
- H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6377–6389, 2020.
- C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855, 2013.
- A. Tornede, D. Deng, T. Eimer, J. Giovanelli, A. Mohan, T. Ruhkopf, S. Segel, D. Theodorakopoulos, T. Tornede, H. Wachsmuth, and M. Lindauer. AutoML in the age of large language models: Current challenges, future opportunities and risks. *Transactions on Machine Learning Research*, 2024.
- B. Tu, A. Gandy, N. Kantas, and B. Shafei. Joint entropy search for multi-objective Bayesian optimization. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 9922–9938, 2022.
- J. Vamathevan, D. Clark, P. Czodrowski, I. Dunham, E. Ferran, G. Lee, B. Li, A. Madabhushi, P. Shah, M. Spitzer, and S. Zhao. Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery*, 18(6):463–477, 2019.
- J. N. van Rijn and F. Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2367–2376, 2018.
- J. Vanschoren. Meta-learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 35–61. Springer International Publishing, Cham, 2019.
- J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017.
- M. Velikova and H. Daniels. Decision trees for monotone price models. *Computational Management Science*, 1:231–244, 2004.
- A. Vellido, J. D. Martín-Guerrero, and P. J. G. Lisboa. Making machine learning models interpretable. In *20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 163–172, 2012.

References

- A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In T. Linzen, G. Chrupała, and A. Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, 2018.
- T. Wang and Q. Lin. Hybrid predictive models: When an interpretable model collaborates with a black-box model. *Journal of Machine Learning Research*, 22(137):1–38, 2021.
- Z. Wang and S. Jegelka. Max-value entropy search for efficient Bayesian optimization. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3627–3635, 2017.
- C. White, W. Neiswanger, S. Nolen, and Y. Savani. A study on encodings for neural architecture search. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20309–20319, 2020.
- C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10293–10301, 2021a.
- C. White, S. Nolen, and Y. Savani. Exploring the loss landscape in neural architecture search. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161, pages 654–664, 2021b.
- C. White, A. Zela, R. Ru, Y. Liu, and F. Hutter. How powerful are performance predictors in neural architecture search? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 28454–28469, 2021c.
- C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter. Neural architecture search: Insights from 1000 papers, 2023. URL <https://arxiv.org/abs/2301.08727>.
- C. K. I. Williams and C. E. Rasmussen. *Gaussian Processes for Machine Learning*. The MIT press, Cambridge, MA, 2006.
- A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pages 370–378, 2016.
- J. T. Wilson, R. Moriconi, F. Hutter, and M. P. Deisenroth. The reparameterization trick for acquisition functions. In *NIPS Workshop on Bayesian Optimization*, 2017.
- J. T. Wilson, F. Hutter, and M. P. Deisenroth. Maximizing acquisition functions for Bayesian optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.
- M. Wistuba, A. Rawat, and T. Pedapati. A survey on neural architecture search, 2019. URL <https://arxiv.org/abs/1905.01392>.
- M. Wistuba, A. Kadra, and J. Grabocka. Supervising the multi-fidelity race of hyperparameter configurations. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 13470–13484, 2022.

-
- L. Xiang, R. Hunter, M. Xu, Ł. Dudziak, and H. Wen. Exploiting network compressibility and topology in zero-cost NAS. In A. Faust, R. Garnett, C. White, F. Hutter, and J. R. Gardner, editors, *Proceedings of the Second International Conference on Automated Machine Learning*, volume 224, pages 18/1–14, 2023.
- Y. Xin, S. Luo, H. Zhou, J. Du, X. Liu, Y. Fan, Q. Li, and Y. Du. Parameter-efficient fine-tuning for pre-trained vision models: A survey, 2024. URL <https://arxiv.org/abs/2402.02242>.
- K. Yang, M. T. M. Emmerich, A. Deutz, and T. Bäck. Efficient computation of expected hyper-volume improvement using box decomposition algorithms. *Journal of Global Optimization*, 75: 3–34, 2019.
- L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.
- C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-bench-101: Towards reproducible neural architecture search. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 7105–7114, 2019.
- A. Zela, J. Siems, L. Zimmer, J. Lukasik, M. Keuper, and F. Hutter. Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks. In *The Tenth International Conference on Learning Representations*, 2022.
- C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.
- E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - A comparative case study. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 292–304, 1998.
- B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *The Fifth International Conference on Learning Representations*, 2017.
- B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

Excluded Publications

Some works of the author have not been included as contributions in this thesis. Two major software packages within the R system for statistical computing¹ are given by *mlr3pipelines*² and *mlr3mbo*³. The *mlr3pipelines* package extends *mlr3*⁴ with a flexible dataflow programming toolkit, enabling the composition of diverse pipelining operators into graphs for data preprocessing, model fitting, and ensemble learning, which can be treated as *mlr3* learners for resampling, benchmarking, and tuning. The *mlr3mbo* package provides a modern and flexible framework for BO and model-based optimization, relying on the *bbotk*⁵ package to offer both ready-to-use algorithms and customizable building blocks for single- and multi-objective optimization across mixed search spaces, seamlessly integrating with *mlr3* for HPO of learners. The author of this thesis contributed to both software packages. HPO in *mlr3* and the *mlr3mbo* package are introduced in more detail in Becker et al. (2024) and Schneider and Becker (2023). Other works not included in this thesis are given by publications in which the author of this thesis contributed as a co-author with minor contributions and responsibility. We list all publications excluded as contributions in the following bibliography.

¹<https://www.R-project.org/>

²<https://cran.r-project.org/package=mlr3pipelines>

³<https://cran.r-project.org/package=mlr3mbo>

⁴<https://cran.r-project.org/package=mlr3>

⁵<https://cran.r-project.org/package=bbotk>

Excluded Publications

- M. Becker, L. Schneider, and S. Fischer. Hyperparameter optimization. In B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors, *Applied Machine Learning Using mlr3 in R*, pages 85–115. Chapman and Hall/CRC, New York, 2024.
- L. Schneider and M. Becker. Advanced tuning methods and black box optimization. In B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors, *Applied Machine Learning Using mlr3 in R*, pages 116–145. Chapman and Hall/CRC, New York, 2023.
- M. Binder, F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl. mlr3pipelines - Flexible machine learning pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021.
- R. Hornung, M. Nalenz, L. Schneider, A. Bender, L. Bothmann, B. Bischl, T. Augustin, and A.-L. Boulesteix. Evaluating machine learning models in non-standard settings: An overview and new findings, 2023. URL <https://arxiv.org/abs/2310.15108>.
- F. Karl*, T. Pielok*, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. Multi-objective hyperparameter optimization in machine learning – An overview. *ACM Trans. Evol. Learn. Optim.*, 3(4), 2023.
- L. O. Purucker, L. Schneider, M. Anastacio, J. Beel, B. Bischl, and H. H. Hoos. Q(D)O-ES: Population-based quality (diversity) optimisation for post hoc ensemble selection in AutoML. In A. Faust, R. Garnett, C. White, F. Hutter, and J. R. Gardner, editors, *Proceedings of the Second International Conference on Automated Machine Learning*, volume 224, pages 10/1–34, 2023.

Contributing Publications

- T. Nagler*, L. Schneider*, B. Bischl, and M. Feurer. Reshuffling resampling splits can improve generalization of hyperparameter optimization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 40486–40533, 2024.
- L. Schneider*, L. Schäpermeier*, R. P. Prager*, B. Bischl, H. Trautmann, and P. Kerschke. HPO × ELA: Investigating hyperparameter optimization landscapes by means of exploratory landscape analysis. In G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, editors, *Parallel Problem Solving from Nature – PPSN XVII*, pages 575–589, 2022.
- L. Schneider, B. Bischl, and J. Thomas. Multi-objective optimization of performance and interpretability of tabular supervised machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 538–547, 2023.
- L. Schneider, F. Pfisterer, P. Kent, J. Branke, B. Bischl, and J. Thomas. Tackling neural architecture search with quality diversity optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188, pages 9/1–30, 2022a.
- F. Pfisterer*, L. Schneider*, J. Moosbauer, M. Binder, and B. Bischl. YAHPO Gym – An efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188, pages 3/1–39, 2022.
- L. Schneider, F. Pfisterer, J. Thomas, and B. Bischl. A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2136–2142, 2022b.
- J. Moosbauer*, M. Binder*, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers. *IEEE Transactions on Evolutionary Computation*, 26(6):1336–1350, 2022.
- L. Schneider, F. Pfisterer, M. Binder, and B. Bischl. Mutation is all you need. In *8th ICML Workshop on Automated Machine Learning*, 2021.
- R. P. Prager*, K. Dietrich*, L. Schneider, L. Schäpermeier, B. Bischl, P. Kerschke, H. Trautmann, and O. Mersmann. Neural networks as black-box benchmark functions optimized for exploratory landscape features. In *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 129–139, 2023.

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12. Juli 2011, § 8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Während des Überarbeitungsprozesses habe ich Grammarly verwendet, um Rechtschreibfehler und grammatikalische Fehler zu erkennen und zu korrigieren. Außerdem habe ich ChatGPT (GPT-4o) als Hilfsmittel verwendet, um bestimmte einzelne Sätze zu verbessern. Insbesondere habe ich mir Vorschläge zur Aufteilung längerer Sätze und zur Verbesserung der Lesbarkeit generieren lassen. Diese Vorschläge wurden nicht wortwörtlich übernommen, sondern manuell überprüft, angepasst und umformuliert, um sie mit meinem eigenen Schreibstil in Einklang zu bringen.

München, den 17.03.2025

Lennart Paul Schneider

