Selbstorganisation, Selbstausrichtung und Selbstanpassung in lernenden Multi-Agenten-Systemen

Fabian Ritz

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

Tag der Einreichung: 13.08.2024

Selbstorganisation, Selbstausrichtung und Selbstanpassung in lernenden Multi-Agenten-Systemen

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

eingereicht von

Fabian Ritz

13.08.2024

1. Gutachterin:	Prof. Dr. Claudia Linnhoff-Popien
2. Gutachterin:	Prof. Dr. Paula Herber
Tag der mündlichen Prüfung:	12.02.2025

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, 17.02.2025 Fabian Ritz

Danksagung

Einen ganz besonderen Dank möchte ich zunächst an Claudia Linnhoff-Popien richten, die mich seit 2019 in diesem für mich neuen Terrain begleitete und unterstützte. Sie ermöglichte mir, im Auftrag ihrer Industriepartner SWM und Siemens zu forschen und Erfahrung in der Lehre zu sammeln, beispielsweise bei der Durchführung des Praktikums Autonome Systeme. Auch bei der Durchführung ihrer jährlichen Konferenz DIGICON konnte ich auch abseits meines fachlichen Schwerpunkts viel Neues lernen.

Des Weiteren gilt mein herzlicher Dank Paula Herber für das mehrfache Feedback zu dieser Dissertation, wodurch Struktur und Fokus stark profitierten, sowie dafür, dass sie sich als Zweitberichterstatterin zur Verfügung stellte. Mein Dank gilt darüber hinaus Martin Wirsing, der den Vorsitz der Prüfungskommission übernahm.

Ebenso gilt mein Dank den Kolleginnen, Kollegen und Weggefährten am Lehrstuhl für Mobile und Verteilte Systeme. Ohne Katja Grenner und Melisa Delic im Sekretariat wäre ich im Dschungel der Formulare und Vorschriften verschollen. Meinem Zimmerkollegen Robert Müller ist es zu verdanken, dass ich lernte, wie der Lehrstuhl funktioniert und das erste Jahr heil überstand. Thomy Phan, der mir seit dem gemeinsamen Bachelorstudium stets einen Schritt voraus ist, war mir über alle Publikationen und Lehrveranstaltungen hinweg ein großartiger Mentor. Er hat zusammen mit Andreas Sedlmeier maßgeblich dafür gesorgt, dass wir insgesamt fünf Jahre lang für Siemens forschen durften. Hinzu kommen all die engagierten Ko-Autoren, die mir halfen, meine Gedanken verständlich darzustellen. Hier möchte ich insbesondere Thomas Gabor und Lenz Belzner erwähnen. Ihr erfahrener Blick über die Fragestellung der jeweiligen Publikation hinaus war von unschätzbarem Wert.

Darüber hinaus möchte ich Andrea Gensicke und Michael Hannemann dafür danken, dass sie mich auf die vielen sprachlich schlechten Angewohnheiten aufmerksam machten, die sich über die Jahre durch den begrenzten Platz in meine wissenschaftlichen Publikationen einschlichen. So konnte ich diese in der vorliegenden Dissertation vermeiden.

Zu guter Letzt möchte ich den drei wichtigsten Menschen in meinem Leben für ihre unerschöpfliche Geduld und Unterstützung danken: meiner Frau Diana und meinen Kindern Jonathan und Elena. Ohne die Zeit, die sie mir für Experimente und Publikationen nach Feierabend und am Wochenende einräumten, hätte ich diese Dissertation nicht realisieren können.

Zusammenfassung

Die Kontrolle globaler Effekte, die aus lokalen Interaktionen entstehen, ist eine Herausforderung bei der Entwicklung von Multi-Agenten-Systemen, da Emergenz schwer vorherzusehen ist. Dennoch sollten sich autonome Agenten koordinieren, wenn Abhängigkeiten zwischen ihren Aktivitäten bestehen, und kooperieren, wenn es das globale Wohlergehen verbessert, selbst wenn sie individuelle Ziele verfolgen. Diese Arbeit untersucht, wie Agenten sich *selbst organisieren*, wenn ihnen diesbezüglich keine Regeln vorgeben sind, sondern ihnen die Entscheidungen überlassen werden. Die Agenten werden mittels *Multi-Agent Reinforcement Learning* realisiert, einer Variante des *Machine Learning* (ML), bei der mehrere Teile eines Systems (*Agenten*) durch wiederholte Interaktion mit einer Problemstellung und Beobachtung der Konsequenzen lernen. Daraus erwächst eine weitere Herausforderung: Auch wenn den Agenten bewusst ein hoher Freiheitsgrad gelassen wird, dürfen diese kein unerwünschtes oder gefährliches Verhalten entwickeln, sondern sie sollen ihr Verhalten selbstständig an bestimmten Regeln *ausrichten*. Wenn der gesamte Lebenszyklus solcher Agenten aus der Perspektive des Software Engineering betrachtet wird, können solche Agenten *selbstanpassende* Systeme realisieren, wenn ihnen entsprechende Meta-Ziele vorgegeben und in Feedbackschleifen berücksichtigt werden. Da diese Anpassung idealerweise automatisch erfolgt, ist es essentiell, den Ablauf vorab präzise definieren und zur Laufzeit kontrollieren zu können.

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf lokaler und globaler Ebene lösen können. Die Arbeit besteht aus drei Abschnitten, die diese These in den folgenden Bereichen untersuchen:

(i) **Selbstorganisation**

(ii) **Selbstausrichtung**

(iii) **Selbstanpassung**

Abschnitt (i) untersucht die zentrale These experimentell in einem unterspezifizierten Szenario, Abschnitt (ii) in einem genau spezifizierten. Abschnitt (iii) setzt die gewonnen Erkenntnisse in den Kontext eines Prozessmodells für die Entwicklung von ML-Komponenten, das deren gesamten Lebenszyklus abdeckt. Auf dieser Basis könnte künftig ein *Self-Engineering* konzipiert werden, bei dem lernende Agenten Teile des Entwicklungsprozesses selbst durchführen.

Abstract

Controlling global effects arising from local interactions is a challenge in engineering multi-agent systems because emergence is difficult to predict. Nevertheless, autonomous agents should always *coordinate* when there are dependencies between their activities, and *cooperate* when it improves global welfare, even if they have different goals. This thesis investigates how agents *organize* themselves when no according rules are imposed and the decision is left to them. The agents are realized with *Multi-Agent Reinforcement Learning*, a variant of *Machine Learning* (ML) in which multiple parts of a system (*agents*) learn by repeatedly interacting with a problem and observing the consequences. This results in another challenge: even if the agents are deliberately given a high degree of freedom, they should not develop undesirable or even dangerous behavior but *align* with applicable rules. Considering the entire lifecycle of such agents from a software engineering perspective, these agents can realize *self-adaptive systems* if additional meta-goals are defined and considered in feedback loops. Since the adaptation process should ideally be completely automated, it is essential to be able to precisely define this process in advance and control it at runtime.

The central hypothesis of this thesis is that learning agents can solve problems on a local and global scale themselves. The thesis consists of three sections that investigate this hypothesis in the following areas:

(i) **Self-Organization**

(ii) **Self-Alignment**

(iii) **Self-Adaptation**

Section (i) examines the central hypothesis experimentally in an underspecified scenario, section (ii) in a well-specified one. Section (iii) places the gained insights in the context of a process model for the development of ML components that covers their entire lifecycle. Based on our findings, a *self-engineering* could be designed in the future, where learning agents perform parts of the engineering process themselves.

Inhaltsverzeichnis

1 Einleitung

Abseits des jüngsten Hypes um große Sprachmodelle wie GPT-4 [23] ist künstliche Intelligenz in Form autonomer Systeme an anderen Stellen längst in der Praxis angekommen. Moderne PKW verfügen über eine Vielzahl von Assistenzsystemen, die zunehmend komplexe Aufgaben wie das Einparken übernehmen können, und *Fahrerlose Transportfahrzeuge* (FTF) transportieren Werkstücke in intelligenten Fertigungssystemen, die Kleinstmengen bis hin zur Lot-Size One [70] effizient produzieren. Stand heute werden FTF jedoch meist durch zentrale Controller koordiniert [89, 135] und PKW fahren noch nicht durchweg autonom. Die Fähigkeit der Systeme, das Verhalten an dynamische Probleme anzupassen [140], ist in diesen Beispielen noch nicht voll ausgeprägt. Ideale autonome Agenten sind gemäß Wooldrige et al. [160]

- *reaktiv*: sie erkennen Veränderungen und reagieren rechtzeitig,
- *proaktiv*: sie verfolgen Ziele nicht rein ereignisorientiert, sondern erkennen Chancen und ergreifen Initiative,
- *sozial*: sie ignorieren andere Agenten nicht, sondern interagieren mit ihnen, beispielsweise mittels Kommunikation.

Diese Arbeit betrachtet autonome Agenten in Szenarien, in denen nicht alle (Konflikt-)Situationen a priori bekannt sind, beispielsweise in offenen Umgebungen, in denen sich weitere Akteure wie andere Agenten oder Menschen befinden. Agenten verfolgen hier weder zwangsläufig das gleiche Ziel, noch verhalten sie sich stets kooperativ. Dies ist bei der Entwicklung zu berücksichtigen, da das Gesamtsystem ansonsten versagt [109]. Die Eigenschaft *sozial* spielt dabei eine zentrale Rolle und kann gemäß Consoli et al. differenziert werden [29]: Einerseits sollen sich Agenten *koordinieren*, da Abhängigkeiten zwischen ihren Aktivitäten bestehen können. Dies ist beispielsweise relevant, wenn sich Agenten Ressourcen wie Ladestrom oder Parkplätze teilen. Andererseits sollen Agenten *kooperieren*, um ein gemeinsames Ziel zu erreichen. Kooperation erfordert Koordination, um sich initial auf ein gemeinsames Ziel zu einigen, und ist beispielsweise dann relevant, wenn ein Agent eine Aufgabe nicht allein erfüllen kann. Darüber hinaus können Agenten *verhandeln* und *koalieren* [136], was jedoch außerhalb des Fokus dieser Arbeit liegt.

Diese Arbeit realisiert autonome Agenten mittels *Multi-Agent Reinforcement Learning* (MARL) [4], einer Variante des *Machine Learning* (ML) [95], bei der mehrere Komponenten eines Systems (Agenten) durch wiederholte Interaktion mit einer Problemstellung und Beobachtung der Konsequenzen lernen. Die

Agenten werden mit der derzeit häufigsten Technik realisiert, *Deep Learning* (DL), aufgrund der Fähigkeit zu generalisieren [73]. Dies ist hinsichtlich der Lösung dynamischer Probleme und dem Einsatz in entsprechenden Szenarien hilfreich.

1.1 Herausforderungen

Im Folgenden werden autonome Agenten betrachtet, die zu situativer Koordination und Kooperation ohne a priori definierte Regeln fähig sind. Solche Agenten fallen in den Bereich der *(Selbst-)Organisation* in Multi-Agenten-Systemen [16, 35, 113, 162]. Die Selbstorganisation entsteht hier als globaler Effekt auf Basis lokaler Interaktion. Solche emergenten Effekte sind schwer zu kontrollieren und nicht vollständig vorhersehbar. Darin besteht jedoch auch eine Chance, siehe Serugendo et al. [35]:

> „Aufgrund der Dynamik und Offenheit heutiger Agentenumgebungen [...] ist das Verständnis der Mechanismen, die zur Modellierung, Bewertung und Entwicklung von selbstorganisierendem Verhalten in MAS verwendet werden können, von großem Interesse. [...] Anstatt zu versuchen, emergente Phänomene zu eliminieren, könnte es interessant sein, zu untersuchen, wie diese bewusst erreicht und genutzt werden können.“ [35, S. 166-171, übersetzt]

Da autonome Agenten per Definition aktiv Einfluss auf einander und ihre Umgebung nehmen, beeinflussen sie die Ausprägung emergenter Effekte [56, 112]. Jedoch ist die Frage nach der Entstehung emergenter Kooperation nicht abschließend beantwortet [30, 103]. Der Fokus dieser Arbeit liegt auf Szenarien, in denen Kooperation für Individuen nicht unmittelbar rational, für das Gesamtsystem aber erstrebenswert ist, weil dadurch langfristig das globale Wohlergehen steigt [8, 55]. Insbesondere werden den Agenten keine Verhaltensregeln vorgegeben und es wird untersucht, wann emergente Kooperation zwischen den Agenten entsteht.

Die Kontrolle globaler Effekte, die aus lokalen Interaktionen entstehen, war bereits vor dem verbreiteten Einsatz von ML-Techniken eine fundamentale Herausforderung bei der Entwicklung von Multi-Agenten-Systemen [82]. Wenn Agenten zusätzlich *lernen* und dadurch ihr Verhalten im Laufe der Zeit anpassen, ist dies mit von Hand definierten Regeln langfristig unmöglich. Daraus ergibt sich eine weitere Herausforderung, der sich diese Arbeit widmet: Auch wenn Agenten bewusst einen hoher Freiheitsgrad gelassen wird, sollen diese in bestimmten Bereichen geltende Regeln einhalten und kein unerwünschtes oder sogar gefährliches Verhalten lernen [44, 51, 128]. Dies wird in der Literatur als *Alignment* (Ausrichtung) bezeichnet [43, 59, 79]. Im Rahmen dieser Arbeit lernen Agenten selbstständig, ihr Verhalten an den gegebenen Regeln auszurichten, indem ihnen diese zugänglich gemacht werden.

Wird der gesamte Lebenszyklus lernender Agenten aus der Perspektive des *Software Engineering* (SE) betrachtet, können solche Agenten *selbstanpassende Systeme* [63] realisieren, wenn ihnen entsprechende Meta-Ziele vorgegeben und in Feedbackschleifen berücksichtigt werden. Da die Anpassung im Idealfall vollständig automatisch erfolgt, ist es essentiell, ihren Ablauf vorab präzise definieren und zur Laufzeit kontrollieren zu können. In der Literatur wurden verschiedene Ansätze präsentiert, um das iterative Vorgehen beim ML mit agilen Entwicklungsmethoden zu kombinieren [75, 91, 141]. Diese Arbeit schließt eine Lücke, indem sie sich auf die Abhängigkeiten zwischen Aktivitäten, beispielsweise der Definition von Problemstellung und Lernsignal, und Artefakten, beispielsweise einer gegebenen Spezifikation oder den (trainierten) Agenten, fokussiert. Diese präzise mittels einer grafischen Syntax darzustellen, die das Paradigma der Selbstanpassung berücksichtigt, ist die dritte Herausforderung, der sich diese Arbeit widmet.

1.2 Zentrale These

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf lokaler und globaler Ebene lösen können. Inhaltlich ist die Arbeit in drei Abschnitte gegliedert, die diese These in den folgenden Bereichen untersuchen:

(i) **Selbstorganisation**: Wenn Agenten individuelle Ziele in offenen Umgebungen verfolgen, müssen sie Einigkeit über die Nutzung gemeinsamer Ressourcen erlangen, um vorübergehend zusammenarbeiten zu können. Um optimale Regeln dafür a priori zu definieren, müssten alle entsprechenden (Konflikt-) Situationen vorhergesehen werden können, was in der Praxis selten möglich ist. Diese Arbeit schlägt vor, die Agenten selbst lernen zu lassen, wann und wie sie miteinander interagieren.

(ii) **Selbstausrichtung**: In geschlossenen Umgebungen oder definierten Situationen existiert meist eine klare Spezifikation des Problems und des gewünschten Verhaltens der Agenten. Um die Einhaltung dieser Spezifikation zu gewährleisten, wird nicht konformem und potentiell gefährlichem Verhalten oft mit zusätzlichen Mechanismen vorgebeugt, beispielsweise mittels vorgegebener Rückfallstrategien. Das manuelle Design dieser Mechanismen wird mit zunehmend autonomen Agenten problematisch. Diese Arbeit schlägt vor, die Agenten von vornherein mit der gesamten Spezifikation zu trainieren, sodass sie selbst lernen, ihr Verhalten daran auszurichten.

(iii) **Selbstanpassung**: Wenn Agenten lernen können, wie sie sowohl in unterspezifizierten Situationen wie in Abschnitt (i) als auch in gut spezifizierten Situationen wie in Abschnitt (ii) erfolgreich sind, können sie mit Hilfe von Meta-Zielen selbstanpassende Systeme realisieren. Diese

Arbeit schlägt ein formales Prozessmodell mit grafischer Darstellung für die Entwicklung ML-basierter Systeme vor, das systematisch durch deren Lebenszyklus führt und eine gezielte Kontrolle ermöglicht.

1.3 Beiträge & Vorveröffentlichungen

Die drei Abschnitte dieser Arbeit können an unterschiedlichen Stellen eines typischen ML-Entwicklungszyklus (siehe Abbildung 1.1) verortet werden.

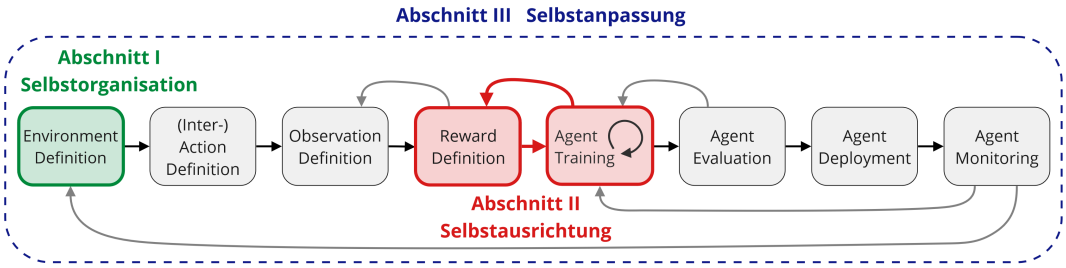


Abbildung 1.1: Bezug der drei Abschnitte dieser Arbeit zu Aktivitäten bei der Entwicklung von RL-Agenten innerhalb eines *ML-Entwicklungszyklus*. Darstellung basierend auf [91] und in ähnlicher Form zuvor veröffentlicht in [123].

Abschnitt (i) **Selbstorganisation** befindet sich am Anfang des Entwicklungszyklus und legt den Schwerpunkt auf die Definition der Problemstellung, die im *Reinforcement Learning* (RL) durch die Umgebung (englisch: Environment) repräsentiert wird. In diesem Abschnitt wird gezeigt, dass Agenten mittels RL koordiniertes und kooperatives Verhalten lernen können, wenn die Umgebung geeignete Voraussetzungen bietet. Dazu werden eigennützige Agenten im Umgang mit einer geteilten, selbsterneuernden Ressource trainiert. Untersucht wird, unter welchen Umständen die Agenten die Ressource restlos aufbrauchen, was zu einem Zusammenbruch des Systems führt, und wann sie lernen, nachhaltig zu handeln, indem sie stets einen Teil der Ressource übrig lassen, was zu einem stabilen System führt. Da den Agenten keine Regeln vorgegeben werden und bereits einzelne gierige Individuen einen Kollaps des Systems verursachen können, erfordert nachhaltiges Handeln eine erlernte Organisation aller Agenten. Wissenschaftlicher Beitrag dieses Abschnitts ist eine systematische Analyse, wie die Parametrisierung der Umgebung RL-Agenten hinsichtlich Lernerfolg, nachhaltigem Verhalten einzelner und mehrerer Agenten (Koordination), sowie aktive Zusammenarbeit bei der Nutzung der Ressource (Kooperation) beeinflusst. Die Inhalte dieser Analyse wurden bereits in zwei wissenschaftlichen Publikationen [120, 124] veröffentlicht. Die Beiträge der (Ko-)Autoren und die Übernahme der Inhalte im Rahmen dieser Arbeit sind Kapitel 3.1 dargelegt.

Abschnitt (ii) **Selbstausrichtung** befindet sich in der Mitte des ML-Entwicklungszyklus und legt den Fokus auf die Definition des Feedbacks während des Trainings, das im RL durch den Reward repräsentiert wird. In diesem Abschnitt wird gezeigt, dass Agenten mit RL spezifikationskonformes Laufzeitverhalten lernen können, sodass auf zusätzliche Sicherheitsschichten und manuell definierte Rückfallstrategien verzichtet werden kann. Dazu werden Agenten darauf trainiert, eine Systemspezifikation aus funktionalen und nicht-funktionalen Anforderungen einzuhalten. Anstatt den Einsatz von RL auf die funktionalen Anforderungen zu beschränken, werden den Agenten die gesamten Anforderungen schrittweise durch *Reward Shaping* so vermittelt, dass sie sukzessive lernen, ihr Verhalten an der Systemspezifikation auszurichten. Wissenschaftlicher Beitrag dieses Abschnitts ist zunächst eine empirische Analyse der Auswirkungen von *Reward Embeddings*, die verschiedene Aspekte einer Spezifikation widerspiegeln, auf das Training verschiedener MARL-Algorithmen in einer simulierten Lot-Size One Produktion. Zusätzlich stellt dieser Abschnitt ein konkretes Anwendungsbeispiel für proaktiv sicheres Laufzeitverhalten auf globaler Ebene vor. Die Analyse und das Anwendungsbeispiel wurden bereits in zwei wissenschaftlichen Publikationen [121, 122] veröffentlicht. Die Beiträge der (Ko-)Autoren und die Verwendung der Inhalte im Rahmen dieser Arbeit sind in Kapitel 4.1 dargelegt.

Abschnitt (iii) **Selbstanpassung** betrachtet den ML-Entwicklungszyklus als Teil eines größeren SE-Prozesses. In diesem Abschnitt wird eine Lücke geschlossen, die zwischen bestehenden SE-Prozessmodellen und dem ML-Entwicklungszyklus hinsichtlich einer einheitlichen Beschreibung von Aktivitäten, Artefakten und Feedback-Schleifen bestand. Wissenschaftlicher Beitrag dieses Kapitels ist zunächst eine grafische Syntax für eine einheitliche Beschreibung der Entwicklung von ML-Komponenten, die den gesamten Lebenszyklus abdeckt und gängigen ML-Methoden berücksichtigt. Die Syntax zeigt das Zusammenspiel von Aktivitäten und Artefakten und bildet gängige Phasen und Aktivitäten ab (beispielsweise die der beiden vorherigen Abschnitte). Zusätzlich stellt dieser Abschnitt einen Entwurf für eine ML-basierte Selbstanpassung auf der Grundlage automatisierter Feedback-Schleifen und V&V-Integrationspunkte vor. Das grafische Prozessmodell und der Entwurf für ML-basierte Selbstanpassung wurden bereits in einer wissenschaftlichen Publikation [123] veröffentlicht. Die Beiträge der (Ko-)Autoren und die Verwendung der Inhalte im Rahmen dieser Arbeit sind in Kapitel 5.1 dargelegt.

1.4 Struktur der Arbeit

Die weitere Arbeit ist folgendermaßen strukturiert:

- Kapitel 2 erklärt die Grundlagen der Anwendung von RL in MAS und schlägt die Brücke zu den in dieser Arbeit verwendeten Algorithmen und

Techniken. Anschließend erläutert es Herkunft und Verwendung der Begriffe *(Selbst-)Organisation*, *Ausrichtung* und *selbstanpassende Systeme*.

- Kapitel 3 beinhaltet Abschnitt (i) **Selbstorganisation**. Es untersucht experimentell, unter welchen Umständen Agenten mit RL in offenen Systemen selbstständig Koordination und Kooperation erlernen können.
- Kapitel 4 enthält Abschnitt (ii) **Selbstausrichtung**. Es analysiert in Experimenten, wie Agenten mit RL lernen können, gemeinsam eine gegebene Spezifikation zu erfüllen.
- Kapitel 5 beinhaltet Abschnitt (iii) **Selbstanpassung**. Es legt konzeptuell dar, wie mit Agenten der beiden vorhergegangenen Kapitel und zusätzlichen Meta-Zielen Systeme realisiert werden können, die sich über den gesamten Lebenszyklus selbst verbessern.
- Kapitel 6 fasst die wichtigsten Erkenntnisse zusammen und skizziert die daraus resultierenden offenen Fragen für künftige Forschung.

2 Definitionen und Grundlagen

Dieses Kapitel enthält die für den Rest der Arbeit erforderlichen Grundlagen und Definitionen. Zunächst führt Unterkapitel 2.1 *Multi-Agenten-Systeme* ein. Dann gibt Unterkapitel 2.2 einen Überblick über *Lernende Multi-Agenten-Systeme* und skizziert verschiedene Varianten, autonome Agenten mit Techniken des Reinforcement Learning zu realisieren. Die darauf folgenden Unterkapitel beschreiben die Grundlagen der drei Forschungsbereiche *(Selbst-) Organisation* in Unterkapitel 2.3, *Ausrichtung im ML* in Unterkapitel 2.4 und *Entwicklung selbstanpassender Systeme* in Unterkapitel 2.5, denen sich im späteren Verlauf der Arbeit jeweils ein ganzer Abschnitt widmet.

2.1 Multi-Agenten-Systeme

Gemäß Albrecht et al. [4] besteht ein *Multi-Agenten-System* (MAS) aus einer *Umgebung* (Environment) und mehreren entscheidungsfähigen *Agenten*, die in und mit dieser Umgebung interagieren, um bestimmte Ziele zu erreichen (siehe Abbildung 2.1). Die Umgebung eines MAS ist ein physisches oder simuliertes Problem, dessen *Zustand* (State) sich im Lauf der Zeit verändert. Die Aktionen der Agenten nehmen dabei Einfluss auf die Umgebung. Die Umgebung definiert, welche *Aktionen* welcher Agent zu einem Zeitpunkt ausführen kann und welche Informationen Agenten mittels *Beobachtung* (Observation) über den Zustand der Umgebung erhalten. Der Zustand der Umgebung kann diskret, kontinuierlich oder als Kombination aus beidem repräsentiert werden. Multi-Agenten-Umgebungen sind häufig dadurch gekennzeichnet, dass Agenten nur eine begrenzte Sicht auf die Umgebung haben. In solchen *partiell beobachtbaren* Umgebungen erhalten einzelne Agenten zu einzelnen Zeitpunkten nur unvollständige Informationen über den globalen Zustand, die sich zudem von denen anderer Agenten unterscheiden können. Insgesamt repräsentiert die Umgebung das zu lösende Problem sowie die zu unterschiedlichen Zeitpunkten darüber zur Verfügung stehenden Informationen samt Freiheitsgrade zur Lösung.

Ein Agent eines MAS ist eine autonome Einheit, die durch Beobachtung Informationen über den Zustand der Umgebung erhält und verschiedene Aktionen wählen kann, um deren Zustand zu beeinflussen. Agenten können unterschiedliches Vorwissen über die Umgebung haben. Das können beispielsweise alle möglichen Zustände der Umgebung oder Informationen darüber sein, wie Zustände durch Aktionen beeinflusst werden können. Agenten haben bestimmte

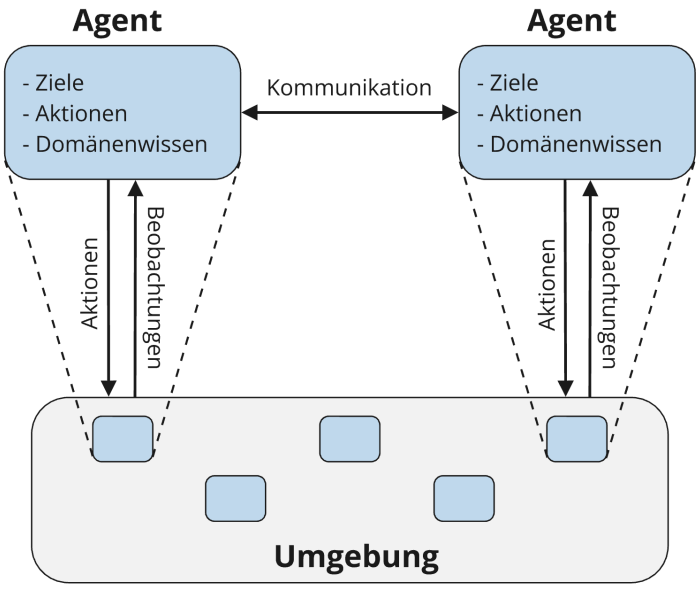


Abbildung 2.1: Schematische Darstellung eines Multi-Agenten-Systems bestehend aus einer Umgebung (hellgrau) und mehreren Agenten (blau), die Entscheidungen treffen. Die Agenten erhalten Informationen durch Beobachtung der Umgebung sowie Kommunikation miteinander und führen Aktionen aus, um ihre Ziele zu erreichen. Darstellung und Beschreibung gemäß [4, S.2].

Ziele und wählen ihre Aktionen so, dass sie diese Ziele erreichen. Ziele können darin bestehen, einen bestimmten Zustand der Umgebung zu erreichen oder bestimmte Metriken zu optimieren.

Zum Erreichen der Ziele können Algorithmen des *Sequential Decision Making* (SD) verwendet werden, beispielsweise *Monte Carlo Tree Search* [22] oder *Reinforcement Learning* (RL) [145]. Hier werden Ziele durch Funktionen repräsentiert, die Agenten Feedback in Form skalarer *Rewards* geben, wenn sie bestimmte Aktionen in bestimmten Zuständen ausführen, und die *Strategie* (englisch: policy) ist eine Funktion, mit der ein Agent basierend auf seinen Informationen über den Zustand der Umgebung eine Aktionen auswählt. Die iterative Interaktion zwischen Agenten und Umgebung wird in Abbildung 2.2 dargestellt.

Diese Arbeit betrachtet partiell beobachtbare Probleme, in denen die Agenten keinen Zugriff auf den globalen Zustand der Umgebung haben und Entscheidungen auf der Grundlage lokaler und unvollständiger Informationen treffen. Hier berücksichtigen Strategien üblicherweise nicht nur die aktuelle, sondern auch vorhergegangene Beobachtungen. Szenarien mit einem einzigen lernenden Agenten wie in Kapitel 3.5 werden als *Partially Observable Markov Decision Process* (POMDP) [69] formalisiert. Ein POMDP ist gegeben durch ein Tupel $M_{POMDP} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \Omega, b_0 \rangle$. Dabei ist $\mathcal{S}$ eine Menge aus Zuständen mit

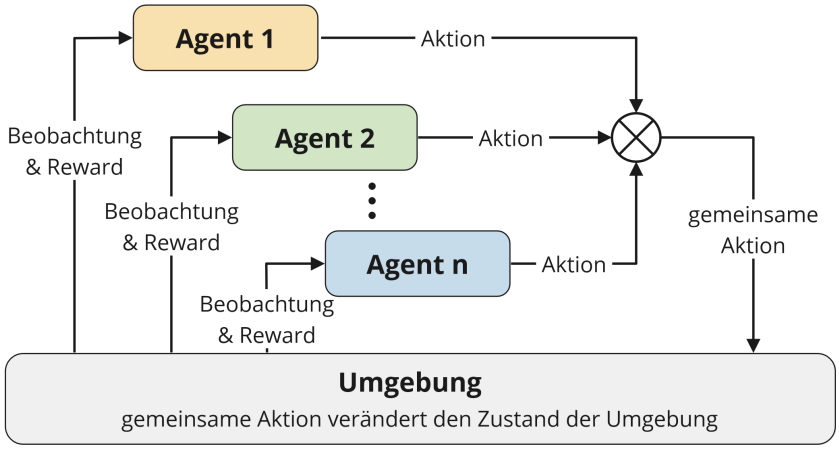


Abbildung 2.2: Schematische Darstellung des *Multi-Agent Reinforcement Learning*, einer Form des *Sequential Decision Making*, in einem *allgemeinen Szenario*. Hier erhalten n Agenten individuelle Beobachtungen der Umgebung und wählen basierend darauf Aktionen, die dann den Zustand der Umgebung verändern. Anschließend erhält jeder Agent einen individuellen Reward sowie eine aktualisierte Beobachtung und der Ablauf wiederholt sich. Darstellung und Beschreibung gemäß [4, S.6].

$s_t, s_{t+1} \in \mathcal{S}$ und $\mathcal{A}$ ist eine Menge aus Aktionen mit $a_t \in \mathcal{A}$. Die Funktion $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ definiert die Wahrscheinlichkeit, von einem Zustand s_t mit einer Aktion a_t den Folgezustand s_{t+1} zu erreichen und beschreibt dadurch die Dynamik der Umgebung. Des Weiteren ist $r_t = \mathcal{R}(s_t, a_t) \in \mathbb{R}$ der Reward und $\mathcal{Z}$ ist eine (endliche) Menge aus Beobachtungen mit $z_t \in \Omega$. Die Funktion $\Omega(s_t)$ definiert die Wahrscheinlichkeit, die Beobachtung z_t eines Zustands s_t zu machen. b_0 die Wahrscheinlichkeitsverteilung der Anfangszustände s_0. Die Stochastizität in den Anfangszuständen b_0 und Beobachtungen Ω resultiert aus der Unsicherheit über die wahren Zustände $\mathcal{S}$ der Umgebung. Um diese Unsicherheit zu verringern, wird in der Praxis eine *Historie* von Zuständen $\tau_t = \langle a_0, z_1, ..., z_{t-1}, a_{t-1}, z_t \rangle$ verwendet, die der lernende Agent vorhält. Formal ordnet eine Strategie $\pi(s \mid a)$ zu jedem Zeitpunkt t jeder möglichen Aktion $a \in A$ eines Zustands $s \in S$ eine Wahrscheinlichkeit zu und beschreibt dadurch das Verhalten des Agenten. Basierend darauf kann der *Wert* (englisch: value) V^π einer Strategie π berechnet werden, wobei eine optimale Strategie π^* zu optimalen Werten V^* führt.

Szenarien mit zwei oder mehr lernenden Agenten (siehe Abbildung 2.2), die diese Arbeit in den Kapiteln 3.6 und 4 untersucht, werden als *Partially Observable Stochastic Game* (POSG) [4, S.48] formalisiert. Ein POSG ist gegeben durch ein Tupel $M_{POSG} = \langle \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \Omega, b_0 \rangle$, in dem $\mathcal{D} = \{1, ..., n\}$ die Menge aller Agenten i ist. $\mathcal{S}$ ist die Menge globaler Zustände mit $s_t \in \mathcal{S}$ und $\mathcal{A} = \langle \mathcal{A}_i \rangle_{i \in \mathcal{D}}$ ist die Menge gemeinsamer Aktionen, zu der jeder Agent i zu

jedem Zeitpunkt t eine Aktion a_i beiträgt gemäß $a_t = \langle a_{t,1}, ..., a_{t,n} \rangle = \langle a_{t,i} \rangle_{i \in \mathcal{D}}$. Die Funktion $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ definiert die Wahrscheinlichkeit, von einem Zustand s_t mit einer gemeinsamen Aktion a_t den Folgezustand s_{t+1} zu erreichen. $r_t = \langle r_{t,i} \rangle_{i \in \mathcal{D}} = \mathcal{R}(s_t, a_t) \in \mathbb{R}$ ist der gemeinsame Reward und $\mathcal{Z}$ ist eine Menge lokaler Beobachtungen $z_{t,i}$ für die Agenten $i \in \mathcal{D}$. Die Funktion $\Omega(s_t) = z_t = \langle z_{t,i} \rangle_{i \in \mathcal{D}} \in \mathcal{Z}^n$ definiert die Wahrscheinlichkeit, die gemeinsame Beobachtung z_t eines Zustands s_t zu erhalten. Wie in einem POMDP ist b_0 die Wahrscheinlichkeitsverteilung der Anfangszustände s_0. Jeder Agent i verfügt über eine *lokale Historie* $\tau_{t,i} \in (\mathcal{Z} \times \mathcal{A}_i)^t$, die ein Teil der *gemeinsamen Historie* $\tau_t = \langle \tau_{t,i} \rangle_{i \in \mathcal{D}}$ ist. Die *gemeinsame Strategie* $\pi = \langle \pi_i \rangle_{i \in \mathcal{D}}$ der *individuellen Strategien* π_i ist die Wahrscheinlichkeit für die gemeinsame Aktionswahl $\pi(a_t \mid \tau_t) = \prod_{i \in \mathcal{D}} \pi_i(a_{t,i} \mid \tau_{t,i})$. Eine *optimale gemeinsame Strategie* $\pi^* = \langle \pi_i^* \rangle_{i \in \mathcal{D}}$ maximiert die Summe der individuellen Werte V_i^* aller Agenten $i \in \mathcal{D}$. Diese Summe wird als *Gemeinwohl* bezeichnet.

Im Kontext des POSG ist ein *Nash Equilibrium* eine gemeinsame Strategie $\pi^+ = \langle \pi_i^+ \rangle_{i \in \mathcal{D}}$, bei der kein Agent von seiner individuellen Strategie π_i^+ abweichen kann, ohne seinen individuellen Wert V_i^+ in seiner Historie τ_t zu verringern. Dabei wird angenommen, dass kein anderer Agent von seiner jeweiligen individuellen Strategie abweicht. Insgesamt ermöglicht dies die Beschreibung folgender Ausprägungen von MAS:

- **Kooperative Szenarien**, englisch *Cooperative Games*, sind Szenarien, in denen alle Agenten das gleiche Ziel, basierend auf dem gleichen Reward, optimieren. Formal ausgedrückt: $r_i = r_j$ für alle Agenten $i, j \in \mathcal{D}$. Hier ist eine optimale gemeinsame Strategie gleich einem Nash Equilibrium, formal $\pi^+ = \pi^*$ [21]. Probleme, die verteilt gelöst werden, beispielsweise die Fabrikautomatisierung [111, 121, 122], die in Kapitel 4 detailliert behandelt werden, können mit kooperativen Szenarien abgebildet werden.
- **Allgemeine Szenarien**, englisch *General-Sum Games* beziehungsweise *Mixed Cooperative-Competitive Scenarios*, sind weder rein kooperativ, noch rein kompetitiv. Hier optimieren Agenten individuelle Ziele basierend auf individuellen Rewards, die jedoch von den Aktionen der anderen Agenten abhängen können. Je nach Rewardstruktur können in allgemeinen Szenarien *soziale Dilemmas* auftreten [55, 78]. In sozialen Dilemmas ist eine gemeinsam optimale Strategie kein Nash Equilibrium, formal $\pi^+ \neq \pi^*$, und Nash Equilibria sind hinsichtlich des Gemeinwohls nicht optimal, formal $V_i^+ \leq V_i^*$ für alle Agenten $i \in \mathcal{D}$. Allgemeine Szenarien eignen sich beispielsweise, um die Interaktion autonomer Agenten mit individuellen Zielen in einer gemeinsamen Umgebung abzubilden [120, 124], was in Kapitel 3 detailliert behandelt wird.
- **Nullsummenspiele**, englisch *Zero-Sum Games*, sind Szenarien, in denen $n = 2$ Agenten entgegengesetzte Ziele basierend auf entgegengesetzten Rewards optimieren, sodass $r_1 + r_2 = 0$ gilt. Nullsummenspiele eignen

sich beispielsweise zur Modellierung von Testfällen für die Systemvalidierung oder zur Realisierung von robusten RL-Techniken [42]. Eine Variante ist, dass ein testender Agent (Antagonist) Probleminstanzen (Tests) für einen oder mehrere zu testende Agenten (Protagonisten) erzeugt. Durch die Formulierung als Nullsummenspiel ist das Ziel des Antagonisten automatisch, die Probleminstanz zu finden, die für die Protagonisten am schwersten zu lösen ist (Worst-Case). Ziel der Protagonisten ist weiterhin, die gegebene Probleminstanz möglichst gut zu lösen. Dadurch entspricht ein Nash Equilibrium für jeden Agenten der Strategie π_i, die im Worst-Case optimal ist, formal $V_i^+ = max_{\pi_i} min_{\pi_j} V_i^\pi$ [109]. Einige bekannte Anwendungen des RL basieren auf *Self-Play* in Nullsummenspielen [137, 149]. In dieser Arbeit werden jedoch nur kooperative und allgemeine Szenarien betrachtet.

2.2 Lernende Multi-Agenten-Systeme

Dieses Unterkapitel beginnt mit dem Szenario eines einzelnen lernenden Agenten in einem POMDP und beschreibt damit in den Unterkapiteln 2.2.1 sowie 2.2.2 zwei grundlegende RL-Ansätze sowie die entsprechenden in dieser Arbeit verwendeten Algorithmen. Anschließend erweitert Unterkapitel 2.2.3 das Szenario auf mehrere lernende Agenten in einem POSG. Auch hier werden Herangehensweisen und die entsprechenden in dieser Arbeit verwendeten Algorithmen beschrieben.

Ziel des RL ist eine Strategie π, die den Erwartungswert der diskontierten Summe der Rewards innerhalb eines Horizonts T maximiert [145, S.57]:

$$G_t = \sum_{k=0}^{T-1} \gamma^k \, \mathcal{R}(s_{t+k}, a_{t+k}) = \sum_{k=0}^{T-1} \gamma^k \, r_{t+k} \tag{2.1}$$

G_t wird als *Ertrag* (englisch: return) bezeichnet und $\gamma \in [0, 1]$ als *Diskontierungsfaktor* (englisch: discount factor). Je kleiner γ ist, desto stärker werden aktuelle gegenüber künftigen Rewards gewichtet. Zur Suche und Auswahl einer geeigneten Strategie müssen diese bewertet und verglichen werden können. Dies geht einerseits a posteriori über den beobachteten Return G_t. Alternativ kann der Wert V^π gegeben einem (Ausgangs-) Zustand s_t über den Erwartungswert $\mathbb{E}$ des (künftigen) Ertrags berechnet werden [145, S.59]:

$$
\begin{aligned}
V^{\pi}(s_t) &= \mathbb{E}_{\pi,P}\big[G_t \mid s_t\big] && (2.2)\\
&= \mathbb{E}_{\pi,P}\big[\, r_{t+1} + \gamma\, G_{t+1} \mid s_t \,\big] \\
&= \sum_{a_t} \pi(a_t \mid s_t) \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \Big[r_t + \gamma\, \mathbb{E}_{\pi,P}\big[G_{t+1} \mid s_{t+1}\big] \Big] \\
&= \sum_{a_t} \pi(a_t \mid s_t) \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \big[r_t + \gamma\, V^{\pi}(s_{t+1}) \big] && (2.3)
\end{aligned}
$$

Gleichung 2.3 wird als *Bellman Gleichung* von V^{π} bezeichnet und beschreibt, wie der Wert eines Zustands (rekursiv) über den unmittelbaren Reward und den Wert des Folgezustands definiert ist. Aufgrund der partiell beobachtbaren Umgebung verwendet diese Arbeit abweichend von Sutton et al. [145] die POMDP Notation, in der r_t kein Teil von $\mathcal{P}\left(s_{t+1} \mid s_t, a_t\right)$ ist [69]. Hinsichtlich der Suche und Auswahl einer geeigneten Strategie kann nun formuliert werden, dass eine Strategie dann optimal ist, wenn es keine andere Strategie gibt, die einen größeren Wert liefert, formal $V^{*}(s_t) \geq V^{\pi'}(s_t)$ für alle $s_t \in S$ und $\pi' \neq \pi^{*}$.

Ein erstes Problem ist, dass es in einem POMDP zur Laufzeit keinen (direkten) Zugriff auf die Zustände $s \in \mathcal{S}$ gibt, sondern nur die Historie τ verwendet werden kann [69]:

$$
V^{\pi}(\tau_t) = \sum_{s_t \in \mathcal{S}} V^{\pi}(s_t)\, b\left(s_t \mid \tau_t\right) \tag{2.4}
$$

Der *belief state* $b\left(s_t \mid \tau_t\right)$ ist die Wahrscheinlichkeit, dass sich die Umgebung gegeben einer Historie τ_t im Zustand s_t befindet und kann gemäß Bayes' Regel berechnet oder approximiert werden [125]. In der Praxis erschwert die Unsicherheit über den tatsächlichen Zustand der Umgebung den Umgang mit dem *Exploration-Exploitation Dilemma*. Agenten müssen sowohl so handeln, dass sie Informationen über das Problem und dessen Dynamik sammeln (Exploration), als auch so, dass sie mit gesammelten Informationen das Ziel erreichen (Exploitation). Zu wenig Exploration kann dazu führen, dass das Problem gar nicht oder nicht optimal gelöst werden kann. Jedes Abweichen von der besten bekannten Strategie zum Sammeln weiterer Informationen kann dazu führen, dass das Problem schlechter gelöst wird. Beides muss unter Berücksichtigung von (Vor-)Wissen und Problemdynamik gewichtet werden. In Kapitel 3.5 ist ein zweistufiger Trainingsprozess notwendig, um das Exploration-Exploitation Dilemma zu lösen. Ansonsten kommen in dieser Arbeit die regulären Mechanismen der jeweiligen Algorithmen zum Einsatz, da sie sich in den übrigen Szenarien als ausreichend erwiesen haben.

Ein zweites Problem ist, dass die exakte Berechnung von V^{π} nur möglich ist, wenn ein Modell M_{POMDP} zur Verfügung steht, dass P vollständig abbildet. Um

darauf verzichten zu können, verwendet diese Arbeit *Model-free RL*, was bedeutet, dass π^* auf Basis von Erfahrungen $e_t = \langle \tau_t, a_t, r_t, z_{t+1} \rangle$ mittels *Funktionsapproximation* durch Techniken des *Deep Learning* (DL) angenähert wird [96, 98], beispielsweise einem *Neural Network* (NN). Dafür gibt es zwei Vorgehensweisen, die in den folgenden Kapiteln 2.2.1 und 2.2.2 erläutert werden.

2.2.1 Value-based Reinforcement Learning

Die Vorgehensweise von *wertbasierten* (englisch: value-based) Ansätzen ist, die Strategie aus dem Wert eines Zustands abzuleiten. Üblicherweise wird dafür die *Q-Funktion* genutzt [145, S.58]:

$$Q^\pi(\tau_t, a_t) = \mathbb{E}_{\pi,P,b}\left[G_t \mid \tau_t, a_t\right] \tag{2.5}$$

Analog zu V^π ist die exakte Berechnung von Q^π in einem POMDP ohne ein Modell M_{POMDP} nicht möglich. Eine Alternative ist, Q^π im Rahmen des *Q-Learning* iterativ mit dem *Temporal Difference* Verfahren anzunähern [154]:

$$Q^\pi(\tau_t, a_t) \leftarrow Q^\pi(\tau_t, a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1}} Q^\pi(\tau_{t+1}, a_t) - Q^\pi(\tau_t, a_t) \right] \tag{2.6}$$

Dabei ist α die Lernrate und τ_{t+1} die um a_t und z_{t+1} ergänzte Historie τ_t. Durch die Verwendung des *max*-Operators nähert sich Q^π schrittweise einer optimalen Q-Funktion Q^*. Umgekehrt kann π^* aus Q^* abgeleitet werden, indem stets die Aktion mit dem höchsten Q-Wert gewählt wird:

$$\pi^*(\tau_t) = \max_{a_t} Q^*(\tau_t, a_t) \tag{2.7}$$

In den Kapiteln 3 und 4 kommt der Algorithmus *Deep-Q-Networks* (DQN) [98] zum Einsatz, der Q^π mit einem NN $\hat{Q}^\psi$ approximiert und dieses gemäß dem Temporal Difference Verfahren schrittweise an Q^* annähert. Zwar gibt es beim Einsatz solcher nichtlinearer Funktionsapproximatoren keine Konvergenzgarantie. Aber eine exakte Darstellung aller Q-Werte, beispielsweise mittels einer Tabelle, ist in sehr großen Zustandsräumen $\mathcal{S}$ und Aktionsräumen $\mathcal{A}$ ansonsten nicht praktikabel [98, 145]. Ziel des Trainings ist, mit den vom Agenten gesammelten Daten die Präzision von $\hat{Q}^\psi$ zu verbessern, da die Wahl der optimalen Aktion gemäß dem *max*-Operator eine korrekten Schätzung der Q-Werte erfordert. Dazu werden Parameter ψ dieses NN $\hat{Q}^\psi$ so angepasst, dass sie die folgende Fehlerfunktion (englisch: loss) $L(\psi)$ minimieren:

$$L(\psi) = \mathbb{E}_{e_t \sim Exp}\left[\left(r_t + \gamma \max_{a_{t+1}} \hat{Q}^{\psi^-}(\tau_{t+1}, a_{t+1}) - \hat{Q}^{\psi}(\tau_t, a_t)\right)^2\right] \tag{2.8}$$

Hier berücksichtigt diese Arbeit zwei von Mnih et al. [98] vorgeschlagene, optionale Verbesserungen. Erstens werden die Erfahrungen e_t zufällig aus einem *Replay Buffer Exp* gezogen, um die Auswirkung der Korrelation zwischen den gesammelten Erfahrungen zu verringern. Zweitens stabilisiert ein zweites NN, genannt *Target*, die Schätzung der Q-Funktion. Dessen Parameter ψ^- sind eine ältere Iteration von ψ. Die Änderungen in ψ werden alle c Schritte auf ψ^- übertragen. In der Praxis funktioniert dieses Vorgehen bei der Approximation von Q^π so gut, dass beispielsweise eine frühe Version von DQN [97] der erste RL-basierte Ansatz war, der auf mehreren Atari-Spielen ohne spielspezifisches Feature-Engineering das Niveau von Menschen erreichte.

2.2.2 Policy-based Reinforcement Learning

Die Vorgehensweise von *strategiebasierten* (englisch: policy-based) Ansätzen ist, eine optimale Strategie π^* direkt zu approximieren. Dazu wird eine Strategie π_θ mit den Parametern θ betrachtet, beispielsweise den Gewichten eines NN, deren stochastisches Verhalten (Aktionswahl) durch $\pi_\theta(a_t \mid \tau_t)$ beschrieben wird. Als Ausgangspunkt für die Approximation ist ein Maß für die Leistung der Strategie π_θ notwendig. Davon ausgehend, dass es wie zuvor kein Modell der Umgebung gibt, kann V^{π_θ} nicht exakt berechnet und als Maß verwendet werden. Eine Alternative ist $J(\theta)$, das, ausgehend von einer (initialen) Zustandsverteilung, die Leistung der Strategie über den erwarteten Reward ausdrückt und in einem numerischen Wert resultiert [144]:

$$J(\theta) = \sum_{\tau_t \in (\mathcal{Z}\times\mathcal{A})^t} d^{\pi_\theta}(\tau_t) \sum_{a_t \in \mathcal{A}} \pi_\theta(a_t \mid \tau_t)\ \mathcal{R}_{a_t}^{\tau_t} \tag{2.9}$$

Hier ist $d^{\pi_\theta}(\tau_t) = \lim_{t\to\infty} P(\tau_t \mid b_0, \pi_\theta) \in [0,1]$ die stationäre Verteilung der Historie τ_t unter π_θ und $\mathcal{R}_{a_t}^{\tau_t} = \mathbb{E}[r_{t+1} \mid \tau_t, a_t] \in \mathbb{R}$ ist der erwartete Reward. Um mit π_θ die optimale Strategie π^* zu approximieren, muss $J(\theta)$ maximiert werden. Vorausgesetzt, dass $J(\theta)$ differenzierbar ist, kann dazu ein Anstiegsverfahren auf θ in positiver Richtung des Gradienten ∇ durchgeführt werden [131]:

$$\nabla J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid \tau_t)\right] \tag{2.10}$$

Hier ist der rechte Term $\nabla_\theta \log \pi_\theta(a_t \mid \tau_t)$ eine Schätzung des Gradienten. Der linke Term Ψ_t definiert die Richtung des Gradienten. Im Sinne des Anstiegsverfahrens repräsentiert Ψ_t den *Vorteil* (englisch: advantage) von Veränderungen

der Strategie, der bei ungünstigen Veränderungen auch negativ sein kann. Eine Variante ist, den Ertrag G_t aus Gleichung 2.1 als Ψ_t zu verwenden [155]. Dies hat eine hohe Varianz zu Folge. Eine andere Variante ist, eine *Actor-Critic* Architektur zu verwenden. Diese besteht aus zwei Komponenten: dem Actor, der die Strategie π_θ repräsentiert und so optimiert, dass sie Ψ_t maximiert sowie dem *Critic*, der V^{π_θ} repräsentiert. Mit Hilfe des Critic kann das folgende *TD-Residual* berechnet und als Ψ_t verwendet werden [49]:

$$r_t + \gamma\, V^{\pi_\theta}(\tau_{t+1}) - V^{\pi_\theta}(\tau_t) \tag{2.11}$$

Diese Variante weist eine geringe Varianz auf. In der Praxis wird die Wertfunktion meist approximiert, was einen hohen Bias zur Folge hat. *Generalized Advantage Estimation* [131] kombiniert beide zuvor geschilderten Varianten und approximiert den Vorteil, indem der Parameter λ die Gewichtung von Bias und Varianz mittels exponentiell gleitendem Mittelwert steuert:

$$\begin{aligned} \delta_t &= r_t + \gamma\, \hat{V}(\tau_{t+1}) - \hat{V}(\tau_t) \\ \hat{A}_t &= \sum_{k=0}^{T-1} (\lambda\,\gamma)^k\, \delta_{t+k} \end{aligned} \tag{2.12}$$

Hier ist $\hat{V}$ eine Approximation von V^{π_θ} durch den Critic. Der Algorithmus *Proximal Policy Optimization* (PPO) [132] verwendet $\hat{A}_t$ als Ψ_t und begrenzt die maximale Veränderung von θ pro Optimierungsschritt, um ein robustes Lernverhalten zu erreichen. Ausgehend von einer Historie τ_t vergleicht PPO dazu mittels der Funktion $\rho_t(\theta)$ zunächst die Wahrscheinlichkeiten, eine Aktion a_t zu wählen, vor einer Änderung $\pi_{\theta_{old}}$ und nach einer Änderung π_θ :

$$\rho_t(\theta) = \frac{\pi_\theta\,(a_t \mid \tau_t)}{\pi_{\theta_{old}}\,(a_t \mid \tau_t)} \tag{2.13}$$

Bei $0 < \rho_t(\theta) < 1$ war die Aktion mit der vorherigen Strategie wahrscheinlicher, bei $\rho_t(\theta) > 1$ ist sie mit der aktuellen Strategie wahrscheinlicher. $\rho_t(\theta)$ kommt in der folgenden, zu minimierenden Fehlerfunktion L^{CLIP} zum Einsatz:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[min\Big(\rho_t(\theta)\,\hat{A}_t,\ clip\big[\rho_t(\theta),\ 1-\epsilon,\ 1+\epsilon\big]\,\hat{A}_t\Big)\right] \tag{2.14}$$

Einerseits wird extremen Veränderungen der Strategie entgegengewirkt, indem $\rho_t(\theta)$ im rechten Term des *min*-Operators auf einen Bereich $[1-\epsilon, 1+\epsilon]$ beschränkt wird. Andererseits bietet die unbeschränkte Verwendung von $\rho_t(\theta)$

im linken Term des *min*-Operators eine Art Absicherung: Bei negativem $\hat{A}_t$, das auf eine Verschlechterung hindeutet, kombiniert mit einer starken Veränderung der Strategie, die in einem Clipping $\rho_t(\theta) > 1 + \epsilon$ resultiert, wird das Clipping ignoriert, da der linke Term $\rho_t(\theta)$ kleiner ist. Das resultiert in einem stärkeren Gradienten in Richtung der ursprünglichen, mutmaßlich besseren Strategie. PPO kommt in Kapitel 3.6 zum Einsatz. Zusätzlich zur *Generalized Advantage Estimation* und zu L^{CLIP} verwendet die in dieser Arbeit verwendete Implementierung einen Replay-Buffer. Wie bei DQN (siehe Unterkapitel 2.2.1) verringert das die Korrelation der gesammelten Erfahrungen e_t. Anders als rein wertbasierte Ansätze konvergieren Policy-Gradient-Methoden sowohl unter Verwendung linearer [144], als auch nichtlinearer Funktionsapproximatoren wie NN garantiert zu lokalen Optima, jedoch muss in letzterem Fall sichergestellt sein, dass dabei genügend Zustände exploriert werden [96]. Die Konvergenz zu einem globalen Optimum konnte noch nicht uneingeschränkt gezeigt werden [151]. RL mit Policy-Gradient-Methoden hat zuletzt bemerkenswerte Leistungen erzielt, beispielsweise in den Brettspielen Go, Schach und Shogi [137], im Multiplayer-Videospiel Starcraft II [149], und bei der Feinjustierung des Sprachmodells GPT-4 [23], die alle sehr große Zustandsräume $\mathcal{S}$ und Aktionsräume $\mathcal{A}$ haben.

2.2.3 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) ist die Suche nach Strategien für mehrere Agenten, um abhängig vom Szenario den individuellen Ertrag einzelner Agenten oder das Gemeinwohl zu maximieren. Wie zuvor können dafür wert- oder strategiebasierte Ansätze genutzt werden. Nachfolgend wird die POSG-Notation verwendet, in der τ_{t+1} die *gemeinsame Historie* aller Agenten $i \in D$ zum Zeitpunkt $t+1$ ist, die aus $\langle \tau_t, a_t, z_{t+1} \rangle$ besteht. Dabei sind a_t die *gemeinsame Aktion* und τ_t die *gemeinsame Historie* aller Agenten zum Zeitpunkt t, und z_{t+1} ist die *gemeinsame Beobachtung* des nachfolgenden Zustands, die wiederum durch $\Omega(s_{t+1})$ definiert ist. b ist die Wahrscheinlichkeit, dass sich die Umgebung gegeben einer Historie τ_t im Zustand s_t befindet [69, 125]. Damit kann der Wert individueller Strategien über die folgenden Funktionen V_i^π und Q_i^π definiert werden [23, S.64]:

$$V_i^\pi(\tau_t) = \sum_{a_t \in \mathcal{A}} \pi(a_t \mid \tau_t)\, Q_i^\pi(\tau_t, a_t) \tag{2.15}$$

$$Q_i^\pi(\tau_t, a_t) = \sum_{s_t \in \mathcal{S}} b(s_t \mid \tau_t) \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} \mid s_t, a_t) \left[r_{t,i} + \gamma\, V_i^\pi(\tau_{t+1}) \right] \tag{2.16}$$

$$= \sum_{s_t \in \mathcal{S}} b(s_t \mid \tau_t) \left[r_{t,i} + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} \mid s_t, a_t)\, V_i^\pi(\tau_{t+1}) \right] \tag{2.17}$$

Die Formulierung von Q_i^π in Gleichung 2.16 orientiert sich an der Bellman

Gleichung 2.3. In der Praxis wird häufig die äquivalente [4, S.39] Formulierung der Gleichung 2.17 verwendet.

V_i^π ist hier der Wert für den Agenten i, wenn alle Agenten D gegeben der Historie τ der gemeinsamen Strategie π folgen. Analog dazu ist Q_i^π der Wert für den Agenten i, wenn alle Agenten gegeben der Historie τ die gemeinsame Aktion a ausführen und dann der gemeinsamen Strategie π folgen. Wie zuvor im POMDP verwendet diese Arbeit kein Modell der Umgebung M_{POSG}, um V_i^π und Q_i^π exakt zu berechnen. Stattdessen wird π_i^* iterativ über Erfahrung approximiert, die bei der Interaktion des Agenten $i \in D$ mit einer Umgebung gesammelt wird, in der er sich zusammen mit allen anderen Agenten befindet.

In *allgemeinen Szenarien* nimmt diese Arbeit an, dass alle Agenten separate autonome Systeme mit unterschiedlichen Zielen repräsentieren [112]. Dadurch können die Strategien π_i unabhängig voneinander mit Techniken des Single-Agent RL trainiert werden. Dieses Paradigma heißt *Independent Learning* (IL) und ist beispielsweise für modulare Systeme relevant, bei denen jedes Modul unabhängig von den anderen trainiert, aber in einer gemeinsamen Umgebung eingesetzt wird. IL skaliert gut, da Agenten parallel lernen können. Allerdings kann die daraus resultierende Nichtstationarität das Erlernen von Koordination und Kooperation erschweren, und es gibt keine Konvergenzgarantie. Ungeachtet dessen kann IL solide Ergebnisse in gängigen MARL Benchmarks erzielen [158].

Eine Variante des IL ist *Parameter Sharing* (PS), bei dem sich alle Agenten die Parameter des Funktionsapproximators (beispielsweise die Gewichte eines NN) teilen, der die Strategie repräsentiert. Da alle Agenten ihre Erfahrung in das Training dieser Strategie einbringen, skaliert PS hinsichtlich Sample-Effizienz sehr gut mit der Anzahl der Agenten. Jedoch eignet sich PS vor allem für homogene Agenten, denn die Approximation einer gemeinsamen Strategie wirkt insbesondere bei einer begrenzten Anzahl Parameter einer Spezialisierung unterschiedlich ausgeprägter Agenten entgegen. Diese Arbeit verwendet IL mit und ohne PS in den Kapiteln 3 und 4.

In *kooperativen Szenarien* nimmt diese Arbeit an, dass Agenten zwar separate autonome Systeme sind, aber dasselbe Ziel verfolgen. Aus der Umgebung gibt es daher häufig kein individuelles, sondern nur ein globales Feedback für alle Agenten, beispielsweise in Form eines globalen Rewards bei Erreichen des Ziels. Daraus folgt die Herausforderung, im Rahmen des *Credit Assignment* den individuellen Beitrag der Agenten zu diesem globalen Reward abzuleiten, um basierend darauf die individuellen Strategien der Agenten anpassen zu können. Das *Centralized Training for Dezentralized Execution* (CTDE) Paradigma basiert auf der Annahme, dass das Training aller Agenten in einer Simulation oder kontrollierten Umgebung stattfindet, in der Zugang zu globalen Informationen besteht, um $\hat{Q}_i \approx Q_i^\pi$ oder $\hat{V}_i \approx V_i^\pi$ gemeinsam an zentraler Stelle zu lernen [111, 117, 142]. Basierend darauf werden dann individuelle, dezentrale Strategie-Approximatoren $\hat{\pi}_i$ mit $\hat{Q}_i$ oder $\hat{V}_i$ trainiert, z.B. über Actor-Critic

Ansätze oder eine Faktorisierung der (zentralen) Wertfunktion. Nach dem Training können die Agenten $\hat{\pi}_i$ ohne $\hat{Q}_i$ beziehungsweise $\hat{V}_i$ verwenden. CTDE eignet sich nicht nur für kooperative Szenarien, sondern auch für allgemeine Szenarien, in denen eine gemeinsame Wertfunktion für eine Gruppe von Agenten realisiert werden kann und soll. Unterkapitel 3.3.3 beschreibt Ansätze, die Selbstorganisation basierend auf dem CTDE Paradigma realisieren.

Value-Decomposition Networks (VDN) [142] setzt CTDE um, indem eine gemeinsame Q-Funktion $Q_{tot}(\tau, a)$ in Form der Summe der approximierten, individuellen Q-Werte $\hat{Q}_i(\tau_i, a_i)$ aller Agenten $i \in D$ gelernt wird:

$$Q_{tot}(\tau, a) = \sum_{i=1}^{n} \hat{Q}_i(\tau_i, a_i) \tag{2.18}$$

Dies wird realisiert, indem die $L_(\psi)$ Gleichung 2.8 minimiert wird, wobei statt der Approximation eines einzelnen Q-Wertes $\hat{Q}(\tau, a)$ nun die Summe aller (approximierten) Q-Werte $Q_{tot}(\tau, a)$ eingesetzt wird. Wenn alle Agenten (abseits der Exploration) gemäß dem *max*-Operator stets die Aktion mit dem höchsten Wert ihrer individuellen Q-Funktionen wählen, führt das zur selben gemeinsamen Aktion als würde der *argmax*-Operator auf der gemeinsamen Q-Funktion angewendet:

$$\underset{a}{argmax}\, Q_{tot}(\tau, a) = \begin{pmatrix} \underset{a_1}{argmax}\, \hat{Q}_1(\tau_1, a_1) \\ \vdots \\ \underset{a_n}{argmax}\, \hat{Q}_n(\tau_n, a_n) \end{pmatrix} \tag{2.19}$$

Damit erfüllt die Summe die *Individual-Global-Max*-Eigenschaft [139], die bei der Faktorisierung als Voraussetzung für das erfolgreiche Erlernen einer gemeinsamen Strategie gesehen wird. VDN kommt in Kapitel 4 zum Einsatz. Um die Individual-Global-Max-Eigenschaft aus Gleichung 2.19 zu erfüllen, kann an Stelle der Summe ein beliebiger Operator verwendet werden, so lange Monotonie zwischen Q_{tot} und allen Q_i gewährleistet ist:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i \in D \tag{2.20}$$

Monotonic Value Function Factorisation (QMIX) [117] verwendet an Stelle der Summe ein *Mixing Network*, das Q_{tot} auf Basis aller Q_i und des globalen Zustands s_t approximiert. Die Monotonie gemäß Gleichung 2.20 wird erzwungen, indem die Gewichte des Mixing Networks nur positive Werte annehmen können. Es gibt verschiedene Varianten und Erweiterungen von QMIX, beispielsweise die Verwendung von Historien $\tau_{i,t}$ an Stelle von s_t unter Verwendung einer Transformer-Architektur [110]. Ungeachtet dessen ist das Faktorisieren

der zentralen Wertfunktion mit einem Mixing Network zum Lösen des Assignments derzeit der State-of-the-Art im Bereich CTDE. QMIX kommt in Kapitel 4 zum Einsatz.

2.3 (Selbst-)Organisation in Multi-Agenten-Systemen

Offene, unkontrollierte Umgebungen stellen nicht nur künstliche Agenten vor Herausforderungen. In der Natur hat sich daher der Zusammenschluss von Individuen zu Gruppen an unterschiedlichen Stellen evolutionär durchgesetzt. Viele Fischarten bilden Schwärme, um als Kollektiv die Überlebenschancen gegenüber Raubtieren zu erhöhen [102]. Wanderameisen formen mit ihren Körpern gerüstartige Strukturen, um Artgenossen eine schnelle Fortbewegung über Hindernisse zu ermöglichen [90]. Auch Menschen organisieren sich, teils spontan, auf unterschiedliche Arten: auf lokaler Ebene beispielsweise durch gegenseitige Unterstützung im Freundes- und Familienkreis, auf globaler Ebene beispielsweise durch das Formen einer Gesellschaft, in der Arbeitsteilung und Spezialisierung Leistungen ermöglichen, zu denen kein Individuum alleine im Stande wäre. Viele Aspekte dieser Beispiele wurden auf künstliche MAS übertragen. Gemäß Boissier et al. [17] gibt es zwei Sichtweisen, was unter *Organisation* in MAS zu verstehen ist:

1. Eine kollektive Einheit mit einer Identität, die durch eine Gruppe von Agenten repräsentiert wird (aber nicht mit dieser identisch ist) und relativ stark formalisierte soziale Strukturen aufweist [133].
2. Ein stabiles Muster oder eine stabile Struktur gemeinsamer Aktivitäten, die die Handlungen und Interaktionen der Agenten auf einen bestimmten Zweck hin einschränken oder beeinflussen können [26].

Der Begriff *Organisation* bezieht sich hier auf ein Kooperationsmuster, das unterschiedlich stark formalisiert sein kann und beispielsweise Ausdruck einer Aufgaben-, einer Rollenverteilung oder eines Autoritätssystems ist. Diese Arbeit nimmt die zweite, *Agent-zentrierte* Sichtweise ein, die Picard et al. [113] folgendermaßen beschreiben: Organisation entsteht als beobachtbares, emergentes Phänomen und gewährt eine Bottom-up-Sicht auf das Muster der Zusammenarbeit zwischen den Agenten. Die Agenten organisieren sich durch indirekte Kommunikation über die Umgebung. Ihnen ist nicht bewusst, dass sie Teil der Organisation sind, und sie können nicht explizit über die Organisation nachdenken, weil diese nicht modelliert ist. Dennoch kann ein Beobachter die Organisation auf globaler Ebene entlang verschiedener Dimensionen erkennen, beispielsweise über die (räumliche) Anordnung der Agenten oder ihre Spezialisierung. Die Organisation variiert zu verschiedenen Zeitpunkten des Lebenszyklus des Gesamtsystems, wobei die Ursache entweder eine Veränderung des Verhaltens der Agenten oder eine Veränderung des Szenarios sein kann.

Eine technische Umsetzung der Agent-zentrierten Organisation sind beispielsweise schwarmbasierte Systeme [118]. Ähnlich wie bei Fischschwärmen [102] und Ameisenvölkern [90] in der Natur ist die Organisation selbst nicht als Zwang oder Regel innerhalb der Agenten definiert. Stattdessen ist Organisation das Ergebnis des kollektiven Verhaltens, das sich aus der Art und Weise ergibt, wie sich die einzelnen Agenten verhalten und in einer gemeinsamen, dynamischen Umgebung (miteinander) interagieren. [113]

Wenn die Regeln für die Interaktion der Agenten nicht vorgegeben sind, sondern situativ durch die Agenten gewählt oder sogar erlernt werden, spricht man von *Selbstorganisation*. Hier stützt sich diese Arbeit auf die Definition von Serugendo et al. [35], nach der Selbstorganisation

- der Mechanismus ist, der es einem System ermöglicht, seine Organisation ohne expliziten externen Befehl während seiner Ausführungszeit zu ändern.
- dann stark ist, wenn es in einem System keine explizite zentrale Kontrolle gibt, weder intern noch extern.
- die Ursache für emergentes Verhalten ist, das entweder erwünscht oder unerwünscht sein kann.

Serugendo et al. definieren emergente Phänomene in diesem Kontext als Ergebnis lokaler Interaktionen, die sich nicht exakt aus der Untersuchung individueller Verhaltensweisen ableiten, sondern nur global beobachten lassen. Es gibt keine (zentrale) Instanz, die alle Agenten des Systems kontrolliert. Stattdessen sind die Agenten autonom und bestimmen ihr Verhalten selbst. Insbesondere ist ein selbstorganisiserendes System in der Lage, die emergente Struktur selbständig zu ändern, was die Fähigkeit impliziert, dass es sich an dynamische Veränderungen anpassen kann. Im Raum der Verhaltensweisen der Agenten existieren meist mehrere Gleichgewichte, wenn mehrere Attraktoren für stabile Zustände vorhanden sind. Einen maßgeblichen Einfluss darauf hat die Funktion beziehungsweise das globale Ziel, das das System mit Hilfe der Selbstorganisation zu maximieren versucht. [35]

Das *Adaptive Multi-Agent Systems* (AMAS) Konzept [47] basiert auf Selbstorganisation in MAS. Hier verfolgen Agenten lokale Ziele, während sie versuchen, kooperative Beziehungen mit anderen in das System eingebetteten Agenten zu unterhalten. Die Agenten sind kooperativ in dem Sinne, dass sie nicht-kooperative Situationen vorhersehen müssen, um sie zu vermeiden, und wenn nicht-kooperative Situationen eintreten, müssen die Agenten handeln, um zu kooperativen Interaktionen zurückzukehren. Aus der Sicht des Entwicklers kann das Auffinden von nicht-kooperativen Situationen in einem MAS und deren Beseitigung als eine Ausnahmeregelung in klassischen Programmen betrachtet werden.

Diese Arbeit nimmt die Agent-zentrierte Sichtweise ein und untersucht Selbstorganisation als emergenten Effekt in MAS. Jedoch wird von der Annahme

abgewichen, dass Agenten auf individueller Ebene grundsätzlich Kooperation anstreben. Stattdessen wird davon ausgegangen, dass eigennützige Agenten gegebenenfalls das eigene Wohl über das der Gruppe stellen, wenn das eigene Ziel dadurch schneller erreicht werden kann. Solche Situationen können eintreten, wenn heterogene Agenten, beispielsweise unterschiedlichen Typs oder Herstellers, aufeinander treffen. In der Spieltheorie gibt es Probleme wie das wiederholte Gefangenendilemma, in denen Kooperation für die Individuen keine unmittelbar *rationale* Strategie [8] darstellt, aber das Gemeinwohl langfristig verbessert. Es ist jedoch erstrebenswert, dass autonome Agenten sich grundsätzlich so verhalten, dass langfristig das Gemeinwohl maximiert wird. Die Herausforderung ist, dass die Ursachen für *emergente* Kooperation und Altruismus für den allgemeinen Fall nicht geklärt sind [30, 103]. Diese wird mit Hilfe lernender Agenten ausführlicher in Kapitel 3 untersucht.

2.4 Ausrichtung im Maschinellen Lernen

Vorausgesetzt, dass lernende Agenten, wie zuvor dargelegt, die Form der Interaktion selbst optimieren können, muss sichergestellt werden, dass sie dabei kein unerwünschtes oder gefährliches Verhalten erlernen. Dies kann am Beispiel einer industriellen Lot-Size One Produktion wie im späteren Kapitel 4 verdeutlicht werden: Haben Agenten wenige Freiheitsgrade, ist es praktikabel, diese durch zusätzliche Komponenten in oder Schichten um die Agenten abzusichern. Beispielsweise können Absperrungen verhindern, dass Menschen in den Arbeitsbereich stationärer Maschinen gelangen, und unvorhergesehene Probleme können durch sofortige Abschaltung der Maschinen und manuelles Eingreifen gelöst werden. Für Systeme mit vielen Freiheitsgraden benötigt es andere Vorgehensweisen. Beispielsweise ist es nicht praktikabel, für jedes FTF einen eigenen, isolierten Arbeitsbereich zu schaffen. Folglich begegnen FTF sich und anderen Akteuren, beispielsweise Menschen, und müssen dabei sicher agieren, beispielsweise angemessen ausweichen oder abbremsen, um einen definierten Abstand einzuhalten. Abruptes Einlenken oder Notbremsungen sind nach Möglichkeit zu vermeiden, um unnötigen Verschleiß des FTF und Schäden an den zu transportierenden Gegenständen vorzubeugen. Gegenüber dem Einsatz autonomer Fahrzeuge im Straßenverkehr stellt der Einsatz von FTF in der industriellen Produktion ein Szenario dar, dass vergleichsweise gut kontrollierbar ist: Äußere Einflüsse wie die Witterung sowie das Verhalten anderer Akteure sind weitestgehend vorhersehbar. Ungeachtet dessen müssen autonome Agenten die in ihrem Einsatzbereich geltenden Regeln einhalten. Solche Regeln werden üblicherweise in den nicht-funktionalen Anforderungen berücksichtigt, die zusammen mit den funktionalen Anforderungen das gewünschte Verhalten eines Systems beschreiben.

Die Herausforderungen, die insbesondere im Umfeld lernender Systeme hinsichtlich der Einhaltung solcher *Spezifikation* entstehen, fallen in den Bereich

des *Alignment* (Ausrichtung) [43, 59, 79, 128]. Dieser Bereich ist breit gefächert und die Interpretation der Begriffe variiert. Eine abstrakte Sicht auf diesen Bereich schildert das *AI Value Alignment* [128]. Ziel ist hier das korrekte Ausrichten „mächtiger“ künstlicher Intelligenz, beispielsweise hochentwickelter autonomer Agenten, an menschlichen Werten. Basierend darauf teilt Iason Gabriel [43] die Herausforderungen bei der Realisierung in zwei Kategorien.

Die erste Kategorie umfasst technische Herausforderungen. Hier geht es um die Frage, *wie* Agenten Werte oder Prinzipien vermittelt werden, damit sie zuverlässig die vorgegebenen Ziele erfüllen. Zu den Herausforderungen, die sich speziell für leistungsfähigere Agenten ergeben, gehört das *Reward Hacking*, bei dem Agenten unerwartete und mitunter unerwünschte Wege finden, um ihre Ziele zu erreichen [7, 79, 80]. Mit leistungsfähig ist hier gemeint, dass die kognitiven Fähigkeiten der Agenten die von Menschen bei der Lösung einer Problemstellung potenziell deutlich übersteigen.

Die zweite Kategorie umfasst normative Herausforderungen. Hier geht es um die Frage, *welche* Werte oder Grundsätze den Agenten vermittelt werden, wobei Iason Gabriel zwischen minimalistischen und maximalistischen Konzepten unterscheidet. Bei minimalistischen Konzepten geht es darum, Agenten an ein „plausibles“ Schema menschlicher Werte zu binden und unsichere Ergebnisse zu vermeiden. Bei maximalistischen Konzepten geht es darum, die Agenten an das „richtige“ oder „beste“ Schema menschlicher Werte auf einer gesellschaftsweiten oder globalen Basis anzupassen. Sie basiert auf der Beobachtung, dass die bedingungslose Optimierung einer einzelnen Messgröße aufgrund von unbedachten Seiteneffekten letztlich zu unerwünschten Ergebnissen führen kann. Diese Arbeit widmet sich den technischen Herausforderungen und entsprechend bezieht sich der Begriff (Selbst-)Ausrichtung nachfolgend immer auf diesen Bereich.

Diesem wurde im Kontext großer Sprachmodelle vermehrt Aufmerksamkeit zuteil, da diese mitunter Ausgaben generierten, die als nicht hilfreich, toxisch oder falsch empfunden wurden, was als unzureichende Ausrichtung interpretiert wurde. Ein gängiger Ansatz ist, unerwünschte Ausgaben nach der ersten Trainingsphase mit einer Feinabstimmung auf Basis von menschlichem Feedback zu minimieren [100]. Beispielsweise können Menschen die vom Sprachmodell gewünschten Ausgaben selbst formulieren. Diese werden dann als Demonstrationen beziehungsweise Ziel für ein weiteres Training des Sprachmodells verwendet. Alternativ oder zusätzlich können Menschen die Ausgaben des Sprachmodells bewerten. Mit *Reinforcement Learning from Human Feedback* (RLHF) [28] wird dann aus den so mitgeteilten Präferenzen eine Rewardfunktion approximiert und ein weiteres Training des Sprachmodells mit RL durchgeführt. RLHF erfordert keine Formulierung des Ziels im Sinne einer Sprache mit Syntax und Grammatik, da die Rewardfunktion aus paarweisen Vergleichen abgeleitet werden kann. Das ist insbesondere dann hilfreich, wenn eine präzise oder vollständige Beschreibung des gewünschten Ziels schwer ist.

Diese Arbeit verwendet RL, zieht die Verwendung von RLHF in Kapitel 4 jedoch nicht in Betracht, da die Annahme ist, dass hier eine (präzise) Definition des erlaubten oder gewünschten Verhaltens in Form einer Spezifikation vorliegt.

Die technischen Herausforderungen der Ausrichtung wurden ebenfalls im Kontext der *AI Safety* betrachtet, dessen Ziel die Vermeidung von Unfällen, Missbrauch und anderer negativer Folgen durch den (potentiellen) Einsatz hochentwickelter autonomer Systeme ist. Beispielsweise diskutieren Amodei et al. fünf grundlegende Probleme [7]. Dabei verwenden sie den Begriff Ausrichtung (noch) nicht, jedoch werden in nachfolgender Literatur [43] und in der Praxis die von ihnen definierten Probleme, insbesondere *Negative Side-Effects* (unerwünschte Nebeneffekte) und *Reward Hacking*, als Ursache für beziehungsweise Ausprägung von unzureichender Ausrichtung genannt. Leike et al. schlagen später eine Reihe von Szenarien vor, die spezifische Sicherheitseigenschaften von RL-Agenten testen [80]. Sie betrachten nicht nur die von Amodei et al. genannten, sondern auch einige RL-spezifische Bereiche, beispielsweise die Exploration. Dabei unterteilen sie die Probleme in die Bereiche *Robustheit* und *Spezifikation*. Robustheitsprobleme sind dann gegeben, wenn Agenten trotz Zugang zur Zielfunktion über Rewards nicht in Lage sind, diese zu erfüllen. Spezifikationsprobleme sind dann gegeben, wenn die Agenten unvollständigen Zugang zur Zielfunktion haben, weil sie nicht (vollständig) über Rewards abgebildet werden (können), was beispielsweise mit RLHF gelöst werden könnte. Hendrycks et al. verwenden den Begriff Ausrichtung, um eines von vier Feldern ungelöster Probleme beim Einsatz von ML Methoden zu beschreiben [59]. Ihrer Ansicht nach ist die Herausforderung hier, schwer zu spezifizierende menschliche Werte darzustellen und sicher zu optimieren, was hinsichtlich der Spezifikationsprobleme mit RLHF gelöst werden könnte.

Diese Arbeit betrachtet die von Amodei et al. genannten Probleme [7] als Ursachen für unzureichende Ausrichtung. Die Idee ist, die von Leike et al. definierten Spezifikationsprobleme [80] insofern zu vermeiden, als RL-Agenten Zugang zur gesamten Spezifikation über Rewards gegeben wird, damit sie lernen können, diese zu erfüllen, und ihr Verhalten dadurch selbst ausrichten. Hinsichtlich der Definition von Hendrycks et al. [59] fokussiert sich diese Arbeit auf die technische Realisierung der Ausrichtung in Form der sicheren Optimierung. Die Spezifikation menschlicher Werte und die Ableitung einer Rewardfunktion daraus, beispielsweise mittels RLHF, ist nicht das Ziel. Ebenso ist (beweisbar) sicheres Verhalten im Sinne der AI Safety nicht das primäre Ziel, aber es wird davon ausgegangen, dass lernende Agenten mit zusätzlichen Sicherheitsanforderungen konfrontiert sein werden. Der vorgestellte Ansatz hat das Potential, die Anzahl zusätzlicher Sicherheitsschichten zu reduzieren. Dies wird in Kapitel 4 durch beispielhafte Sicherheitsanforderungen innerhalb der nicht-funktionalen Anforderungen der Spezifikation gezeigt.

2.5 Entwicklung selbstanpassender Systeme

Die Entwicklung von Systemen, die sich selbstständig an dynamische Probleme anpassen, ist kein neuer Trend. Eine Möglichkeit ist, Systeme zu entwickeln, die ihre Parameter zur Laufzeit an veränderte Bedingungen anpassen. Erfolgreich umgesetzt werden kann eine solche *Parameteranpassung* (englisch: parameter adaptation) [94] meist nur, wenn die Anzahl und Art der anzupassenden Parameter a priori bekannt ist, wie beispielsweise beim *Transmission Control Protocol* (TCP) [37]. Eine Alternative sind Systeme, die Veränderungen auf einer höheren Ebene durchführen, um veränderten Bedingungen gerecht zu werden. In der Literatur wird dieses Konzept als *kompositorische* (englisch: compositional) [94] oder *architekturbasierte* (englisch: architecture-based) [45] Anpassung bezeichnet. Die vorliegende Arbeit folgt dieser Argumentation und klassifizert ein System dann als *selbstanpassend*, wenn es sich autonom verändert, um (mindestens) einen der folgenden drei Aspekte zu erfüllen [46]:

1. Die Implementierung einer Komponente wird durch eine andere ersetzt.
2. Die Struktur des Systems ändert sich, indem sich entweder die Verknüpfung zwischen den Komponenten ändert, Komponenten hinzugefügt oder entfernt werden.
3. Die Verteilung der Systemkomponenten ändert sich ohne Änderung der logischen Systemstruktur, beispielsweise indem Komponenten migrieren.

DL-Techniken ermöglichen ML-Systemen, größere Zustands- und Aktionsräume zu bewältigen als das mit klassischen Algorithmen möglich ist. Dies wurde bereits am Beispiel der RL-Algorithmen DQN in Kapitel 2.2.1 und PPO in Kapitel 2.2.2 verdeutlicht. Abbildung 2.3 gibt einen abstrakten Überblick über die Entwicklung von traditionellen, ML-basierten und DL-basierten Systemen. Entscheidend ist das Training, bei dem ein statistisches Modell aus Daten abgeleitet wird, anstatt explizit programmiert zu werden. Diese Arbeit versteht unter Modell respektive ML-Modell ein NN (oder mehrere) zur Funktionsapproximation und bezieht sich mit dem Begriff ML-Systeme insbesondere auf solche DL-Systeme. ML-Systeme sind jedoch nicht nur zu einer Parameteranpassung fähig, sondern sie könnten auch eine wichtige Rolle bei der Realisierung selbstanpassender Systeme spielen. Beispielsweise ist es nicht unüblich, bereits im Einsatz befindliche ML-Modelle mit im Betrieb gesammelten Daten weiter beziehungsweise erneut zu trainieren, um die Vorhersagequalität zu verbessern. Wenn die dafür erforderlichen Aktivitäten und Abläufe automatisiert werden, wären die resultierenden Systeme im Sinne der obigen Definition selbstanpassend.

Die Forschung hat sich bereits vor den jüngsten Fortschritten im Bereich des DL mit den Herausforderungen der Selbstanpassung befasst. Beispielsweise beschreibt Sinreich mit *MAPE-K* [138] einen High-Level-Architekturentwurf, der *Autonomic Computing* in verschiedene Phasen unterteilt und damit viele spätere Ansätze prägte. Autonomic Computing selbst zielt darauf ab, Aktivitäten

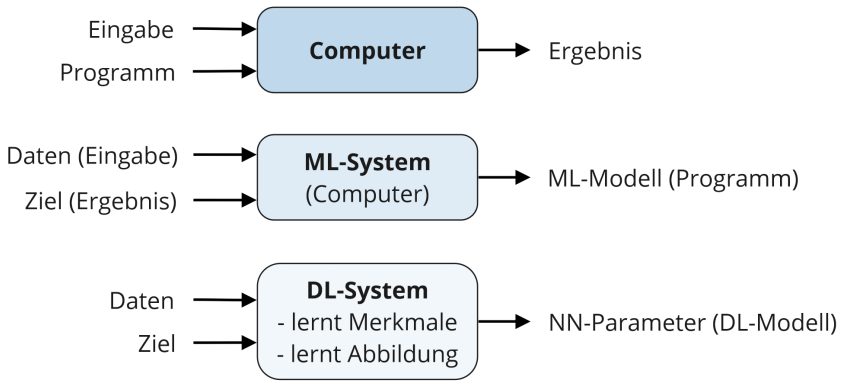


Abbildung 2.3: Vergleich der Entwicklung traditioneller (oben) und ML-basierter Systeme (mittig). DL-Systeme (unten) sind ML-Systeme, die relevante Merkmale in Daten und deren Abbildung auf eine Zielfunktion durch Anpassung von Parametern in neuronalen Netzen (NN) lernen. Darstellung angelehnt an [48].

des IT-Systemmanagement und Softwareprodukte mit Hilfe eines Softwareentwicklungszyklus zu entkoppeln. Um das zu ermöglichen, verwendet MAPE-K intelligente Kontrollschleifen, welche die Umgebung überwachen (**M**onitor), analysieren (**A**nalyze), einen **P**lan erstellen, diesen ausführen (**E**xecute) sowie dabei Wissen (**K**nowledge) verwenden und erweitern. Der Architekturentwurf enthält Anleitungen zum Aufbau und zur Implementierung der Wissensbasis, der Sensoren, der Aktoren und der fünf Phasen. Außerdem wird beschrieben, wie autonome Elemente zusammengesetzt werden können, um die Selbstverwaltung der beteiligten Systeme zu orchestrieren. Autonomic Computing war ursprünglich für die Verwaltung von IT-Systemen konzipiert, konnte jedoch auf ein breites Spektrum von Problemen übertragen werden, von denen das autonome Fahren wohl das bekannteste ist. Diese Arbeit widmet sich jedoch der Entwicklung von Systemen, die insbesondere Techniken des DL nutzen.

Bernon et al. entwickelten *ADELFE* [14], um die Softwareentwicklung von komplexen MAS zu erleichtern. ADELFE basiert auf dem *Rational Unified Process* [76] und bietet Werkzeuge für verschiedene Aufgaben des Softwaredesigns. Übersetzt lautet das französische Akronym in etwa „Toolkit für den Entwurf von Software mit emergenten Funktionalitäten“. Dabei berücksichtigt ADELFE das in Unterkapitel 2.3 vorgestellte AMAS-Konzept: ADELFE modelliert Systemkomponenten als Agenten mit Hilfe einer Reihe von Stereotypen, beispielsweise *perception*, *skill* oder *interaction*. Anschließend wird gemäß dem *Model-Driven Development* der Code (teil-) automatisiert auf Basis der modellierten Agenten generiert [18]. MAS bilden die Grundlage dieser Arbeit, aber im späteren Kapitel 5 soll ein Prozessmodell geschaffen werden, das nicht ausschließlich für diese konzipiert ist. Auch geht ADELFE wie andere AMAS-Ansätze davon aus, das MAS grundsätzlich Kooperation anstreben.

Die vorliegende Arbeit teilt diese Ansicht wie in Unterkapitel 2.3 dargelegt nicht, da die Annahme ist, dass künftig auch heterogene Agenten in offenen Umgebungen aufeinander treffen, die individuelle Ziele verfolgen und daher möglicherweise nicht bedingungslos miteinander kooperieren.

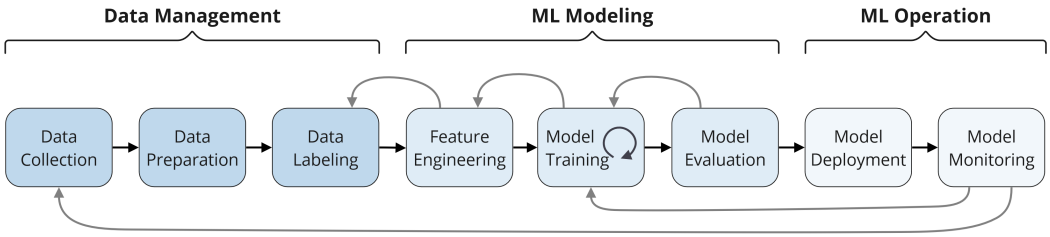


Abbildung 2.4: Phasen, Aktivitäten und Feedbackschleifen des ML-Entwicklungszyklus. Darstellung basierend auf [91] und in ähnlicher Form zuvor veröffentlicht in [123].

ML-basierte Systeme sind technisch gut geeignet, um Probleme zu lösen, in denen eine Parameteranpassung erforderlich ist, aber sie lösen nicht sämtliche Herausforderungen im Bereich selbstanpassender Systeme ad hoc. Stattdessen führen sie zunächst zu neuen Herausforderungen, beispielsweise im Bereich der Entwurfsmethoden und -prozesse, der Absicherung ML-basierter Software zur Laufzeit sowie der Einhaltung von Vorschriften und Gesetzen [19, 48, 93]. Für ein besseres Verständnis, woher diese Herausforderungen kommen, kann die Entwicklung von ML-Systemen betrachtet werden, deren Ablauf abstrakt in Abbildung 2.4 dargestellt ist. In dieser Darstellung des *ML-Entwicklungszyklus* werden die Begriffe von Lwakatare et al. [91] übernommen, die dem *Supervised Learning* (SL) [130] entstammen. SL ist eine Variante des DL, bei dem Modelle auf gegebenen Daten Vorhersagen treffen, die anders als im RL nicht sequentiell miteinander verknüpft sind, beispielsweise zur Klassifikation oder Regression. Für das Training werden entsprechend gekennzeichnete Daten benötigt, da diese Kennzeichnung das Ziel für die zu approximierende Klassifikation respektive Regression darstellt. Der zu minimierende Fehler wird technisch über verschiedene Fehlerfunktionen (englisch: loss) ausgedrückt.

Gemäß Abbildung 2.4 beginnt die Entwicklung von ML-Systemen in der Phase *Data Management*, zu der das Sammeln, Aufbereiten und Kennzeichnen der Daten gehört. Anschließend folgt eine zweite Phase *ML Modeling*, die mit dem Herausarbeiten relevanter Merkmale und Dimensionen in den Daten beginnt, gefolgt von wiederholten Trainings- und Testzyklen des ML-Modells, beispielsweise zur Abstimmung von Hyperparametern, bis die gewünschten Ergebnisse erzielt werden. Abschließend folgt die dritte Phase *ML Operation*, bei der das ML-Modell in den Betrieb übergeht und fortan laufend überwacht wird. Das Feedback aus dem Betrieb wird entweder dazu verwendet, ein bestehendes ML-Modell weiter zu trainieren, beispielsweise um unerwünschte Vorhersagen zu minimieren, oder um bestehende Datensätze zu erweitern.

Auf diesem Abstraktionsgrad unterscheiden sich verschiedene Varianten des DL nicht signifikant, weshalb mit dieser Darstellung auch das Vorgehen bei der Entwicklung von RL-Systemen beschrieben werden kann, was beispielhaft in Abbildung 1.1 zu sehen ist. Zu beachten ist lediglich, dass die Daten für das Training von RL-Agenten (meist) ad hoc aus der Interaktion mit der Umgebung generiert werden. Im RL umfasst das Data Management also Aktivitäten zur Repräsentation der Aufgabenstellung als SD-Problem (beispielsweise über eine Simulation), zur Modellierung der Interaktion des Agenten mit der Problemstellung (Beobachtungen und Aktionen) sowie zur Definition des Ziels, das im RL durch Rewards repräsentiert wird. Die Rewardfunktion übernimmt gewissermaßen eine automatisierte Kennzeichnung der Daten.

Bislang wurden einige Varianten des ML-Entwicklungszyklus auf der Grundlage erster Erfahrungen großer Softwareunternehmen mit DL-Verfahren berichtet, beispielsweise von IBM [3], Microsoft [6] und SAP [115]. Die jeweiligen Fallstudien konzentrieren sich auf abstrakte Aktivitäten des ML-Entwicklungszyklus und berücksichtigen nur am Rande, wie diese mit den bestehenden Vorgehensweisen der Softwareentwicklung verknüpft werden können. Moderne Paradigmen wie DevOps [81] favorisieren kurze, schnelle Entwicklungszyklen, die ineinander greifen beziehungsweise aufeinander aufbauen und eine kontinuierliche Bereitstellung von Softwareänderungen. Da ML-basierte Systeme ebenfalls iterativ entwickelt werden, liegt es nahe, beides miteinander zu verknüpfen. Daraus entwickelte sich *Machine Learning Operations* (MLOps) [75, 91], in dem die Aktivitäten rund um die Entwicklung von ML-Modellen nicht mehr isoliert sind. Die weitere Zusammenarbeit zwischen Industrie und Wissenschaft resultierte im *Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance Framework* (CRISP-ML(Q)) [141], das unter anderem zusätzlich technische Aktivitäten zur Qualitätssicherung definiert. Das später folgende Unterkapitel 5.3.2 geht genauer auf MLOps und CRISP-ML(Q) ein. Aus der Sicht dieser Arbeit fehlt MLOps und CRISP-ML(Q) eine klare Darstellung der Abhängigkeiten von Aktivitäten und den beteiligten Artefakten.

Ein zentraler Aspekt der klassischen Ansätze für selbstanpassende Systeme sind Feedback- beziehungsweise Kontrollschleifen [27], die auch in MAPE-K und ADELFE Verwendung finden. Das Ziel in Kapitel 5 ist, diese mit dem iterativen Vorgehen moderner Softwareentwicklung für ML-Systeme wie MLOps zu verknüpfen, ohne dies auf eine spezielle Vorgehensweise wie das AMAS-Konzept oder eine spezielle Technik wie RL einzuschränken. Ideal wäre, wenn Feedbackschleifen der ML-Entwicklung, die derzeit mehrheitlich rund um das Training der Modelle platziert sind, den gesamten Lebenszyklus dieser ML-Komponenten abdecken und automatisiert nachvollziehbare Entscheidungen über die weitere Anpassung im Sinne einer gegebenen Spezifikation getroffen werden könnten.

3 Selbstorganisation

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf individueller und globaler Ebene lösen können. Dieses Kapitel beinhaltet den ersten von insgesamt drei Abschnitten, die verschiedenen Aspekten dieser These nachgehen. Es befindet sich am Anfang des RL-Entwicklungszyklus (siehe Abbildung 1.1) und legt den Schwerpunkt auf die Definition der Problemstellung, die im RL durch die Umgebung repräsentiert wird.

Dieses Kapitel verfolgt das Ziel, autonome Agenten zu entwickeln, die ohne die a priori definierten Regeln der AMAS Ansätze [16] auskommen. Dazu wird untersucht, unter welchen Umständen Agenten mit RL in offenen, unkontrollierten Systemen lernen, sich selbst zu organisieren, also wann und wie sie miteinander interagieren. Besonderes Augenmerk liegt auf den Umständen der Entstehung emergenter Koordination und Kooperation bei der Interaktion der Agenten. Anders als verwandte Arbeiten verzichtet diese Arbeit dabei auf Techniken wie Reward Shaping [78], Kommunikation [66] und zentrale Koordination [88], um möglichst wenig Einfluss auf die Agenten zu nehmen.

Dieses Kapitel ist folgendermaßen strukturiert: Zunächst listet Unterkapitel 3.1 die Vorveröffentlichungen auf und legt dar, welche Inhalte in diese Arbeit übernommen wurden und welche Beiträge die (Ko-)Autoren geleistet haben. Dann beschreibt Unterkapitel 3.2 den Ansatz dieser Arbeit und grenzt diesen in Unterkapitel 3.3 von verwandten Ansätzen ab. Anschließend erklärt Unterkapitel 3.4, wie die verwendete Umgebung konzeptionell und technisch gestaltet ist. Unterkapitel 3.5 untersucht die Parametrisierung von Umgebung, RL-Algorithmus sowie den Ablauf des Trainings eines einzelnen Agenten und zeigt, dass nachhaltiges, also langfristig optimales Verhalten erlernbar ist. Basierend darauf analysiert Unterkapitel 3.6 die Umstände für das Entstehen von Koordination und Kooperation mit mehreren lernenden Agenten. Abschließend fasst Unterkapitel 3.7 die gewonnen Erkenntnisse zusammen.

3.1 Vorveröffentlichungen

Kapitel 3 stützt sich auf zwei wissenschaftliche Publikationen. Im Folgenden werden deren Verwendung im Rahmen der vorliegenden Arbeit sowie die Beiträge der (Ko-)Autoren erörtert.

Die erste Publikation mit dem Titel *Towards Ecosystem Management from Greedy Reinforcement Learning in a Predator-Prey Setting* [120] setzt RL ein,

um einen Jäger in einer 2D-Simulation zu trainieren, reproduzierende Beute zu fangen. Der Agent lernt durch Training in zwei Phasen sowie einer Kombination aus diskontierten Rewards und einer Historie vergangener Beobachtungen eine nachhaltige Strategie: Nur zu jagen, wenn noch Beute übrig ist, um die Beutepopulation zu erhalten. Die Publikation zeigt, dass eigennützige Ziele ausreichen, um einen Agenten mit RL zu realisieren, der langfristig optimal agiert. Der Agent erhält ein Gleichgewicht in seiner Umgebung aufrecht, indem er die Ressource nicht vollständig erschöpft. Die Umgebung der vorliegenden Arbeit basiert auf [56], wobei dort Beute mittels RL Fluchtstrategien lernt. Der Autor der vorliegenden Arbeit entwickelte in Diskussionen mit Thomas Gabor und Thomy Phan die Idee, stattdessen einen Jäger mit RL zu trainieren und hinsichtlich des Umgangs mit reproduzierender Beute zu untersuchen. Er überarbeitete die Implementierung der Umgebung, indem er beispielsweise elastische Kollisionen und simulierte Reibung ergänzte, und implementierte den verwendeten RL-Algorithmus DQN. Im Rahmen einer studentischen Projektarbeit implementierte Felix Hohnstein eine Pipeline für Training und Plotting und führte die Experimente unter Anleitung durch. Der Autor der vorliegenden Arbeit verfasste den Großteil der Publikation. Thomas Gabor unterstützte beim Verfassen der Einleitung und Robert Müller beim Verfassen des Kapitels *Related Work*. Thomy Phan trug das Kapitel *Reinforcement Learning* bei. Claudia Linnhoff-Popien stand für inhaltliche und strukturgebende Diskussionen der vorgestellten Konzepte zur Verfügung. Die Inhalte der nachfolgenden Kapitel 3.4 und 3.5 entstammen dieser Publikation.

Die zweite Publikation mit dem Titel *A Sustainable Ecosystem through Emergent Cooperation in Multi-Agent Reinforcement Learning* [124] überträgt den zuvor beschriebenen Ansatz [120] in ein Multi-Agenten-Szenario und ergänzt eine Hungermechanik, die Jäger nach einer gewissen Zeit ohne Fang verhungern lässt. Ein einzelnes gieriges Individuum kann so zum Aussterben von Beute und Jägern führen. Die Publikation analysiert systematisch, unter welchen Umständen bis zu drei Jäger mittels RL koordiniertes, nachhaltiges Verhalten lernen und so einen Kollaps des simulierten Ökosystems verhindern. Darauf aufbauend wird untersucht, welche Umstände die Jäger zu komplexer Kooperation in Form von Gruppenjagd inzentivieren. Der Autor der vorliegenden Arbeit entwickelte die Idee, das Konzept und leitete die Umsetzung. Daniel Ratke implementierte im Rahmen einer studentischen Projektarbeit die Hungermechanik, erweiterte die bestehende Pipeline für das Training mehrerer Agenten und führte die Experimente unter Anleitung durch. Thomy Phan und Lenz Belzner halfen durch regelmäßige Diskussionen, die Analyse zu schärfen. Thomy Phan brachte zusätzlich die Idee des *Stunning* Mechanismus ein. Der Autor der vorliegenden Arbeit verfasste den Großteil der Publikation. Thomy Phan trug die Kapitel *Reinforcement Learning* und *Multi-Agent Reinforcement Learning* bei. Claudia Linnhoff-Popien stand für inhaltliche und strukturgebende Diskussionen der vorgestellten Konzepte zur Verfügung. Die Inhalte der nachfolgenden Kapitel 3.4 und 3.6 entstammen dieser Publikation.

3.2 Problemmodellierung und Vorgehen

Eine erstrebenswerte Eigenschaft autonomer Agenten ist, nicht „blind“ mit anderen Agenten zu koexistieren, sondern geeignet (sozial) miteinander zu interagieren [160]. Wie Consoli et al. [29] unterscheidet die vorliegende Arbeit zwischen *Koordination* und *Kooperation*. Koordination ist erforderlich, wenn Abhängigkeiten zwischen den Aktivitäten der Agenten bestehen, also beispielsweise Ressourcen wie Ladestrom oder Parkplätze geteilt werden. Kooperation erfordert Koordination, um sich initial auf ein gemeinsames Ziel zu einigen und ist beispielsweise erforderlich, wenn ein Agent eine Aufgabe nicht allein erfüllen kann oder das globale Wohlergehen dadurch verbessert wird. Diese Arbeit nimmt die in Kapitel 2.3 erläuterte *Agent-zentrierte* Sichtweise ein, in der *Organisation* als globales Phänomen auf Basis lokaler Interaktion der Agenten entsteht. Dabei wird ein kooperatives Szenario angenommen, in dem Agenten dasselbe oder ein zentral koordiniertes Ziel verfolgen und kooperativ handeln [1, 16]. Dies ist in isolierten, kontrollierten Umgebungen wie Lagerhäusern oder Fertigungsstraßen realistisch. Diese Arbeit geht hingegen von einem *allgemeinen Szenario* aus, in dem Agenten individuelle Ziele verfolgen, was häufig in offenen Umgebungen der Fall ist, und stets ihr eigenes Wohlergehen maximieren, was als *eigennütziges* Handeln bezeichnet wird. Die Agenten werden mit RL realisiert, einer bisher wenig verwendeten Technik [162] zur Untersuchung von Organisation in MAS.

Problemstellungen werden im RL durch die Umgebung repräsentiert. In diesem Kapitel modelliert die Umgebung die Nutzung (im Sinne des Konsums oder Abbaus) einer selbsterneuernden Ressource, wie sie in Form nachwachsender Organismen in natürlichen Lebensräumen zu finden ist. Üblicherweise ist es erstrebenswert, eine solche Ressource so stark wie möglich zu nutzen ohne sie vollständig zu erschöpfen, was einer Ausrottung der entsprechenden Organismen gleichkäme und eine künftige Nutzung der Ressource mangels Erneuerung verhindern würde. Wechselwirkungen mit weiteren Organismen, die üblicherweise in natürlichen Ökosystemen vorkommen, werden in dieser Arbeit nicht berücksichtigt.

Das Ziel in dieser Umgebung ist, die Summe der abgebauten Ressource vor einem infiniten Horizont zu maximieren. Hier profitiert ein einzelner Agent davon, die abzubauende Ressource niemals vollständig zu erschöpfen, da er sich so langfristig einen in Summe höheren Ertrag sichern kann. Nutzen weitere Agenten diese Ressource, müssen sich alle Agenten darauf einigen, die Ressource niemals zu erschöpfen, um das global optimale Ergebnis zu erreichen. Jedoch kann es zu Konflikten kommen: Da die Zuteilung der Ressource nicht geregelt ist, können Agenten *gierig* handeln, indem sie versuchen, ihren Anteil an der Ressource zu maximieren. Die rationale Reaktion der übrigen Agenten auf gieriges Handeln ist, selbst einen möglichst großen Teil der (verbleibenden) Ressource zu sichern, um nicht ausgenutzt zu werden, was als *ängstliches* Handeln bezeichnet wird. Die Agenten befinden sich hier in einem

sozialen Dilemma, in dem Gier und Angst zum restlosen Abbau der Ressource führen können [78]. Ebenso können äußere Faktoren, beispielsweise unmittelbarer Überlebensdruck, den Anstoß geben, ungeachtet der langfristigen Folgen kurzsichtig zu handeln. In Spieltheorie und Wirtschaftswissenschaften sind solche Szenarien als *Tragik der Allmende* bekannt [57, 86]. In der Praxis finden sich Parallelen zum Umgang der Menschen mit natürlichen Ressourcen, an denen keine exklusiven Verfügungsrechte definiert sind, beispielsweise der Überfischung der Meere.

Auf abstrakter Ebene ist eine Möglichkeit zur Lösung des Problems, gemeinsame Ressourcen zu rationieren und Regeln für die Zuteilung zu definieren. Dies funktioniert jedoch nur, wenn bei Nichteinhaltung der Regeln wirksame Sanktionen verhängt und durchgesetzt werden können, was sich in der Praxis als schwierig erweist. In diesem Beispiel ist zu beachten, dass es komplex sein kann, eine nachhaltige und langfristig optimale Nutzung einer selbsterneuernden Ressource zu berechnen, um daraus Regeln abzuleiten. Dies würde die Identifizierung und Simulation relevanter Wechselwirkungen sowie eine Berücksichtigung der Dynamik zur Laufzeit erfordern, beispielsweise eine Änderung der Anzahl der Agenten.

Diese Arbeit schlägt daher einen anderen Weg ein und sucht nach Umständen, unter denen lernende Agenten ohne vorgegebene Regeln die Tragik der Allmende vermeiden. Dies wird als *Koordination* betrachtet, einer Form der in Kapitel 2.3 eingeführten *Selbstorganisation*, die als emergenter Effekt global beobachtet und im Umgang mit der gemeinsamen Ressource gemessen werden kann. Konkret kann das beispielsweise eine temporäre Schonung der Ressource bei geringem Vorkommen sein, ohne dass dies explizit forciert wird, beispielsweise wie [16] durch Definition von erstrebenswerten, kooperativen Situationen. Ebenso werden die Bedingungen untersucht, unter denen komplexe *Kooperation* wie beispielsweise Gruppenjagd entsteht, eine weitere Form der Selbstorganisation. Der Ansatz dieser Arbeit ist, Agenten selbst lernen zu lassen, wann und wie sie kooperieren. Um die entstehende Lösung technisch möglichst wenig zu beeinflussen, wird auf Mechanismen verzichtet, die eine bestimmte Form der Organisation begünstigt, beispielsweise Kommunikation [66] und Koordination durch eine zentrale Wertfunktion [88].

3.3 Verwandte Arbeiten

Im Folgenden wird der entwickelte Ansatz in insgesamt fünf Richtungen abgegrenzt: *Selbstorganisation* (Unterkapitel 3.3.1), *Organisation in Schwarmsystemen* (Unterkapitel 3.3.2), *Organisation durch globale Informationen* (Unterkapitel 3.3.3), *Langfristig optimales Verhalten* (Unterkapitel 3.3.4) und *Entstehung von Altruismus und Kooperation* (Unterkapitel 3.3.5).

3.3.1 Selbstorganisation

Die erste Richtung der Abgrenzung ist Selbstorganisation in Multi-Agenten-Systemen, auf der das in Kapitel 2.3 eingeführte AMAS-Konzept [47] beruht. Es nutzt die Kooperation autonomer Agenten auf lokaler Ebene, um die Komplexität auf globaler Ebene zu bewältigen. Diese Bottom-up Vorgehensweise zielt auf Anpassungsfähigkeit, Skalierbarkeit und Robustheit ab. In [16] stellen Boes et al. das Systemsteuergerät *ESCHER* vor, das nach dem AMAS-Konzept entwickelt wurde. Die Agenten in ESCHER steuern mit Hilfe kooperativer Selbstorganisation einen Verbrennungsmotor in Echtzeit. ESCHER testet die Wirkung von Änderungen in den Eingabeparametern auf die Ausgabeparameter in Black-Box-Manier und leitet daraus ab, welche Aktionen notwendig sind, um die Zielvorgaben zu erfüllen. Der grundlegende Unterschied zur vorliegenden Arbeit ist, dass kooperative und nicht-kooperative Situationen, deren Erkennung sowie entsprechende Handlungsvorgaben für die Agenten a priori durch die Entwickler definiert wurden.

Ye et al. [162] geben einen Überblick über Mechanismen zur Selbstorganisation in Multi-Agenten-Systemen, wobei nur ein kleiner Teil der Ansätze RL nutzt. Die Agenten von Abdallah et al. [1] sind Teil eines Graphen und lösen ein kooperatives Ressourcenzuweisungsproblem. Sie können Aufgaben selbst abarbeiten oder an benachbarte Agenten weiterleiten, Erfahrung teilen und die Struktur des Graphen verändern, also Nachbarn hinzufügen oder entfernen. Anders als die vorliegende Arbeit nutzen Abdallah et al. Kommunikation und ordnen die Agenten in einer expliziten Struktur an. Ebenfalls erwähnenswert sind die hierarchischen Ansätze von Zhang et al. [164, 165], die die Agenten in Gruppen einteilen und die (Selbst-)Organisation auf Gruppenebene heben, wobei jeweils ein Betreuer pro Gruppe eingesetzt wird. Das Ziel ist, die Skalierung von MARL zu verbessern, beispielsweise hinsichtlich lokaler Exploration einzelner Agenten. Anders als die vorliegende Arbeit nutzen Zhang et al. dafür Kommunikation und bilden die Organisation explizit in Form von Gruppen und Betreuern ab. Ein ähnliches Vorgehen, das Organisation explizit auf dem Level der Gesamtpopulation modelliert, wählen Leibo et al. [77] im *Malthusian RL*. Die von ihnen entwickelte (Ko-)Evolution mehrerer *Gemeinschaften* von RL-aAgenten erlaubt das Entstehen von Spezialisierung und Aufgabenteilung. Dazu geben Leibo et al. die Organisationsstruktur in Form der Gemeinschaften vor. Dies widerspricht der Zielstellung dieser Arbeit, Selbstorganisation ohne explizit definierte Struktur zu erreichen. Umgekehrt kann die in dieser Arbeit angewandte Methodik, je nach Zielstellung, innerhalb einzelner Gemeinschaften des Malthusian RL angewendet werden.

3.3.2 Organisation in Schwarmsystemen

Die zweite Richtung der Abgrenzung sind künstliche Schwarmsysteme mit lernenden Agenten, in denen Organisation auf verschiedene Art entsteht oder

erzeugt wird, jedoch nicht im Fokus der Untersuchung steht. Um eine Schwarmformation zwischen homogenen Agenten zu erzielen, verwenden Pinsler et al. [114] *Inverse RL*, um eine entsprechende Reward Funktion für RL abzuleiten, und Bezcioglu et al. [15] setzen RL ein, um die Gewichte eines vorgegebenen Partikelmodells entsprechend anzupassen. In beiden Szenarien gibt es keine Jäger und die Organisation als Schwarm ist das Ziel, das es zu erreichen gilt. Die vorliegende Arbeit geht hingegen davon aus, dass die optimale Form der Organisation nicht bekannt ist und überlässt es den Agenten, diese zu lernen. Sunehag et al. [143] beobachten in Experimenten ebenfalls Schwarmbildung und Formen von Symbiose in einem simulierten Ökosystem mit drei Arten von Agenten, wobei MARL mit *Reward Shaping* zum Einsatz kommt und die Ergebnisse entsprechend beeinflusst. Hahn et al. [56] beobachten, dass eigennützige Beute ohne Reward Shaping einen Schwarm bildet, wenn die Nähe zu anderen Individuen die Wahrscheinlichkeit erhöht, dass ein Jäger von der Verfolgung ablässt und stattdessen andere, gegebenenfalls nähere Individuen verfolgt. Anschließend zeigen sie empirisch [55], dass die Schwarmbildung in diesem Szenario ein *Nash-Equilibrium* ist und ein individuelles Fluchtverhalten das langfristige Überleben der Beute verbessern würde, wenn die gesamte Population diese Fluchtstrategie verwendet. Einzelne Individuen haben jedoch keinen Anreiz, die Schwarmformation zu verlassen, da sie im offenen Raum ein leichteres Ziel für die Jäger sind als die übrigen Agenten, die innerhalb der Schwarmformation bleiben. Abweichend vom Szenario der vorliegenden Arbeit können Jäger und Beute bei Hahn et al. nicht aussterben und statt der Jäger wird die Beute mit RL trainiert. Khan et al. [72] geben einen Überblick über (künstliche) Motivation in Multi-Agenten- und Schwarmsystemen. Mit Motivation ist hier gemeint, dass Agenten fortlaufend Wissen und Fähigkeiten in einer offenen Umgebung erwerben und ihr Verhalten entsprechend weiterentwickeln. Khan et al. kritisieren, dass sich die Mehrheit der Arbeiten (wie in diesem Unterkapitel) auf Partikelschwarmoptimierung oder Schwarmbildung beschränkt und diese Szenarien nicht ausreichend Anreiz für fortlaufendes Lernen bieten. Sie schlagen vor, insbesondere Szenarien zu untersuchen, die zur Lösung die aktive Zusammenarbeit mehrerer Agenten erfordern. Die vorliegende Arbeit geht diesem Vorschlag in Unterkapitel 3.6 nach und untersucht die Umstände, unter denen komplexe Kooperation wie Gruppenjagd entsteht.

3.3.3 Organisation durch globale Informationen

Die dritte Richtung der Abgrenzung ist die Organisation von RL-Agenten mittels globaler Informationen. Hier untersuchen Yang et al. [161] die Dynamik großer Jäger- und Beutepopulationen, in der die Jäger mit einer zentralisierten Variante von DQN trainiert werden, in der alle Aktionen von derselben Strategie getroffen werden. Sie beobachteten eine *Wax-and-Wane* Form zwischen den Populationen, die mit dem Lotka-Volterra-Modell [87] erklärt werden kann. Initial vergrößert Jagderfolg die Jägerpopulation und verringert die

Beutepopulation. Anschließend schrumpft die Jägerpopulation aufgrund von Nahrungsknappheit, was zu einem Anstieg der Beutepopulation führt, bis sich der Zyklus wiederholt. Abweichend von der Modellierung der vorliegenden Arbeit ermöglichen Yang et al. Jägern, sich zur gemeinsamen Jagd explizit zu Gruppen zusammenzuschließen. Lowe et al. [88] entwickelten eine CTDE Variante der Actor-Critic-Architektur, in der mehrere Agenten (Actor) auf Basis lokaler Beobachtungen handeln und während dem Training mit Hilfe einer zentralen Wertfunktion (Critic) lernen. Der Critic verfügt über globale Information in Form der Beobachtungen und Aktionen aller Agenten. Lowe et al. führen Experimente in gemischt kooperativ-kompetitiven Szenarien durch, darunter auch ein Jäger-Beute-Szenario, wobei der Fokus auf koordiniertem Vorgehen der Agenten in kurzen Aufgaben liegt. Diese CTDE Variante erreicht bessere Ergebnisse als vollständig dezentrale Actor-Critic-Architekturen, was darauf hindeutet, dass die Organisation der Agenten leichter an zentraler Stelle zu erlernen ist, wenn Szenarien Koordination erfordern. Hüttenrauch et al. stellen in [66] eine CTDE Variante für Schwarmsysteme vor. Ihre Agenten kommunizieren in lokaler Nachbarschaft, beispielsweise um Informationen über zu lokalisierende Ziele auszutauschen. Basierend darauf stellen Hüttenrauch et al. [65] eine effiziente Repräsentation der lokalen Beobachtungen homogener Agenten in Jäger-Beute-Szenarien vor, um die Skalierung zu verbessern. Sie interpretieren Agenten als Stichprobe einer Verteilung und treffen dezentral Entscheidungen auf Basis des empirischen Mittelwerts, der invariant hinsichtlich der Anzahl der effektiv sichtbaren Agenten ist. Anders als die drei vorhergehenden Arbeiten modelliert die vorliegende Arbeit keinen expliziten Austausch von Informationen zwischen den Agenten, beispielsweise über Kommunikation oder durch einen zentrale Wertfunktion. Die Agenten müssen sich stattdessen über lokale, gegenseitige Beobachtung koordinieren.

3.3.4 Langfristig optimales Verhalten

Die vierte Richtung der Abgrenzung ist die Verwendung spezieller Techniken des RL, um langfristig optimales Verhalten zu erreichen. Kurzfristige Belohnungen sind für eine erfolgreiche Exploration wichtig, jedoch kann eine zu starke Gewichtung dieser dazu führen, dass der Agent in lokalen Optima stecken bleibt, ohne das eigentliche Ziel zu erreichen [145]. Die *FTW* Architektur [68] agiert auf zwei Zeitskalen, um konsistente Aktionsfolgen im Sinne langfristiger Strategien in komplexen Aufgaben zu ermöglichen. Ein schnelles *Recurrent Neural Network* (RNN) approximiert die kurzfristige Dynamik der Umgebung und ein langsames RNN approximiert die langfristige Korrelation von Aktionen und Beobachtungen. Jaderberg et al. verwenden diese Architektur beim Training einer diversen Population von Agenten, die in Teams (kooperativ) gegen andere Teams (kompetitiv) antreten. Erfolgreiche Individuen bilden die Grundlage für spätere Generationen. In [149] erweitern Vinyals et al. diesen Ansatz um weitere spezialisierte Netzarchitekturen wie *Pointer Networks* [150],

LSTMs [61] und *Transformern* [148], wobei einzelne Agenten das Niveau der besten menschlichen Spieler in einem modernen Echtzeitstrategiespiel erreichen. Einen anderen Weg gehen Ha et al. [52] und Hafner et al. [53, 54], die ein explizites Modell der Dynamik der Umgebung mit RNNs lernen. Agenten können mit diesem Modell der Umgebung „träumen“, also in fiktiven Umgebungen planen und lernen. Die vorliegende Arbeit verwendet kein (gelerntes) Modell der Umgebung. Inspiriert von den populationsbasierten Ansätzen [68, 149] werden in Unterkapitel 3.5 Agenten mit DQN in zwei Phasen trainiert. An Stelle von RNNs wird jedoch eine Historie aus vergangenen Beobachtungen τ in variabler Länge genutzt. Zusammen mit einem geeignet gewählten Parameter γ, der den Einfluss unmittelbarer und künftig zu erwartender Rewards gewichtet, reicht dies, um im untersuchten Szenario ein adaptives, weitsichtiges Verhalten zu erreichen. Daher kann hier auf komplexe und langwierig zu trainierende Architekturen verzichtet werden.

3.3.5 Entstehung von Altruismus und Kooperation

Die fünfte Richtung der Abgrenzung ist die Analyse der Umstände, unter denen Altruismus und Kooperation entstehen. Die nicht-kooperative Spieltheorie geht davon aus, dass Akteure grundsätzlich eigennützig handeln und sich sozial positive Gleichgewichte nur durch Zufall in Einzelfällen, aber nicht zwangsläufig im allgemeinen Fall einpendeln. Beispielsweise kann bei gemeinsam genutzten Ressourcen eine Tragik der Allmende [57, 86] beobachtet werden, bei der Ressourcen durch egoistisches Verhalten erschöpft anstatt gerecht geteilt werden. Analytische Spieltheorie wird primär in Matrixspielen angewandt, in denen Kooperation und Defekt atomare Aktionen sind und genau zwei Akteure teilnehmen. Dies erlaubt eine präzise Formulierung von Strategien wie *tit-for-tat* [8], die Kooperation auf Basis von *Gegenseitigkeit* ermöglicht. Im wiederholten Gefangenendilemma, einem Matrixspiel, kombinieren Tanabe et al. [146] einen RL-Algorithmus mit einer evolutionären Strategie und zeigen, dass auch Lernen das Entstehen von Kooperation ermöglicht. Die Autoren sehen ihre Ergebnisse als bestätigendes Beispiel für den *Baldwin-Effekt*, bei dem Lernen die Evolution in Richtung der optimalen Lösung beschleunigt. Szenarien mit mehr als 2 Akteuren oder nicht atomaren (im Sinne der Spieltheorie) Aktionen können nur mit Einschränkung auf solche Matrixspiele abgebildet werden. Eine Alternative ist, die Szenarien mit Agent-basierten Simulationen zu modellieren und empirisch auszuwerten. Beispielsweise zeigt die anthropologische Simulation von Burtsev [25], dass das Ressourcenangebot das Verhältnis von friedlicher Zusammenarbeit und Aggression beeinflusst. Enger verwandt mit dem Szenario der vorliegenden Arbeit untersuchen Leibo et al. in [78] mit zwei lernenden Agenten, wie Konflikte aus der Nutzung gemeinsamer Ressourcen entstehen können und kommen zu zwei für die vorliegende Arbeit wichtigen Erkenntnissen. Erstens handeln Agenten aggressiver, wenn die Ressource knapp ist, was eine Tragik der Allmende begünstigen kann. Zweitens kann das

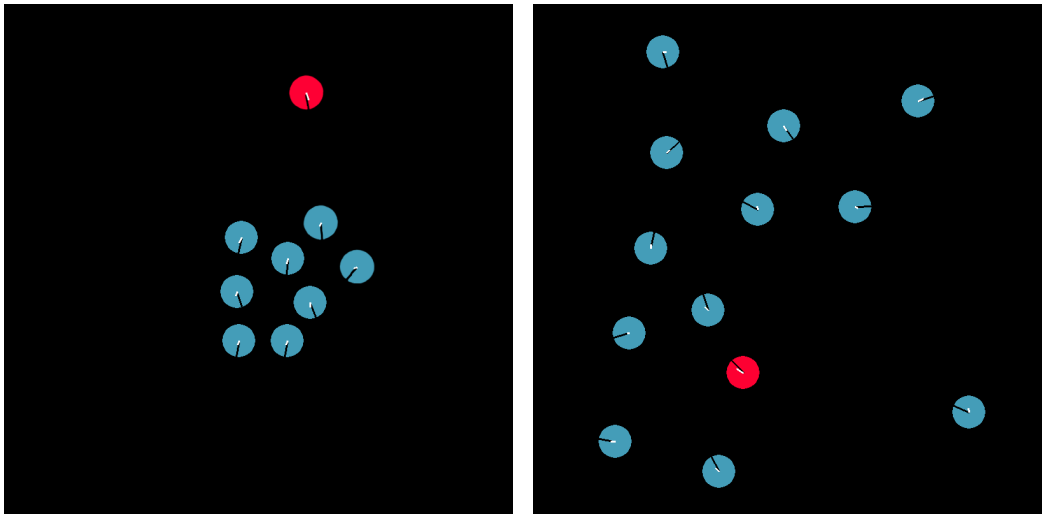
Entstehen von Kooperation über *Reward Shaping* in Form zusätzlicher Rewards gefördert werden, was sie in einem Szenario demonstrieren, in dem zwei Jäger gemeinsam Beute fangen. Ein weiteres Beispiel für die Incentivierung von Kooperation liefern Phan et al. mit *MATE* [112], einem Protokoll, gemäß dem Agenten sich abhängig von ihrem individuellen Wohlergehen und den Aktionen anderer Agenten zusätzliche Rewards schicken. In diesem Abschnitt der vorliegenden Arbeit wird auf zusätzliche Rewards für Interaktionen verzichtet, um menschlichen Bias in der Lösung zu minimieren. Perolat et al. untersuchen in [104] mit einer großen Anzahl von Agenten, welches Verhalten eigennützige Agenten bei der Nutzung gemeinsamer Ressourcen erlernen. Sie berichten über unterschiedliche Strategien zu unterschiedlichen Zeitpunkten des Trainings, die zusätzlich von den Ausgangsbedingungen beeinflusst werden, und messen diese anhand sozialer Metriken wie *Frieden*, *Effizienz*, *Gleichheit* und *Nachhaltigkeit*. Die vorliegende Arbeit misst ebenfalls die Nachhaltigkeit der Agenten, jedoch ist die Metrik anders definiert. Ebenso anders als in [78, 104] können Jäger im Szenario der vorliegenden Arbeit verhungern. Indem diese Dauer gezielt beeinflusst wird, können die Umstände, die zu einer Tragik der Allmende führen, genauer untersucht werden.

3.4 Fallstudie: Jäger-Beute-Szenario

Um im weiteren Verlauf dieses Kapitels die Entstehung von Koordination und Kooperation zu untersuchen, wird eine kontinuierliche, zweidimensionale Simulation mit zwei Arten von Agenten verwendet: Jägern, beispielsweise Haien, und Beute, beispielsweise kleineren Fischen. Die Beute modelliert eine natürliche, selbsterneuernde Ressource mit exponentiellem Wachstum. Das Ziel der Jäger ist, möglichst viel Beute zu fangen. Im Rahmen der folgenden Untersuchungen werden die Jäger mit RL trainiert. Das Ziel der Beute ist, nicht gefangen zu werden. Die Beute wird von Heuristiken gesteuert und reproduziert sich, wenn es die in Kapitel 3.4.1 definierten Regeln erlauben. Beute flieht entweder gemeinsam als Schwarm oder unabhängig voneinander (siehe Abbildung 3.1). Beide Fluchtstrategien wurden von Hahn et al. [56] definiert und untersucht. Hahn et al. beobachteten, dass individuell fliehende Beute höhere Überlebenschancen hat.

3.4.1 Simulation

Das Ziel ist ein Szenario, in dem das Fangen von Beute (alleine oder gemeinsam) keine atomare Entscheidung respektive atomare Aktion ist. Die Metapher für das Szenario entstammt zwar der Natur, aber es soll eine Brücke in Richtung komplexer, kontinuierlicher Kontrolle geschlagen werden, wie sie beispielsweise in der Robotik nützlich ist. Hier stellt sich nicht nur die Fra-



(a) als Schwarm fliehende Beute (b) individuell fliehende Beute

Abbildung 3.1: 2D Visualisierung der Jäger-Beute-Simulation, Jäger in rot, Beute in cyan. Darstellung mit angepassten Farben übernommen aus [120].

ge, ob Agenten miteinander kooperieren, sondern auch, wie sie ihr Verhalten dynamisch aufeinander abstimmen.

In Jäger-Beute-Szenarien spielen Fluchtmöglichkeiten eine zentrale Rolle. Daher wird der Raum, in dem sich die Agenten befinden, flexibel gestaltet: Je nach Experiment ist er entweder durch Wände begrenzt oder offen. Ist der Raum offen, betreten Agenten, die den Raum auf einer Seite verlassen, ihn wieder auf spiegelverkehrter Seite. Geometrisch entspricht dies einem Torus. Um die Effizienz zu steigern, können Agenten immer nur eine begrenzte Anzahl von Wänden und anderen Agenten wahrnehmen, weshalb das Szenario als *partiell beobachtbar* einzuordnen ist. Die gewählte Variante der nächsten Nachbarn ist detailliert im folgenden Kapitel 3.4.3 beschrieben. Agenten nehmen andere Agenten oder Wände nur innerhalb eines definierten Sichtradius wahr. Die Beute überblickt nur einen Teil des Raumes. Um den Jägern in Gegenwart von wenig Beute eine präzise Entscheidung zu ermöglichen, können sie den gesamten Raum überblicken.

Agenten werden durch Kreise gleichen Durchmessers repräsentiert. Sie bewegen sich, indem sie ihre lineare Geschwindigkeit (*Beschleunigung*) und ihre Winkelgeschwindigkeit (*Orientierung*) verändern. Diese *Doppelintegrator Dynamik* modelliert eine einfache Masse unter der Wirkung einer zeitlich veränderlichen Krafteinwirkung und ist ein Anwendungsfall für Kontrollsysteme zweiter Ordnung [116]. Die Höchstgeschwindigkeit jedes Agenten wird durch simulierte Reibung begrenzt und ist der Quotient aus Reibungskonstante und maximaler Beschleunigung.

Agenten können miteinander oder mit Wänden kollidieren, wobei eine elas-

tische Kollision simuliert wird. Eine Elastizitätskonstante bestimmt, wie viel kinetische Energie erhalten bleibt. Dabei haben alle Agenten die gleiche Masse und Wände werden als unbeweglich (unendliche Masse) modelliert. Wenn ein Jäger mit einem Artgenossen kollidiert, der ihm abgewandt ist, wird der Artgenosse betäubt (*Stunning*), d. h. er kann für eine bestimmte Anzahl von Schritten weder Beschleunigung, noch Orientierung ändern. Wenn beide Jäger während des Zusammenstoßes einander zugewandt sind, werden beide betäubt. Wenn ein Jäger mit einer Beute kollidiert, gilt die Beute als gefangen und wird aus der Simulation entfernt.

Eine Beute kann sich vermehren, also eine weitere Beute an ihrer derzeitigen Position erzeugen, wenn alle folgenden Bedingungen erfüllt sind:

1. Es befindet sich kein Jäger im Sichtradius der Beute.
2. Seit der letzten Reproduktion ist eine definierte Anzahl von Zeitschritten vergangen.
3. Das Populationslimit der Beute ist nicht überschritten.

Die erste Bedingung hilft, schlechte Jäger von solchen zu unterscheiden, die bewusst Abstand zu Beute halten, um sie zu schonen. Die zweite und die dritte Bedingung ermöglichen eine Regulierung des Wachstums der Beutepopulation. Insbesondere kann so vermieden werden, dass die Beute sich so stark vermehrt, dass sie keinen Platz zum fliehen hat. Jäger, die gewisse Zeit keine Beute fangen, verhungern und werden aus der Simulation entfernt. Die initiale Überlebensdauer ohne Fang variiert in den nachfolgenden Experimenten, die zusätzliche Überlebenszeit pro gefangener Beute ist hingegen konstant. Die Parameter der Simulation sind in Tabelle 3.1 aufgelistet.

3.4.2 Aktionen

Der Aktionsraum A_k besteht aus drei kontinuierlichen Werten. Der erste Wert repräsentiert die Beschleunigung, der zweite die Orientierung. Die Beschleunigung ist auf $[-1, 1]$ und die Orientierung auf $[-\pi, \pi]$ begrenzt. Der dritte Wert definiert, ob eine Reproduktion stattfinden soll. Jäger reproduzieren sich nicht, Beute reproduziert sich immer so schnell, wie es die in Kapitel 3.4 definierten Regeln erlauben. Agenten, die von Heuristiken gesteuert werden, können auf den vollständigen, kontinuierlichen Aktionsraum zugreifen. Mathematisch bedingt erreichen manche RL-Algorithmen wie DQN nur mit einer geringen Anzahl diskreter Aktionen eine adäquate Performance. Für solche RL-Algorithmen wird der kontinuierliche Aktionsraum wie folgt diskretisiert: Fünf Aktionen kombinieren volle lineare Beschleunigung mit den Orientierungsänderungen $-\frac{\pi}{3}$, $-\frac{\pi}{6}$, 0, $\frac{\pi}{6}$, $\frac{\pi}{3}$. Die sechste Aktion ist eine NO-OP ohne Beschleunigung und Orientierungsänderung.

Simulationsparameter	Wert
Größe des zweidimensionalen Raums	30×30
Begrenzt durch Wände	ja
Durchmesser von Jägern und Beute	2
Populationslimit der Beute b^{max}	11
Sichtradius der Beute	10
Sichtradius der Jäger	30
Maximale Anzahl wahrnehmbarer Beute	3
Maximale Beschleunigung der Jäger	0.06
Maximale Beschleunigung der Beute	0.04
Maximale Orientierungsänderung der Jäger	$\frac{\pi}{3}$
Maximale Orientierungsänderung der Beute	π
Minimale Schritte bis zur Reproduktion der Beute	100
Reibungskonstante der Jäger	0.10
Reibungskonstante der Beute	0.08
Elastizität (alle Kollisionen)	0.20

Tabelle 3.1: Schlüsselparameter der simulierten RL-Umgebung.

3.4.3 Beobachtungen und Rewards

Der Beobachtungsraum O_k ist für alle Agenten einheitlich strukturiert. Er enthält zunächst die Orientierung des Agenten und ein Flag, ob eine Reproduktion möglich ist. Er enthält weiterhin eine Liste von Wänden und eine Liste anderer Agenten. Abhängig vom Sichtradius und den im Folgenden erläuterten Grenzen können nicht alle Wände und Agenten beobachtet werden. Die Umgebung ist also nur teilweise beobachtbar. Die Liste der Wände besteht aus Entfernung und Winkel zwischen dem aktuellen Agenten und jeder der nächstgelegenen n_w Wände. Wenn aufgrund eines begrenzten Sichtradius oder weil die Umgebung ein Torus ist keine Wände sichtbar sind, wird über *Zero-Padding* (Null-Einträge) der *Offset* (Abstand) der nachfolgenden Elemente konstant gehalten. Anschließend folgen n_{pred} Raubtiere. Jeder Jäger wird, ausgehend vom aktuellen Agenten, durch Entfernung, Winkel und Orientierung charakterisiert. Jäger außerhalb des Sichtradius führen auch hier zu Nulleinträgen. Zuletzt sind n_{prey} Beutetiere in der Beobachtung enthalten, die durch die gleichen Eigenschaften wie Jäger beschrieben werden. n_w, n_{pred} und n_{prey} bleiben während Training und Evaluation gleich. Unabhängig von der Anzahl der beobachtbaren Entitäten ist die Beobachtung durch Zero-Padding mathematisch ein Vektor konstanter Größe.

Jäger erhalten einen Reward von +10 pro gefangener Beute. Sind mehrere Jäger beteiligt, wird der Reward aufgeteilt: Jeder der n_j Jäger innerhalb der *Fangzone*, also in einem definierten Umkreis des Fangorts, erhält eine Beloh-

nung von $\frac{10}{n_j}$. Es gibt keine weiteren Rewards und kein *Reward Shaping*, um (menschlichen) Bias zu minimieren.

3.5 Studie 1: Nachhaltiges Verhalten

Ziel der Studien ist, die Umstände zu untersuchen, unter denen Koordination und Kooperation als emergente Effekte unter eigennützigen, lernenden Agenten entstehen. Dafür werden in dieser ersten Studie zunächst die Komponenten und ihr Zusammenwirken analysiert: die grundlegende Parametrisierung der Umgebung, die Wahl und Parametrisierung des RL-Algorithmus sowie den Ablauf des Trainings. Ziel dieser ersten Studie ist, dass ein einzelner Agent nicht nur kurzfristig (reaktives), sondern auch langfristig optimales (proaktives) Verhalten erlernt. Abstrakt entspricht dies der exklusiven und intensiven Nutzung einer selbsterneuernden Ressource, ohne diese vollends zu erschöpfen. Im vorliegenden Szenario bedeutet dies, so viel Beute wie möglich, aber niemals die gesamte Population zu fangen.

3.5.1 Experimentelles Setup

In den folgenden Experimenten wird die Strategie eines einzelnen Jägers mit DQN [98] erlernt. Ziel dieses RL-Jägers ist, in begrenzter Zeit (einer Episode) möglichst viel Beute zu fangen. Die unmittelbare Herausforderung dabei ist, die Dynamik der Steuerung zu erlernen und effizient Beute zu fangen. Die langfristige Herausforderung besteht darin darin, eine weitsichtige Jagdstrategie zu finden. Fängt der RL-Jäger die komplette Beute direkt zu Beginn einer Episode, kann er für den Rest der Episode nichts mehr fangen. Verschont der RL-Jäger jedoch mindestens eine Beute und hält Abstand, kann sich diese reproduzieren und ermöglicht dem Jäger, langfristig mehr Beute zu fangen.

Die Beute wird durch Heuristiken gesteuert und wählt in dieser Studie immer die maximale lineare Beschleunigung. Die Fluchtstrategie bestimmt die Wahl der Orientierung. Beute kann entweder als Schwarm oder individuell fliehen. Die Untersuchungen von Hahn et al. [56] haben gezeigt, dass Beute eine höhere Überlebenschance hat, wenn sie keinen Schwarm bildet, sondern unabhängig voneinander flieht. Die vorliegende Arbeit konnte dies experimentell bestätigen. Die entsprechenden Ergebnisse sind in Abbildung 3.3 zu sehen. Daher wird beim Training des RL-Jägers individuell fliehende Beute verwendet.

In ersten Tests wurde ebenfalls beobachtet, dass der RL-Jäger erst dann Strategien mit weitsichtigem Verhalten entwickelt, wenn die Grundlagen der Steuerung erlernt waren. Inspiriert von populationsbasierten Ansätzen [68, 149] wird das Training in zwei aufeinander folgende Phasen unterteilt (siehe Abbildung 3.2), in dem der Agent die Erfahrung aus der ersten Phase in die

DQN	Parameter	Wert
Algorithmus		
	Aktionsraum	diskret
	Aktionswahl	ϵ-greedy
	Verringerung der Exploration	linear, pro Schritt
	Startwert	1,0
	Endwert	0,001
	Endwert erreicht bei Episode	2.500
	Diskontierungsfaktor γ	$0{,}970 - 0{,}999$
Architektur		
	Dimension innere MLP Schicht 1	32
	Dimension innere MLP Schicht 2	16
	Aktivierungsfunktion	ELU
	Loss Funktion	Huber
	Optimierer	Adam
	Max. Lernrate des Optimierers	0,005
Training		
	Dauer in Episoden	20.000
	Schritte pro Episode	500
	Bufferkapazität in Schritten	50.000
	Schritte bis Target Update	10.000
	Schritte pro Mini Batch	32

Tabelle 3.2: Details und Parameter der DQN-Implementierung dieses Kapitels.

zweite übernimmt. Im Training des RL-Jägers mit DQN wird eine Historie von Beobachtungen verwendet, d.h. der Agent enthält je Zeitschritt nicht nur die aktuellste, sondern eine Anzahl vorhergehender Beobachtungen τ. Die zur Reproduktion der Ergebnisse erforderlichen Details und Parameter sind in Tabelle 3.2 dargestellt. DQN gewichtet künftig zu erwartende Belohnungen mit dem Faktor γ, der in den Experimenten zwischen 0,970 und 0,999 variiert. Wenn nicht anders angegeben ist γ auf 0,990 gesetzt.

In der ersten Trainingsphase soll der RL-Jäger lernen, Beute zu fangen. Um die Exploration zu erleichtern, ist die Bewegung der Beute initial blockiert, ihre Anzahl wird zufällig zwischen 1 und 10 gewählt und die Reproduktion ist deaktiviert. Mit fortschreitendem Training wird die Geschwindigkeit der Beute alle 1000 Episoden erhöht. Der RL-Jäger ist also zunächst mit unbeweglichen Zielen konfrontiert und muss sein Verhalten bis zum Ende der ersten Phase an immer schneller fliehende Beute anpassen.

In der zweiten Trainingsphase soll der RL-Jäger lernen, die Anzahl der verbleibenden Beute bei der Jagd zu berücksichtigen und langfristig optimal zu

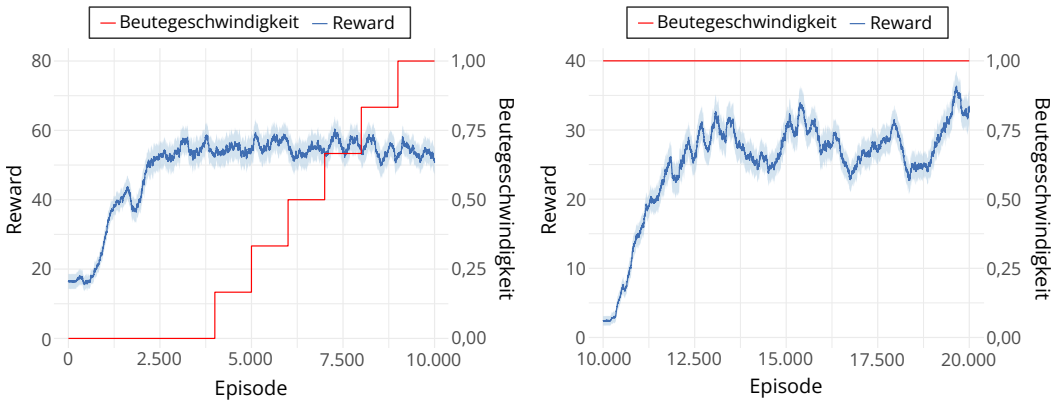


Abbildung 3.2: Trainingsverlauf eines RL-Jägers mit dem Algorithmus DQN. In der ersten Phase des Trainings (links) lernt der Agent, nicht reproduzierende Beute zu fangen, die mit zunehmendem Trainingsverlauf schneller flieht. In der darauf folgenden, zweiten Phase des Trainings (rechts) lernt der Agent, die initial vorhandene Beute zu verschonen, bis diese eine größere Population bildet. So kann der Agent langfristig mehr Beute fangen. Darstellung entnommen aus [120], Beschriftungen übersetzt.

handeln. Daher ist die Anzahl der initial erzeugten Beute auf 3 begrenzt. Die Reproduktion der Beutetiere ist aktiviert und die Geschwindigkeit der Beute wird nicht verringert. In diesem Szenario könnte der RL-Jäger die initiale Beute vollständig fangen, bevor sie sich reproduziert. Da sich Beute abweichend von der ersten Phase nun reproduziert, besteht die optimale Strategie darin, die Beute nie vollständig zu jagen, sondern zeitweise abzuwarten, bis sich die Population erholt hat. Als entscheidend für den Erfolg der zweiten Trainingsphase hat sich herausgestellt, dass eine Episode nicht endet, wenn die gesamte Beute gefangen ist, und ein RL-Jäger, der zu Beginn der Episode gierig handelt, indem er die gesamte Beute fängt, für den Rest der Episode ohne weitere Beute verbleibt.

3.5.2 Szenarien

Die Evaluation folgt auf die zweite Trainingsphase. Dafür werden ein Basisszenario und zwei Szenarien für Ablationsstudien definiert. Das Basisszenario ist gemäß Tabelle 3.1 parametrisiert und folgt auf die zweite Trainingsphase. Hier gibt es initial stets nur eine einzelne Beute. Diese bewegt sich mit voller Geschwindigkeit und kann sich reproduzieren. Mit diesem Szenario wird getestet, ob der RL-Jäger in der Lage ist, die Jagd erst dann zu beginnen, wenn sich die Beutepopulation durch Reproduktion stabilisiert hat.

In der ersten Ablationsstudie wird die Anzahl der anfänglichen Beute gegenüber dem Basisszenario schrittweise von 1 auf 10 erhöht. Zusätzlich wird die

Länge der Episode auf 1000 Schritte erhöht und die Reproduktion der Beute wird deaktiviert, sodass ein Jäger ausreichend Zeit hätte, sämtliche Beute zu fangen. Damit wird untersucht, wie die Anzahl der Beute das Verhalten des trainierten RL-Jägers beeinflusst.

In der zweiten Ablationsstudie wird die Anzahl der Schritte, bis sich die initiale Beute reproduziert, gegenüber dem Basisszenario erhöht. Gemessen wird die Anzahl der Schritte, bis der RL-Jäger seinen ersten Fang macht. Damit wird getestet, wie sich eine Veränderung der Reproduktionsrate der Beute auf das Verhalten des trainierten RL-Jägers auswirkt.

Es werden stets 20 Agenten pro Hyperparameterkombination unabhängig voneinander mit unterschiedlichen Seeds trainiert. Evaluiert wird entweder der beste Agent oder die besten 10 der insgesamt 20 Agenten (dies geht aus der jeweiligen Beschreibung hervor). Jede Evaluation erfolgt über 300 Episoden. Berichtet werden Mittelwert und 95% Konfidenzintervall berechnet über alle jeweiligen Episoden und Agenten.

Um die Leistung des RL-Jägers einordnen zu können, werden zum Vergleich die folgenden vier Heuristiken herangezogen:

1. *random* wählt Aktionen stets zufällig.
2. *static* verfolgt stets die Beute in kürzester Entfernung und ignoriert die Anzahl verbleibender Beute.
3. *static-rand* verfolgt entweder die nächstgelegene Beute oder wählt eine zufällige Aktion, beide mit einer Wahrscheinlichkeit von 50% (wenn nicht anders angegeben).
4. *static-wait* verfolgt nur dann die Beute in kürzester Entfernung, wenn es mindestens eine weitere Beute gibt und bleibt ansonsten stehen.

3.5.3 Auswertung

Abbildung 3.3 vergleicht die Leistung des RL-Jägers mit den zuvor beschriebenen Heuristiken bei unterschiedlich fliehender Beute im Basisszenario. Gemessen an der Anzahl gefangener Beute ist die Leistung des RL-Jägers nach Absolvierung der ersten Trainingsphase gleichauf mit der *static* Heuristik, die gierig handelt. Die *static-wait* Heuristik ist erfolgreicher als die anderen Heuristiken, was zeigt, dass es sich in diesem Szenario langfristig lohnt, die Beute nicht auszurotten. Nach Absolvierung der zweiten Trainingsphase übertrifft der RL-Jäger die ausgewählten Heuristiken. Dies zeigt, dass die *static-wait* Heuristik, die immer genau eine Beute verschont, in diesem Szenario nicht optimal ist. Bemerkenswert ist, dass die *static-rand* Heuristik, die 50% der Aktionen zufällig wählt, erfolgreicher als die gierige *static* Heuristik handelt. Dies wird in Abbildung 3.4 detaillierter analysiert. Hier wird deutlich, dass der Jagderfolg der *static-rand* Heuristik zwar bei 70% zufälligen Aktionen am höchsten

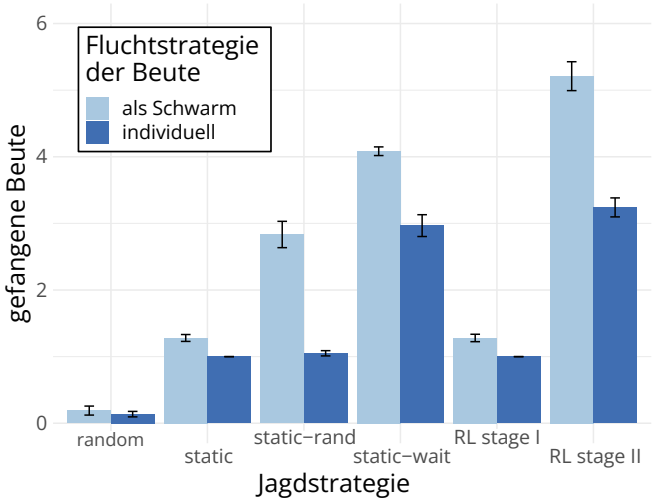


Abbildung 3.3: Vergleich des besten aus 20 RL-Jägern mit verschiedenen Heuristiken bei unterschiedlich fliehender Beute. Mehr gefangene Beute ist besser. Dieses Szenario beginnt mit einer einzigen, reproduzierenden Beute. Darstellung entnommen aus [120], Beschriftungen übersetzt.

ist, dann jedoch auch eine hohe Varianz vorliegt. Abbildung 3.5 visualisiert die Ergebnisse der ersten Ablationsstudie, die die Anzahl der zu Beginn einer Episode vorhandenen Beute mit der Anzahl der am Ende einer Episode verbleibenden Beute vergleicht. Der RL-Jäger verschont hier immer mindestens eine Beute.

Abbildung 3.6 stellt die Ergebnisse der zweiten Ablationsstudie dar und stellt die Anzahl der Schritte, bis sich die Beute reproduziert, sowie die Anzahl Schritte, bis der RL-Jäger das erste Mal Beute fängt, gegenüber. Je länger es dauert, bis sich die Beute reproduziert, umso länger dauert es, bis der RL-Jäger das erste Mal Beute fängt.

Abbildung 3.7 zeigt die Ergebnisse einer Rastersuche im Basisszenario, welche Länge der Historie τ und welche Gewichtung künftig zu erwartender Rewards γ unter ansonsten gleichen Trainingsbedingungen zum besten RL-Jäger führt. Die linke Grafik misst dies mit der Anzahl der gefangenen Beute. Bei $\gamma = 0{,}99$ führt $\tau = 20$ zu etwa 2,4 gefangenen Beuteagenten. Eine Verringerung von τ führt zu einem Rückgang der gefangenen Beute bis 1, und eine Erhöhung von τ bewirkt einen starken Rückgang der gefangenen Beute gegen 0. Dasselbe lässt sich bei $\gamma = 0{,}999$ beobachten, wobei $\tau = 1$ das beste Ergebnis erzielt, während bei $\gamma = 0{,}97$ ein τ zwischen 20 und 40 die besten Ergebnisse ermöglicht. Weitere Erkenntnisse liefert die rechte Grafik, die die Anzahl der Beutereproduktionen misst. Bei geringem τ reproduziert sich die Beute selten, was darauf hindeutet, dass der RL-Jäger hier die initiale Beute sofort fängt. Bei großem τ treten bis zu 10 Reproduktionen der Beute auf, was darauf hindeutet, dass der RL-Jäger hier

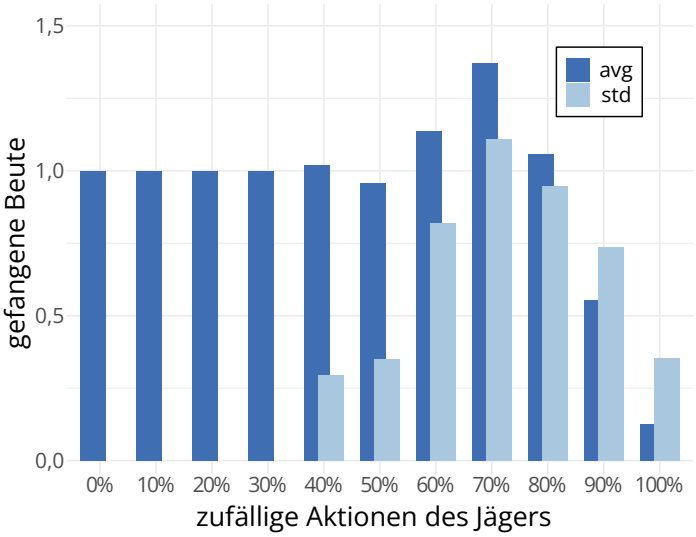


Abbildung 3.4: Auswirkung von zufälligen Aktionen auf den Jagderfolg der *static-rand* Heuristik. Initial ist eine Beute vorhanden, die sich reproduzieren kann. Darstellung entnommen aus [120], Beschriftungen übersetzt.

gar keine Beute fängt. Insgesamt erzielt der mit DQN in zwei Phasen trainierte RL-Jäger die meisten Fänge, wenn sich die Beute etwa 4 mal reproduziert.

3.5.4 Diskussion

Zunächst zeigen die Ergebnisse, dass die verwendete Umgebung geeignet ist, um *reaktives* und *proaktives* Verhalten [160] zu untersuchen. Die Heuristiken demonstrieren, dass im Basisszenario nur Jagdstrategien, die bis zur ersten Reproduktion der Beute einen Mindestabstand einhalten, langfristig zu mehr als einem Fang führen. Nach Absolvierung der ersten Trainingsphase verhält sich der RL-Jäger reaktiv und vergleichbar zur kurzsichtigen, gierigen Heuristik. Nach Abschluss der zweiten Trainingsphase ist der RL-Jäger in der Lage, proaktiv und langfristig optimal zu handeln, und er übertrifft dabei alle Heuristiken unabhängig von der Überlebensstrategie der Beute. Die Beobachtung, dass individuell fliehende Beute schwieriger zu jagen ist als Beute, die im Schwarm flieht, steht in Einklang mit früheren Arbeiten [55, 56] und unterstreicht die Plausibilität der vorliegenden Ergebnisse.

Interessant ist, dass der RL-Jäger langfristig etwas erfolgreicher agiert als die *static-wait* Heuristik, die stets exakt eine Beute verschont. Dies ist ein Beispiel dafür, dass es lohnenswert sein kann, mit RL nach dem Optimum einer Lösung zu suchen, selbst wenn eine (mutmaßlich) geeignete Heuristik für ein komplexes Problems bekannt ist. Erfreulich ist, dass dies im vorliegenden Fall ohne Reward Shaping gelingt, da so ausgeschlossen werden kann, dass menschlicher Bias die Agenten in ein lokales Optimum geführt hat. Umgekehrt ist dies

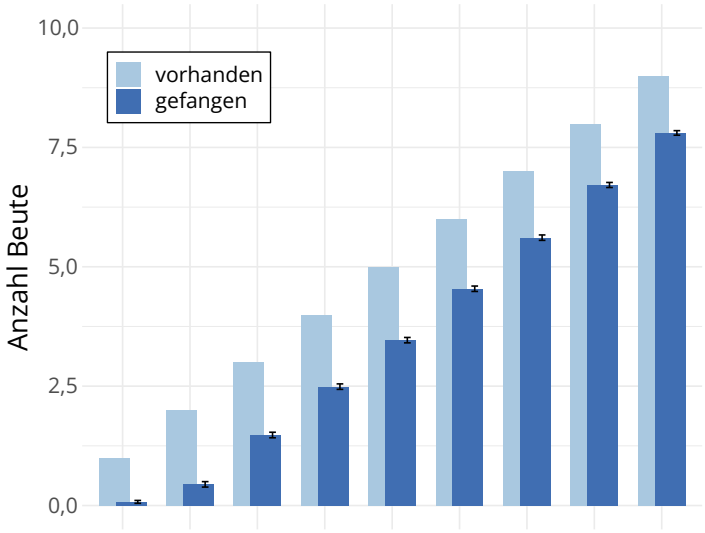


Abbildung 3.5: Gegenüberstellung von initial vorhandener und im Verlauf der Episode gefangener Beute. Evaluiert wird der beste aus 20 trainierten RL-Jägern. Die Beute kann sich nicht reproduzieren. Darstellung entnommen aus [120], Beschriftungen übersetzt.

jedoch kein Beleg dafür, dass die Agenten tatsächlich das globale Optimum gefunden haben. Die Ergebnisse zeigen aber auch, dass das Ergänzen zufälliger Aktionen, bei denen die Chance besteht, dass die Beute von Zeit zu Zeit verschont wird, nicht zu vergleichbaren Ergebnissen führt. Für eine gute Lösung benötigt dieses Szenario also adaptives Verhalten, das über zeitweise nicht perfektes Verhalten hinaus geht.

Die Ergebnisse der beiden Ablationsstudien legen nahe, dass der RL-Jäger gelernt hat, entsprechend zu handeln. Der aktuelle Zeitpunkt innerhalb der Episode und der Zeitpunkt, an dem sich die Beute reproduziert, ist für den RL-Jäger nicht (a priori) ersichtlich. Trotz deaktivierter Beutereproduktion verschont der Jäger stets ein Individuum, mutmaßlich in Erwartung einer künftigen Vergrößerung der Population, wie er sie in der zweiten Trainingsphase beobachtete. Im entgegengesetzten Szenario mit Beutereproduktion nach variabler Zeit zeigt sich, dass der RL-Jäger nicht nur gelernt hat, eine bestimmte Anzahl von Schritten abzuwarten. Er beginnt erst zu jagen, wenn die Beutepopulation eine stabile Größe (mehr als ein Individuum) erreicht hat.

Es ist erfreulich, dass eine verglichen mit der Literatur (siehe Kapitel 3.3.4) einfache Netzwerkarchitektur und Training aus zwei Phasen ausreichen, um dieses Ergebnis zu erzielen, weil dies entsprechend wenig Rechenleistung benötigt. Gleichermaßen ist es wichtig, zu beachten, dass nicht jeder Trainingslauf erfolgreich war und entweder nur der beste RL-Jäger oder die besten 10 aus 20 trainierten RL-Jägern zur Evaluation herangezogen wurden.

Hinsichtlich der bekannten Literatur ist die systematische Analyse des Ein-

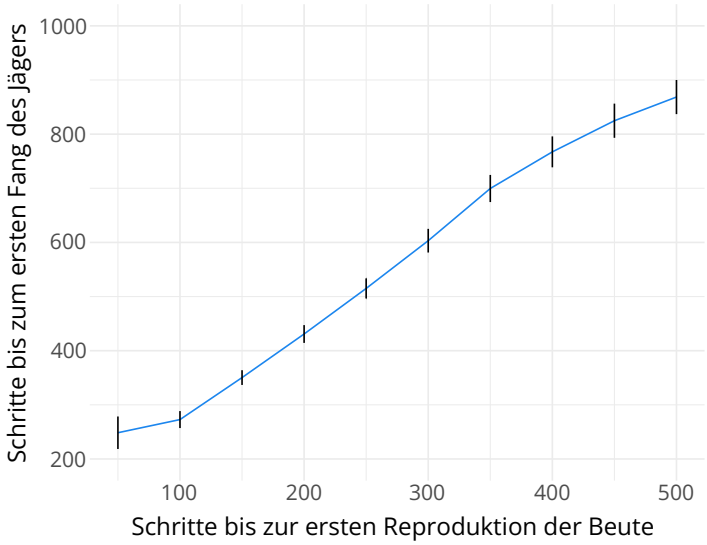


Abbildung 3.6: Korrelation der Anzahl Schritte, bis sich Beute erstmalig reproduziert, und der Anzahl der Schritte, bis der Jäger den ersten Fang macht. Dargestellt ist der beste aus 20 trainierten RL-Jägern. Initial ist eine Beute vorhanden, die sich reproduzieren kann. Darstellung aus [120], Beschriftungen übersetzt.

flusses von γ und τ auf den Lernerfolg des RL-Jägers neu. Dies deutet darauf hin, dass ausgehend von einer NN-Architektur samt Parametrisierung nur ein schmaler Pfad zum Erfolg führt: weder funktioniert kurzfristiges Schätzen des künftigen Nutzens in Kombination mit wenigen Informationen (kleine Werte für γ, τ), noch funktioniert ein langfristiges Schätzen des künftigen Nutzens kombiniert mit zu vielen Informationen (große Werte für γ, τ).

3.6 Studie 2: Kooperation und Koordination

Ziel der Studien ist, die Umstände zu untersuchen, unter denen Koordination und Kooperation als emergente Effekte unter eigennützigen, lernenden Agenten entstehen. Die vorherige Studie untersuchte die Komponenten, ihr Zusammenwirken und wann ein einzelner Agent nicht nur kurzfristig (reaktives), sondern auch langfristig optimales (proaktives) Verhalten mit RL lernt. Die folgende Studie legt der Fokus auf den Umstände, unter denen mehrere Agenten einen koordinierten und kooperativen (sozialen) Umgang mit einer gemeinsamen Ressource mit RL lernen. Besonderes Augenmerk liegt auf der potentiellen Tragik der Allmende und dem Einfluss zusätzlichen Überlebensdrucks, der mit einer Hungermechanik abbildet wird.

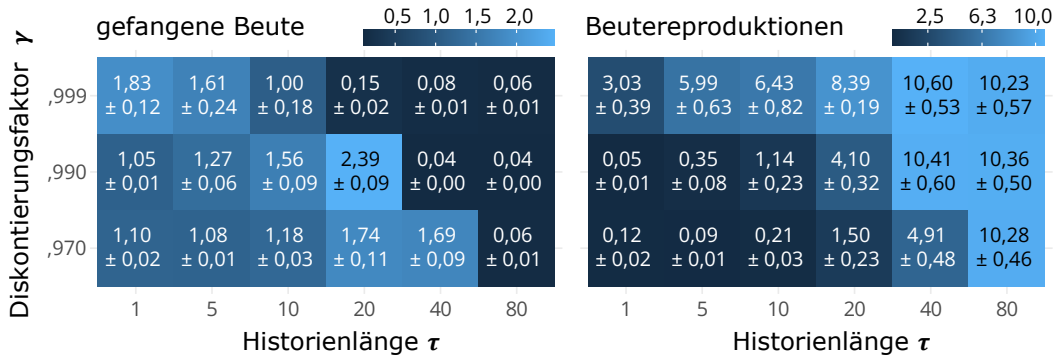


Abbildung 3.7: Einfluss künftiger Rewards (γ) und vergangener Beobachtungen (τ) auf den Erfolg (gefangene Beute) des RL-Jägers. Evaluiert werden die besten 10 aus 20 pro Kombination trainierten Jägern. Darstellung basierend auf [120], Beschriftungen übersetzt und Layout angepasst.

3.6.1 Experimentelles Setup

Die Umgebung der vorherigen Studie bildet die Grundlage für die folgenden Experimente, jedoch werden einige Details verändert. Einerseits zeigte die vorherige Studie, dass Beute eine höhere Überlebenschance hat, wenn sie keinen Schwarm bildet, sondern unabhängig voneinander flieht. Daher wird dies als rationale Strategie betrachtet und für alle folgenden Experimente verwendet. Beute flieht nur dann, wenn sie ein Raubtier wahrnimmt, und bleibt ansonsten stehen. Neu ist außerdem, dass mehrere Jäger in derselben Umgebung lernen. Daher müssen sie sich die verfügbare Beute teilen. Zusätzlich werden Jäger einer Hungermechanik ausgesetzt: Fangen sie zu lange keine Beute, verhungern sie und werden für den Rest der Episode aus der Simulation entfernt.

In der vorhergegangenen Studie wurde aus mehreren trainierten Agenten selektiert, um Individuen mit dem gewünschten Verhalten zu finden. Da in dieser Studie mehrere Agenten gleichzeitig trainiert werden und ein Scheitern einzelner Agenten das Scheitern aller Agenten zur Folge haben kann, ist eine Selektion kein effizientes Vorgehen. Gemäß der Literatur ist zu erwarten, dass der RL-Algorithmus *PPO* [132] bessere Ergebnisse als DQN erzielt. Daher werden RL-Jäger in dieser Studie mit PPO realisiert. Die verwendete Implementierung basiert auf der Bibliothek *OpenAI-Baselines* [34]. Die Implementierung wurde so modifiziert, dass mehrere RL-Agenten parallel individuelles Verhalten lernen können. Im MARL wird dieses Vorgehen als *Independent Learning* (IL) bezeichnet. Die zur Reproduktion der Ergebnisse erforderlichen Details und Parameter sind in Tabelle 3.3 aufgelistet.

Um das Szenario in Gegenwart mehrerer Jäger beurteilen zu können, wird zunächst die Metrik überarbeitet, die das langfristig optimale Verhalten misst. Die **Nachhaltigkeit** σ ist die Summe der innerhalb einer Episode e gefangenen Beute b_e^- und der am Ende dieser Episode lebenden (verschonten) Beute b_e^+,

PPO	Parameter	Wert
Algorithmus		
	Aktionsraum	kontinuierlich
	Wahrscheinlichkeitsverteilungen	gelernt
	Entropiebonus	nein
	Vorteilsfunktion	GAE
	Überblendungsparameter λ	0,95
	Clipping Bereich	0,1
	Gewichtung des *Value Loss*	0,5
	Diskontierungsfaktor γ	0,99
Architektur		
	Dimension innere MLP Schicht 1	64
	Dimension innere MLP Schicht 2	64
	Dimension innere MLP Schicht 3	64
	Aktivierungsfunktion	tanh
	Optimierer	Adam
	Lernrate des Optimierers	0,00009
Training		
	Dauer in Episoden	7.000
	Schritte pro Episode	3.000
	Bufferkapazität in Schritten	2.048
	Anzahl Mini Batches	4
	Durchläufe des Optimierers	4

Tabelle 3.3: Details und Hyperparameter der PPO Implementierung

gemittelt über Anzahl $|E|$ aller Episoden E der Evaluation:

$$\sigma = \frac{1}{|E|} * \sum_{e \in E} b_e^+ + b_e^- \tag{3.1}$$

Dies berücksichtigt, dass es in Gegenwart mehrerer Jäger langfristig erforderlich sein könnte, anstatt einer einzelnen Beute eine größere Population zu verschonen. Die Hungermechanik und ein Limit der Beutepopulation von 11 verhindern, dass Jäger eine übermäßig hohe Nachhaltigkeit erreichen, indem sie dem exponentiellen Wachstum der Beutepopulation freien Lauf lassen, ohne zu jagen. Das grundsätzliche Ziel ist jedoch, die Jäger auf globaler Ebene hinsichtlich emergenter Koordination und Kooperation zu untersuchen. Die grundlegende Herausforderung in diesem Szenario ist, dass sich alle Jäger bei niedriger Beutepopulation implizit darauf einigen müssen, die Verfolgung der verbleibenden Beute abzubrechen und Distanz zu halten (außerhalb des Sichtradius der Beute zu bleiben), damit sich diese wieder reproduziert. In ersten

Tests wurden drei verschiedene Resultate beobachtet:

1. Einzelne Jäger nähern sich dennoch der letzten Beute an oder fangen diese sogar. Dann gibt es zu wenig oder keine Beute und alle Jäger verhungern (Tragik der Allmende).
2. Jäger fangen zu wenig Beute und verhungern. Die Beute vermehrt sich bis zum Populationslimit.
3. Gruppenjagd ist in der Standardparametrisierung nicht zu beobachten. Jäger fangen und verfolgen Beute stattdessen alleine.

Überleben die Jäger bis zum Ende der Episode, indem sie erfolgreich jagen, ohne dass die Tragik der Allmende eintritt, wird dies als *koordiniertes* Verhalten gewertet. Jedoch wird ihr Verhalten erst dann als *kooperativ* gewertet, wenn sie gemeinsam die gleiche Beute jagen, beispielsweise indem sie diese von mehreren Seiten einkreisen (siehe Abbildung 3.8). Dies weicht von der Spieltheorie [8] und dem AMAS Konzept [47] ab, die beide nur zwischen kooperativen und nicht kooperativen Situationen unterscheiden. Die vorliegende Arbeit betrachtet aber sowohl Koordination, als auch Kooperation als Form der Selbstorganisation. Darüber hinaus setzt Kooperation in diesem Szenario Koordination voraus: Für einen kooperativen Fang müssen sich die Jäger nicht nur einig sein, wann sie jagen, sondern auch, welche Beute sie jagen. Verhungern die Jäger, weil sie zu wenig oder zu viel jagen (und die Tragik der Allmende eintritt), wird die Selbstorganisation als gescheitert betrachtet. Insgesamt werden also Metriken benötigt, um drei Situationen voneinander zu unterscheiden, die folgendermaßen definiert sind.

Die **Misserfolgsrate** ϕ ist das Verhältnis der Anzahl nicht erfolgreicher Episoden zur Anzahl $|E|$ aller Episoden der Evaluation E. Nicht erfolgreich ist eine Episode e, wenn die Koordination gescheitert ist. Dies ist einerseits der Fall, wenn sämtliche Beute gefangen wird ($b_e^+ = 0$), und andererseits, wenn die Beutepopulation zum Ende der Episode das Limit erreicht ($b_e^+ = b^{max}$):

$$\phi = \frac{|\, e \in E \colon b_e^+ = 0 \vee b_e^+ = b^{max} \,|}{|E|} \tag{3.2}$$

Die **Koordinationsrate** η ist das Verhältnis der Anzahl von Episoden, in denen die Koordination entstanden ist, zur Anzahl $|E|$ aller Episoden der Evaluation E. Koordination ist entstanden, wenn die Beutepopulation b_e^+ am Ende einer Episode e größer als 0 und kleiner als b^{max} ist:

$$\eta = \frac{|\, e \in E \colon b_e^+ > 0 \wedge b_e^+ < b^{max} \,|}{|E|} = 1 - \phi \tag{3.3}$$

Die **Kooperationsrate** κ ist das Verhältnis der Anzahl kooperativer Fänge zur Anzahl aller Fänge F_e einer Episode e, gemittelt über die Anzahl $|E|$ aller Episoden der Evaluation E. Ein Fang f ist dann kooperativ, wenn sich alle

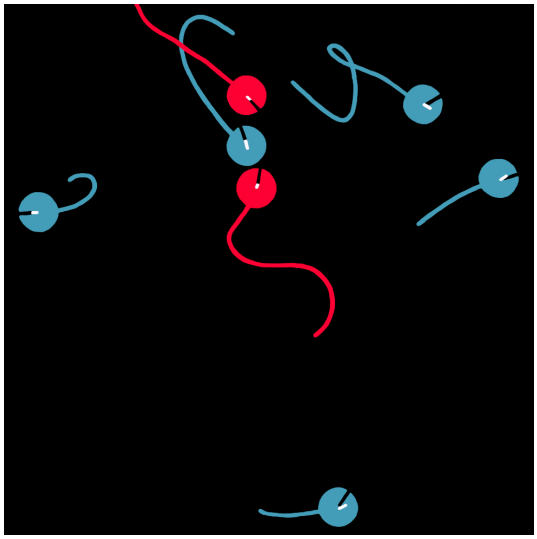


Abbildung 3.8: Zwei mit RL trainierte Jäger (rot) kreisen eine Beute (cyan) von zwei Seiten ein, was als kooperatives Verhalten eingeordnet wird. Die Linien stellen die Trajektorieren der Agenten dar. Darstellung mit angepassten Farben übernommen aus [124].

Jäger $j \in J$ zum Zeitpunkt dieses Fangs innerhalb des Radius $r \in \mathbb{R}$ der *Fangzone* um die gefangene Beute b_f^- befinden:

$$\kappa = \frac{1}{|E|} * \sum_{e \in E} \frac{|\, f \in F_e \colon \forall j \in J \colon dist(j, b_f^+) < r \,|}{|F_e|} \tag{3.4}$$

Die Heuristiken und das Training von DQN in zwei Phasen sind in Unterkapitel 3.5.2 genauer beschrieben. Für DQN werden in dieser Studie die Hyperparameter verwendet, die in der vorherigen Studie die höchste durchschnittliche Anzahl gefangener Beute pro Episode ergaben ($\gamma = 0{,}99$, $\tau = 20$). Mit Ausnahme des ersten Szenarios (siehe Abbildung 3.9) dauern Episoden in Training und Evaluation jeweils 3.000 Schritte. PPO wird mit konstanten Hyperparametern für 7.000 Episoden trainiert, also in insgesamt 21 Millionen Schritten. Wenn nicht anders angegeben, werden die Standardparameter der Umgebung verwendet (siehe Tabelle 3.1). Wie zuvor werden stets 20 Agenten pro Hyperparameterkombination und Szenario unabhängig voneinander mit unterschiedlichen Seeds trainiert. Aufgrund der längeren Episoden wird die anschließende Evaluation auf 20 Episoden pro Agent und Szenario verringert. Agenten werden unter denselben Bedingungen evaluiert, unter denen sie trainiert wurden. Berichtet werden Mittelwert und 95% Konfidenzintervall über alle jeweiligen Episoden und Agenten. Anders als zuvor wird nicht mehr selektiert, sondern die Ergebnisse aller trainierten Agenten gemittelt.

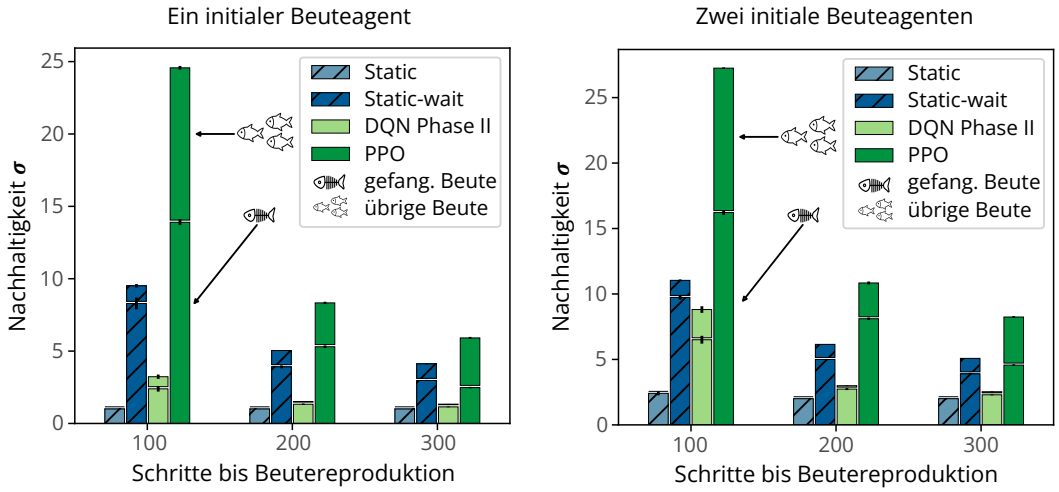


Abbildung 3.9: Nachhaltigkeit σ (Summe gefangener und verschonter Beute pro Episode) während der Evaluation, mehr ist besser. Anders als zuvor werden die Ergebnisse aller Agenten gemittelt. Deshalb schneidet DQN schlechter ab als die adaptive Heuristik *static-wait*. Darstellung basierend auf [124], Layout angepasst, Beschriftungen übersetzt.

3.6.2 Auswertung

Um die Brücke zur vorherigen Studie zu schlagen und die Eignung von PPO zu prüfen, wird zunächst ein einzelner Jäger in Episoden mit 1.000 Schritten trainiert. Wie zuvor ist es erforderlich, die initiale Beute zu verschonen, bis diese sich reproduziert, um langfristig mehr Beute fangen zu können. In Abbildung 3.9 sind die Ergebnisse des Vergleichs von DQN, PPO und den Heuristiken *static* und *static-wait* dargestellt. Gemessen an der Nachhaltigkeit σ erzielt DQN schlechtere Ergebnisse als PPO und kann teils nicht mit der Heuristik *static-wait* mithalten. Insbesondere ist unter den für die zweite Studie trainierten DQN-Agenten keiner, der auch bei langsamerer Reproduktion zuverlässig Beute verschont. Die PPO Agenten hingegen erzielen zuverlässig und mit Abstand das höchste σ. Sie fangen bis auf einen Fall die meiste Beute und erhalten gleichzeitig die größte Population. Dies gilt insbesondere dann, wenn sich Beute langsamer reproduziert. Die Heuristiken funktionieren wie vorgesehen: *static* verschont keine Beute und *static-wait* immer genau eine. Eine weiterer Vorteil von PPO gegenüber DQN ist die vereinfachte Trainingspipeline: DQN benötigt zwei Trainingsphasen, PPO nur eine.

PPO bildet die Grundlage für das Training mehrerer Agenten. Dafür kommt zunächst das MARL Paradigma *Independent Learning* (IL) zum Einsatz, bei dem jeder Agent eine individuelle Strategie lernt. Abbildung 3.10 veranschaulicht den Lernprozess zweier Jäger mit PPO, die gemeinsam drei Stadien durchlaufen: (**a**) unfähig zu jagen, (**b**) kurzsichtig zu jagen und (**c**) nachhaltig zu

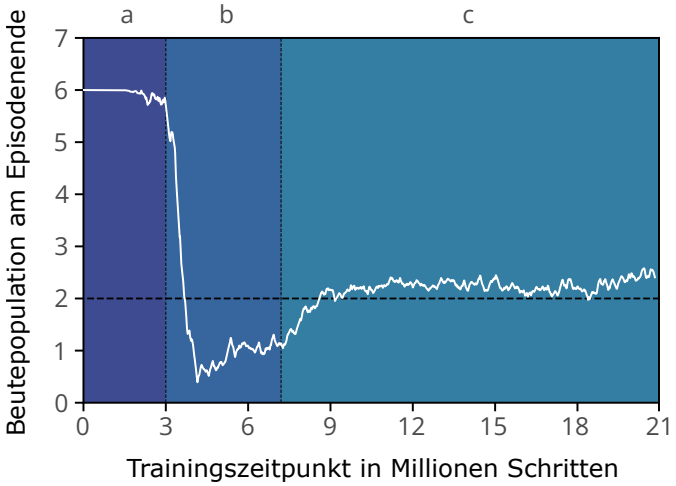


Abbildung 3.10: Verbleibende Beute am Ende jeder Episode während dem Training zweier Jäger mit PPO. Die Beutepopulation ist auf 6 limitiert. Es können drei Verhaltensweisen beobachtet werden. In der ersten Phase **a** (links) fangen die Jäger so wenig Beute, dass zum Ende der Episode das Populationslimit erreicht wird. In der zweiten Phase **b** (mitte) jagen sie so viel, dass zeitweise keine Beute übrig bleibt. In der dritten Phase **c** (rechts) erhalten sie eine Beutepopulation von etwa zwei Individuen. Darstellung aus [124], Beschriftungen übersetzt.

jagen. Ein weiterer Indikator dafür ist Abbildung 3.11 zu sehen. Die Beschleunigung der zwei PPO Jäger geht während dem Trainings von einer unimodalen Verteilung (stets volle Beschleunigung) zu einer bimodalen Verteilung (teils keine, teils volle Beschleunigung) über. Dies deutet darauf hin, dass die Jäger mit zunehmendem Training absichtlich warten, bis die Beutepopulation sich erholt, und dabei vermeiden, sich zu bewegen, um die Reproduktion der Beute nicht zu unterbrechen.

Abbildung 3.12 stellt zwei Schlüsselfaktoren für die Entstehung von Koordination unter zwei PPO Jägern dar. Erstens hat die Anzahl der initialen Beute maßgeblichen Einfluss: Je weniger vorhanden ist, umso höher ist die Koordinationsrate η. Möglicherweise ist hier entscheidend, wie lange die Jäger in Situationen sind, in der nur wenig oder gar keine Beute vorhanden ist. Eine kleine initiale Beutepopulation ist schneller gefangenen als eine große. Der Nutzen einer temporären Schonung der Beute bei geringer Population kommt bei wenig initialer Beute entsprechend schneller zum Tragen. Zweitens spielt die Hungermechanik eine wesentliche Rolle, die maßgeblich von der initialen Überlebenszeit der Jäger beeinflusst wird. Fangen Jäger innerhalb dieser Zeit keine Beute, verhungern sie. Die Variation der initialen Überlebenszeit zusätzlich zur

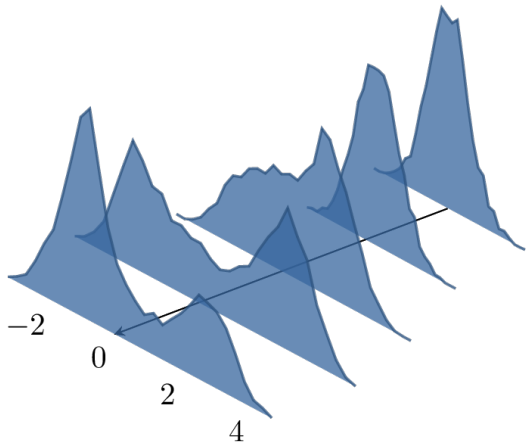


Abbildung 3.11: Von hinten nach vorne: Verteilung der Beschleunigung zu verschiedenen Zeitpunkten des Trainings. Die Verteilung wird zunehmend bimodal. Das deutet darauf hin, dass die beiden PPO-Jäger teils warten, bis sich die Beute reproduziert, und teils schnell bewegen (jagen). Darstellung aus [124].

initialen Beute führt zu dem Ergebnis, dass η bei wenig initialer Beute und mittlerer initialer Überlebenszeit am höchsten ist. Dies sind im vorliegenden Szenario die optimalen Voraussetzungen für das Entstehen von Koordination. Ebenso ist zu beobachten, dass eine geringe initiale Überlebenszeit die Jäger zwingt, unmittelbar zu jagen, um nicht unmittelbar zu verhungern und ihnen bei geringer initialer Beute daher die Chance lässt, nachhaltig zu handeln. Interessanterweise führen eine hohe Anzahl initialer Beute und eine hohe initiale Überlebensdauer nicht zu hoher Koordination gemäß der verwendeten Metrik. Möglicherweise bestand in beiden Fällen kein ausreichender Anreiz für die Jäger, ein entsprechendes Verhalten zu erlernen.

Der Fokus wechselt nun auf komplexe Kooperation in Form von Gruppenjagd. Leibo et al. realisieren das mit zusätzlichen Belohnungen für kooperative Fänge [78]. Die vorliegende Arbeit verändert stattdessen die äußeren Umstände, also die Parametrisierung der Umgebung. Zum einen wird die Beschleunigung der Jäger verringert, sodass die erreichbare Höchstgeschwindigkeit nicht höher und in einigen Szenarien sogar langsamer als die der Beute ist. Zum anderen wird die Begrenzungen der Umgebung entfernt, sodass Agenten, die den Raum auf der einen Seite verlassen, ihn auf spiegelverkehrter Seite wieder betreten. So kann Beute nicht mehr durch einzelne Jäger in eine Ecke getrieben werden. Jäger können Beute nur noch fangen, indem sie sich dieser gemeinsam gleichzeitig von mehreren Seiten nähern. Abbildung 3.8 zeigt, wie das in der Praxis erfolgt. Unter der Annahme, dass dann alle Jäger innerhalb eines bestimmten

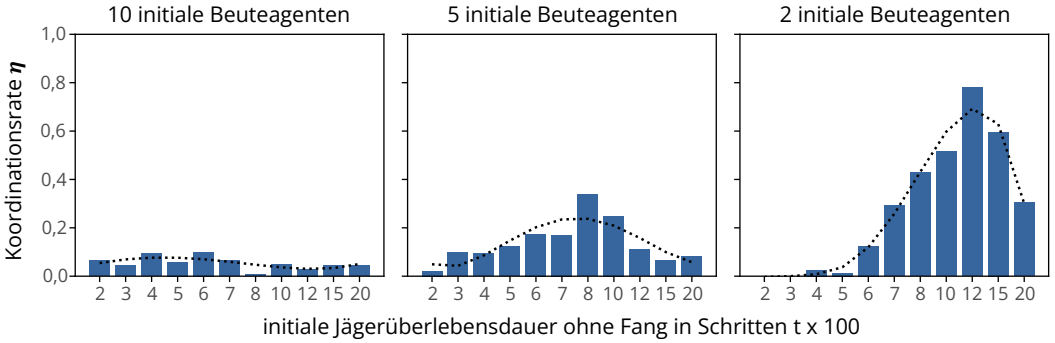


Abbildung 3.12: Einflussfaktoren auf die Koordinationsrate η zweier PPO-Jäger. Von links nach rechts wird die Anzahl initialer Beute verringert, was den Druck auf die Jäger durch die Hungermechanik erhöht. Zusätzlich wird die initiale Überlebensdauer der Jäger ohne Fang variiert, höhere Werte verringern den Druck. Jäger sind hier stets schneller als Beute. Die Kurven wurden mit einem Polynom vierten Grades berechnet. Darstellung entnommen aus [124], Beschriftungen übersetzt.

Radius effektiv zu einem Fang beigetragen haben, wird die *Fangzone* sowohl zur Messung der Kooperationsrate κ, als auch zur Aufteilung der Belohnung unter den beteiligten Jägern verwendet.

Abbildung 3.13 zeigt, dass ein größerer Radius der *Fangzone* und eine geringere Geschwindigkeit der Jäger zu einer höheren Kooperationsrate κ führen. Ähnlich wie bei der Koordination führt eine geringere Anzahl initialer Beute hier zu höherem κ. Dass weniger Beute zu mehr kooperativen Fängen führt, lässt sich auf zwei Faktoren zurückführen: es gibt weniger Beute, auf die sich die Jäger verteilen könnten, und die Beute kann leichter vor einzelnen Jägern fliehen, da sie weniger Hindernissen in Form anderer Agenten ausweichen muss. Ebenso zeigt Abbildung 3.13, dass die Gruppenjagd durch die Umgebung forciert werden muss. Während eine Beschleunigung von 0,03 (die Hälfte des Ausgangswertes) die Jäger praktisch zur Kooperation zwingt, führen höhere Werte zu deutlich weniger Kooperation.

Nicht alle Faktoren der Umgebung haben maßgeblichen Einfluss auf die Kooperation: Der Stunning-Mechanismus (Abbildung 3.13 rechts) ändert κ gegenüber dem Normalfall (Abbildung 3.13 mittig) kaum. Nicht in dieser Arbeit abgebildet ist die qualitative Beobachtung, dass Jäger bei der Jagd anders vorgehen, um Kollisionen miteinander zu vermeiden, mutmaßlich, da ein betäubter Partner bei der weiteren Jagd nicht hilfreich ist. Daraufhin wurden stichprobenartig weitere Experimente durchgeführt, um zu analysieren, ob ähnlich wie bei Leibo et al. [78] Konkurrenzdruck zu aggressiverem Verhalten führt und die Jäger Artgenossen gegebenenfalls bewusst betäuben, um die Beute nicht teilen zu müssen. Jedoch konnte keine signifikante Korrelation zwischen der Anzahl

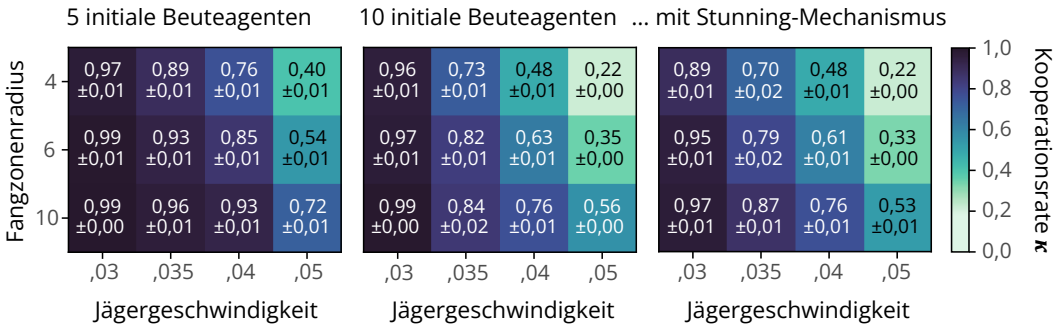


Abbildung 3.13: Einflussfaktoren auf die Kooperationsrate κ zweier PPO-Jäger. Von links zur Mitte wird erst die Anzahl initialer Beuteagenten erhöht, was die Jagd erleichtert, und dann wird rechts den Stunning-Mechanismus ergänzt. Anders als in der vorherigen Analyse der Koordination (siehe Abbildung 3.12) sind Jäger hier nie schneller als Beute. Darstellung basierend auf [124], Layout angepasst, Beschriftungen übersetzt.

betäubter Jäger und der Größe der Beutepopulation festgestellt werden. Daher wurde auf eine Rastersuche verzichtet und zur Kenntnis genommen, dass das vorliegende Szenario nicht zur Untersuchung von aggressivem Verhalten geeignet ist.

In Abbildung 3.15a ist zu sehen, dass eine kürzere initiale Überlebensdauer ohne Fang die Kooperationsrate κ der Jäger kaum verändert. Jäger mit einer geringen Geschwindigkeit erzielen die meisten kooperativen Fänge, jedoch zeigt Abbildung 3.15b, dass eine kurze initiale Überlebensdauer ohne Fang von 500 Zeitschritten und geringe Geschwindigkeit zu einer hohe Misserfolgsquote ϕ führen. Dieses Szenario lässt den Jägern also zu wenig Zeit, um zu lernen, erfolgreich Beute zu fangen. Die erzielten Fänge finden jedoch in Kooperation statt. Ansonsten ist ϕ insgesamt gering.

Ein weiterer wichtiger Einflussfaktor auf die Kooperation ist die Sichtweite der Beute. Eine Verdoppelung der Sichtweite der Beute von 10 auf 20 führt daher in 9 von 12 Szenarien zu annähernd vollständiger Kooperation (siehe Abbildung 3.14). Daraus wird abgeleitet, dass Beute, die weiter sehen kann, einzelnen Jägern leichter entkommt. Die Jäger haben dadurch einen größeren Vorteil von der Jagd in der Gruppe.

Nun werden die Experimente auf drei Jäger skaliert, wobei der Sichtradius der Beute auf 25 gesetzt ist. Ein Fang zählt hier nur dann als kooperativ, wenn alle drei Jäger beteiligt, also innerhalb des Fangzonenradius sind. In Abbildung 3.16 ist zu sehen, dass die Kooperationsrate κ selbst unter den günstigsten Kombinationen aus Jägergeschwindigkeit und Fangzonenradius nicht über 0,58 hinausgeht. Nicht in dieser Arbeit dargestellt sind Experimente mit einer weiteren Vergrößerung des Sichtradius der Beute und einer weiteren Verlangsamung der Jäger, die zu einer unpraktikabel großen Misserfolgsquote ϕ führten.

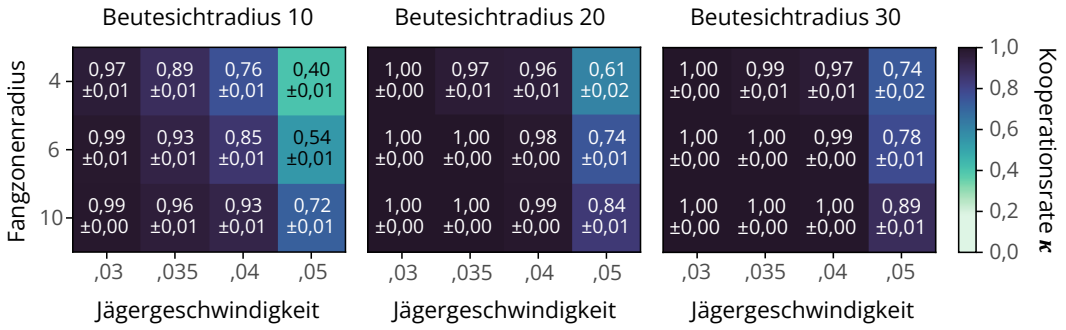


Abbildung 3.14: Einfluss des Sichtradius der Beute auf die Kooperationsrate κ zweier PPO-Jäger. Der Standardwert ist 10. Ein Sichtradius von 30 ermöglicht der Beute, Jäger überall in der Umgebung wahrzunehmen. Darstellung basierend auf [124], Layout angepasst, Beschriftungen übersetzt.

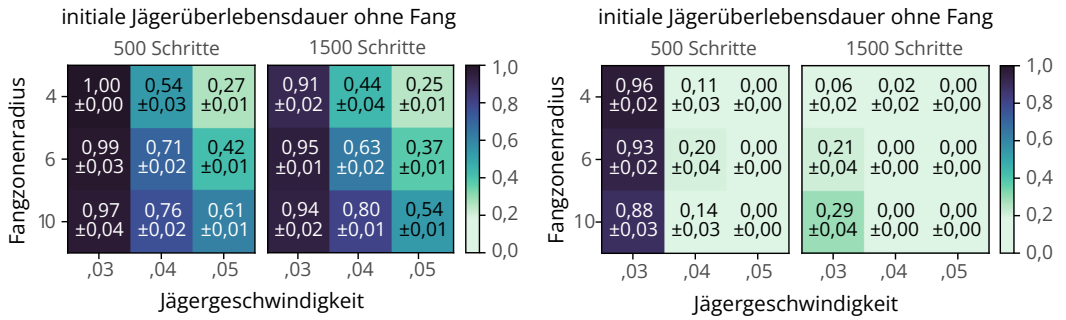
Eine höheres κ als 0,58 ist auf diesem Weg nicht zu erzielen. Insgesamt ist die Tendenz zu erkennen, dass ein größerer Fangzonenradius zu höherem κ führt. Ein Blick auf Abbildung 3.17 zeigt, dass die Anzahl gefangener Beute sich nicht signifikant ändert. Daraus wird abgeleitet, dass die Jagd im vorliegenden Szenario nicht zwingend drei Jäger erfordert, sondern bereits mit zwei Jägern erfolgreich sein kann.

Bislang wurden alle Experimente mit *Independent Learning* (IL) durchgeführt. Um die Skalierbarkeit für zukünftige Arbeiten zu testen, wird *Parameter Sharing* (PS) zum Vergleich herangezogen. Bei PS teilen sich alle Agenten die Parameter eines neuronalen Netzes, folgen also der selben Strategie. Abbildung 3.16 zeigt, dass PS zu etwas weniger Kooperation führt, und Abbildung 3.17 zeigt, dass die durchschnittliche Anzahl der gefangenen Beute vergleichbar bleibt. Dies bedeutet, dass PS zu etwas besseren Strategien führt, da entsprechende Jäger in kleineren Gruppen Beute fangen können. Zudem ist PS im Training etwa 2,6 mal schneller als IL: mit PS dauert das Training einer Gruppe von Jägern etwa 19 Stunden, mit IL hingegen etwa 50 Stunden.

3.6.3 Diskussion

In den Experimenten entwickelten die Agenten Formen von *reaktivem*, *proaktivem* und *sozialem* Verhalten [160], das mit verschiedenen Metriken gemessen wurde: Für reaktives und proaktives Verhalten ist das die *Nachhaltigkeit* σ, für soziales Verhalten sind das die *Misserfolgsrate* ϕ, die *Koordinationsrate* η und die *Kooperationsrate* κ.

Das Szenario ist *allgemein*, also weder rein kooperativ noch rein kompetitiv. Die Zielfunktion gibt den Agenten vor, ihr individuelles Wohlergehen zu maximieren und gibt ihnen keine Restriktionen vor, wie sie das erreichen sollen.



(a) Kooperationsrate κ (b) Misserfolgsrate ϕ

Abbildung 3.15: Kooperationsrate κ und Misserfolgsrate ϕ zweier PPO-Jäger mit zehn initialen Beuteagenten und unterschiedlicher initialer Überlebensdauer ohne Fang. Eine höhere Überlebensdauer verringert den Druck auf die Jäger. Die Misserfolge werden durch Jäger verursacht, die nicht lernen, Beute zu fangen, weil sie frühzeitig verhungern. Darstellung basierend auf [124], Layout angepasst, Beschriftungen übersetzt.

Das Dilemma, mit dem die Jäger in diesem Szenario konfrontiert werden, wird durch Konkurrenz und eine Hungermechanik beeinflusst, und eine Tragik der Allmende tritt ein, wenn einer der Jäger dauerhaft gierig handelt. Dies kann beispielsweise aus Kurzsichtigkeit heraus geschehen oder weil der Jäger sonst unmittelbar verhungert. Die einzige rationale Reaktion der anderen Jäger ist, ebenfalls zu jagen, um zu verhindern, dass sie gar keinen Fang machen. In Gegenwart lernender Agenten kann sich das zu einer Negativspirale aufschaukeln. Dies war in den Experimenten erfreulicherweise aber nicht immer der Fall.

Bei der Diskussion möglicher Gründe muss zunächst berücksichtigt werden, dass Defekt und Kooperation im vorliegenden Szenario keine atomaren Aktionen darstellen, sondern Verhaltensweisen sind, die nur über längere Zeitspannen beobachtet werden können. Darüber hinaus wird zwischen Koordination und Kooperation unterschieden, wobei es bei Vergleichen mit der Spieltheorie ausreicht, eine Vermeidung der Tragik der Allmende als Kooperation zu werten. Weiterhin ist zu berücksichtigen, dass die Jäger langfristig (innerhalb ihres Optimierungshorizonts) das gleiche individuelle Ziel haben. Zudem dauern Episoden in Training und Evaluation aus praktischen Gründen höchstens 3.000 Schritte und der aktuelle Zeitschritt ist den Agenten nicht direkt zugänglich. Es ist möglich, dass damit Folgen des Handelns der Agenten, die erst nach 3.000 Schritten relevant sind, nur unzureichend oder gar nicht abbildet wurden.

Theoretisch sind RL-Algorithmen in der Lage, eine Strategie zu finden, die ein Optimierungsproblem innerhalb eines Horizonts bestmöglich löst. Jedoch lernen die Agenten erst im Lauf der Zeit, die langfristigen Folgen ihrer Entscheidungen korrekt zu schätzen. Insbesondere in Multi-Agenten-Szenarien ist

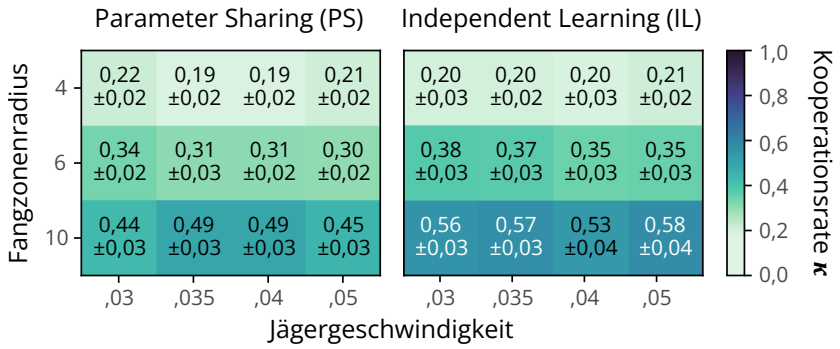


Abbildung 3.16: Kooperationsrate κ dreier PPO Jäger, die mit PPO und unterschiedlichen MARL Architekturen trainiert wurden. Fänge werden nur dann als kooperativ gewertet, wenn alle Jäger beteiligt sind. Darstellung basierend auf [124], Layout angepasst, Beschriftungen übersetzt.

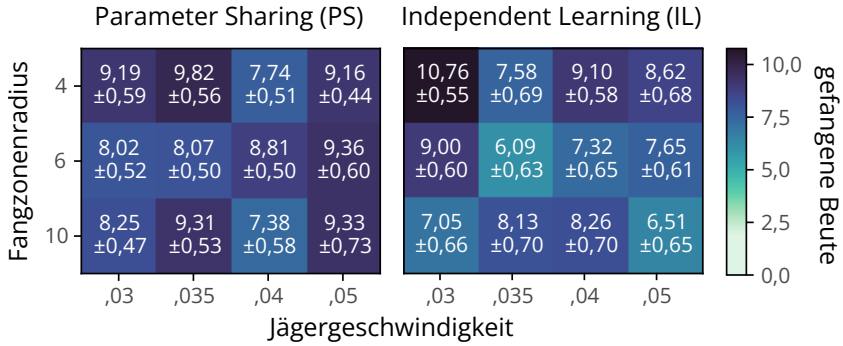


Abbildung 3.17: Kollektiv gefangene Beute dreier RL-Jäger, die mit PPO und unterschiedlichen MARL-Architekturen trainiert wurden. Darstellung basierend auf [124], Layout angepasst, Beschriftungen übersetzt.

es problematisch, dass lernende Agenten häufig zu weniger effizienten Gleichgewichten als dem globalen Optimum konvergieren. Es ist also davon auszugehen, dass die Agenten mindestens zeitweise nicht optimal und im Sinne der Spieltheorie nicht rational handeln. Dies muss aber nicht bedeuten, dass die Agenten dadurch eine Tragik der Allmende verursachen. Dies kann auch bedeuten, dass sie sich dadurch altruistisch (kooperativ) verhalten, obwohl dies in der gegebenen Situation nicht optimal (rational) ist. Eventuell ist die zeitweise Nicht-Optimalität des RL also ein Schlüssel zum Erfolg, um Negativspiralen zu vermeiden.

Ebenso möglich ist, dass aus dem übereinstimmenden langfristigen Ziel folgt, dass jeder Agent aus den Fehlern der anderen Agenten lernen kann. Wenn ein Jäger beobachtet, wie ein anderer Jäger die letzte verbleibende Beute fängt und damit eine Tragik der Allmende herbeiführt, ist das Ergebnis langfris-

tig dasselbe, als wenn er es selbst getan hätte. Die Agenten lernen hier die gleiche Dynamik, die beispielsweise lauten könnte: „Wenn die Population der verbleibenden Beute kleiner als **x** ist, ist die künftig zu erwartende Belohnung 0. Dies ist schlechter als andere Optionen und somit zu vermeiden.“ Tanabe et al. beobachteten [146], dass Lernen das Entstehen von Kooperation beschleunigt und werten dies als bestätigendes Beispiel für den *Baldwin-Effekt*. Für eine solche These fehlt dem vorliegenden Ansatz die evolutionäre Komponente. Die vorliegenden Ergebnisse lassen aber den Schluss zu, dass RL in der Lage ist, Äquivalente zu den Regeln zu lernen, die sich in den sozialen Strukturen von Tieren und Menschen entwickelt haben und die eine Tragik der Allmende abwenden können. Die Ergebnisse deuten jedoch auch darauf hin, dass zur Vermeidung der Tragik der Allmende geeignete Lernbedingungen vorliegen müssen. In den untersuchten Szenarien sind das beispielsweise eine ausreichend lange initiale Überlebensdauer ohne Fang in Kombination mit einer kleinen initialen Beutepopulation.

Im Sinne der Selbstorganisation ist die Vermeidung der Tragik der Allmende zwar koordiniertes Verhalten, aber nur ein Schritt hin zu einer komplexeren Kooperation, in diesem Szenario in Form der Gruppenjagd, die beispielsweise von [72] angeregt wurde. Die Ausgangsparametrisierung des Szenarios ermöglicht es Jägern, Beute erfolgreich alleine zu jagen. Dies ist mutmaßlich leichter zu erlernen, als in einer Gruppe zu jagen, denn Abseits einer gelernten Übereinstimmung, wann die Beute verschont und wann gejagt wird, konnte keine signifikante Gruppenjagd in der Ausgangsparametrisierung der Umgebung beobachtet werden. Gruppenjagd trat erst auf, als ein zusätzlicher Anreiz geschaffen wurde, indem der Beute die Flucht erleichtert wurde, beispielsweise durch Erhöhung ihres Sichtradius und Reduzierung der Geschwindigkeit der Jäger. Dies kann jedoch als Erfolg gewertet werden: Die Agenten verwendeten die mutmaßlich einfachste Lösung, die zum Erfolg führte.

Die Entstehung von Altruismus und Kooperation untersuchen Axelrod et al. [8] an verschiedenen Aspekten des Konzepts der Gegenseitigkeit, technisch umgesetzt mit evolutionären Algorithmen. Die vorliegenden Ergebnisse zeigen, dass Kooperation als emergenter Effekt im MARL ohne Techniken wie Reward Shaping [78], Kommunikation [66] und zentrale Koordination [88] entstehen kann. Die Agenten handelten nicht gierig, sondern im Sinne des globalen Wohlergehens, weil sie langfristig einen Vorteil daraus ziehen konnten. Dies ist klar von bedingungslosem Altruismus abzugrenzen, der mit dem vorliegenden Ansatz nicht erzielt wurde. Die Entstehung von Kooperation in einer Umgebung ohne das unmittelbare Risiko, ausgenutzt zu werden, mag aus spieltheoretischer Sicht trivial sein. Aus praktischer Sicht, insbesondere unter Berücksichtigung von Dafoe et al. [30], ist die beobachtete Kooperation bemerkenswert, weil die Agenten ihre Aktionen weder direkt koordinieren konnten, beispielsweise durch Kommunikation oder die zentrale Wertfunktion des CTDE, noch über zentrale Institutionen verfügten, beispielsweise durch soziale Normen oder Regeln, sondern nur auf der Grundlage vergangener Beobachtungen lernten und han-

delten. Die Skalierung der Experimente auf drei Agenten zeigt aber auch, dass RL auch mit dem aktuellen Stand der Technik eine Gratwanderung zwischen einer zu einfachen Aufgabe, die es den Agenten ermöglicht, individuell erfolgreich zu sein, und einer zu schwierigen Aufgabe, die die (Rechen-)Fähigkeiten der Agenten überfordert, bleibt.

In den durchgeführten Experimenten wurden maximal drei Agenten mit RL trainiert. Es bietet sich an, das Szenario in künftige Untersuchungen mittels PS weiter zu skalieren, da das in dieser Arbeit zu qualitativ vergleichbaren Ergebnissen führte (siehe Abbildung 3.16). Aus biologischer Sicht ist dies mit dem Beobachtungslernen vergleichbar, bei dem ein Individuum ein Verhalten nachahmt, das es bei einem anderen Individuum gesehen hat: Rodríguez et al. [126] berichten, dass im Gehirn eine spezielle Art von Neuron, das sogenannte *Spiegelneuron*, speziell für das Beobachtungslernen ausgelegt ist. Im MARL kann PS als extreme Form des Beobachtungslernens interpretiert werden, wenn Agenten die selbe Zielsetzung und vergleichbare Freiheitsgrade haben. Wenn sich Agenten die Strategie direkt mit den übrigen Agenten teilen, ist dies (nicht nur hinsichtlich Rechenleistung) effizienter, als wenn jeder Agent eine eigene Strategie ausschließlich mit seiner Erfahrung trainiert und die restlichen Informationen aus der Beobachtung anderer Agenten ableitet.

3.7 Zusammenfassung

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf individueller und globaler Ebene lösen können. Dieses Kapitel untersuchte, unter welchen Umständen Agenten mit RL in offenen, unkontrollierten Systemen lernen, sich selbst zu organisieren, also wann und wie sie miteinander interagieren. Besonderes Augenmerk lag auf der Analyse der Umstände, die die Entstehung von Koordination und Kooperation unter lernenden Agenten ermöglichten. Dafür wurde die Problemstellung, die im RL durch die Umgebung repräsentiert wird, gezielt und unter Vermeidung direkter Einflussnahme auf die Agenten variiert: Auf Reward Shaping [78], Kommunikation [66] und zentrale Koordination [88] wurde verzichtet. Durch ein systematisches Vorgehen wurden Bedingungen identifiziert, unter denen bis zu drei RL-Agenten gemeinsam nachhaltig handeln, um eine Tragik der Allmende zu vermeiden (Koordination) sowie aktiv zusammenarbeiten, um gemeinsam ein ad hoc gewähltes Ziel zu erreichen (Kooperation). Gemäß der Definition von Serugendo et al. [35] kann dies als *starke* Form der Selbstorganisation gewertet werden, da es in den untersuchten Szenarien keine explizite zentrale Kontrolle gab, weder intern noch extern. Bekannte AMAS-Ansätze sind nur bedingt in solchen Szenarien einsetzbar, da sie auf a priori definierte Regeln [16] zurückgreifen und dafür (alle) Konfliktsituationen sowie erstrebenswerte Lösungen bekannt sein müssen. Die vorliegenden Ergebnisse zeigen, dass lernende Agenten auch ohne a priori definierte Regeln erfolgreich sein können.

4 Selbstausrichtung

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf individueller und globaler Ebene lösen können. Dieses Kapitel beinhaltet den zweiten von insgesamt drei Abschnitten, die verschiedenen Aspekten dieser These nachgehen. Es befindet sich zentral im RL-Entwicklungszyklus (siehe Abbildung 1.1) und legt den Schwerpunkt auf die Definition der Zielstellung, die im RL durch die Rewardfunktion repräsentiert wird.

Dieses Kapitel beschäftigt sich mit der Frage, wie mehrere Agenten mittels RL lernen, ein zuverlässiges Gesamtsystem zu bilden. Dies wird am Beispiel einer Produktionslinie der Industrie 4.0 betrachtet, in der individuelle Werkstücke gefertigt werden. In diesem kooperativen Szenario sollen mehrere Agenten lernen, ihr Verhalten an einer gegebenen Spezifikation ausrichten. Die Spezifikation umfasst funktionale Aspekte, beispielsweise die Fertigstellung von Produkten in begrenzter Zeit, und nicht-funktionale Aspekte, beispielsweise die Vermeidung von Kollisionen, ohne dass der Aktionsraum der Agenten eingeschränkt wird, beispielsweise durch *Shielding* [5]. Der Fokus liegt auf der Erfüllung der gesamten Spezifikation, die im vorliegenden Fall performantes und gleichzeitig sicheres Verhalten definiert.

Dieses Kapitel ist folgendermaßen strukturiert: Zunächst listet Unterkapitel 4.1 Vorveröffentlichungen auf und legt dar, welche Inhalte in die vorliegende Arbeit übernommen wurden und welche Beiträge die (Ko-)Autoren geleistet haben. Sodann beschreibt Unterkapitel 4.2 das Vorgehen und grenzt dieses in Unterkapitel 4.3 von verwandten Ansätzen ab. In Unterkapitel 4.4 wird der vorgeschlagene Ansatz realisiert und im darauf folgenden Unterkapitel 4.5 empirisch getestet. Dabei werden beispielsweise einzelne Rewardkomponenten, ihr Zusammenwirken und die Skalierung auf eine höhere Anzahl lernender Agenten untersucht und es wird eine Fallstudie durchgeführt, die sicheres Laufzeitverhalten demonstriert. Danach diskutiert Unterkapitel 4.6 die Ergebnisse im Kontext der Zielstellung dieses Kapitels sowie im Kontext verwandter Arbeiten. Abschließend fasst Unterkapitel 4.7 die gewonnen Erkenntnisse zusammen.

4.1 Vorveröffentlichungen

Die zentralen Inhalte dieses Kapitels wurden im Rahmen einer Auftragsforschung für *Siemens Technology* erarbeitet und in zwei Publikationen [121, 122]

veröffentlicht. Diese Auftragsforschung lief bereits mehrere Jahre, als der Autor der vorliegenden Arbeit hinzukam. Dementsprechend ist die Umgebung, die in diesem Kapitel verwendet wird, optisch und konzeptionell verwandt mit den Umgebungen in [42, 109], jedoch war der Autor der vorliegenden Arbeit dort nicht federführend, sondern nur unterstützend tätig. Die Reihenfolge der Autoren spiegelt dies jeweils angemessen wieder und [42, 109] sind kein Beitrag (zu) dieser Dissertation.

Die Publikation mit dem Titel *SAT-MARL: Specification Aware Training in Multi-Agent Reinforcement Learning* [121] überträgt funktionale und nichtfunktionale Anforderungen über Reward Shaping explizit in das Feedback der Agenten. Die Experimente werden in der *Smart Factory* durchgeführt, einer simulierten Multi-Agenten Umgebung, die eine industrielle Lot-Size One Produktion mit bis zu acht Agenten modelliert. Die Ergebnisse zeigen, dass dieses Vorgehen Agenten ermöglicht, die Einhaltung funktionaler und nichtfunktionaler Anforderungen zu lernen. Der Autor der vorliegenden Arbeit entwickelte das Konzept, implementierte den RL-Algorithmus DQN und die Trainingspipeline, führte die Experimente durch und verfasste den Großteil der Publikation. Robert Müller unterstützte bei der Implementierung von DQN mit seinem Wissen über PyTorch. Die Implementierung der Umgebung erfolgte gemeinsam mit Thomy Phan und Andreas Sedlmeier im Rahmen der Auftragsforschung für Siemens Technology. Andreas Sedlmeier realisierte die Visualisierung der Simulation mittels Unity3D (siehe Abbildung 4.1). Thomy Phan trug das Kapitel *Reinforcement Learning* und die Implementierung der Faktorisierung (siehe Kapitel 2.2.3) bei, mit denen die Algorithmen VDN und QMIX auf Basis von DQN realisiert wurden. Thomas Gabor unterstützte beim Verfassen der Kapitel *Introduction* und *Conclusion*. Marc Zeller, Jan Wieghardt, Reiner Schmid, Horst Sauer, Cornel Klein gaben den Fokus der Auftragsforschung vor und halfen in regelmäßigen Diskussionen, die Experimente hinsichtlich der Ausgestaltung der nicht-funktionalen Anforderungen zu schärfen. Claudia Linnhoff-Popien stand für inhaltliche und strukturgebende Diskussionen der vorgestellten Konzepte zur Verfügung. Die Inhalte dieser Publikation fließen in die nachfolgenden Kapitel 4.2 bis 4.6 ein.

Die Publikation mit dem Titel *Specification Aware Multi-Agent Reinforcement Learning* [122] ergänzt die zuvor beschriebene Publikation [121] um eine Fallstudie, die demonstriert, wie das Training von RL-Agenten mit nichtfunktionalen Anforderungen, die in Rewards abgebildet werden, das nachfolgende Laufzeitverhalten dieser Agenten beeinflusst. Die Publikation visualisiert das erzielte, proaktiv sichere Verhalten und diskutiert Alternativen. Der Autor der vorliegenden Arbeit überarbeitete das bestehende Konzept, modifizierte die bestehende Trainingspipeline, führte die Mehrheit der zusätzlichen Experimente durch und überarbeitete die bestehende Publikation. Thomy Phan trug Experimente mit dem Algorithmus VDN bei und half gemeinsam mit Andreas Sedlmeier in regelmäßigen Diskussionen, die visuelle und textuelle Präsentation der Ergebnisse zu verbessern. Abseits davon ist die Pu-

blikationen inhaltlich deckungsgleich mit [121], weshalb auf eine wiederholte Auflistung der Beiträge der übrigen Ko-Autoren verzichtet wird. Die Inhalte dieser Publikation fließen in die nachfolgenden Kapitel 4.2 bis 4.6 ein.

4.2 Problemmodellierung und Vorgehen

Im vorherigen Kapitel 3 wurde gezeigt, dass RL-Agenten mit individuellen Zielen in der Lage sind, Koordination und Kooperation zu erlernen, wenn sie davon langfristig einen Vorteil haben. Da ihnen bewusst Freiheit hinsichtlich des erlernten Verhaltens gelassen wurde, gab es auch Szenarien, in denen eine Tragik der Allmende eintrat. Daraus erwächst eine Herausforderung, der sich dieses Kapitel widmet: In der Praxis müssen sich Agenten an in bestimmten Bereiche geltende Regeln halten und dürfen kein unerwünschtes oder sogar gefährliches Verhalten lernen [44, 51, 128]. Dieses Problem wird in der Literatur als *Ausrichtung* (englisch: Alignment) bezeichnet [43, 59, 79].

Auch in diesem Kapitel werden autonome Agenten mittels RL realisiert. RL-Systeme benötigen grundsätzlich ein klar definiertes Ziel, beispielsweise „Gewinne die Schachpartie“ [137]. Im Schach gibt es Spielregeln, die eindeutig definiert und zu jedem Zeitpunkt überprüfbar sind sowie nicht im Konflikt miteinander stehen. In praktischen Problemen ist die Formulierung eines einzelnen Ziels jedoch nicht ausreichend, sondern es gibt eine Vielzahl von Aspekten [36], die gegeneinander abgewogen werden müssen und miteinander im Konflikt stehen können. In diesem Kapitel wird dies am Beispiel einer adaptiven Lot-Size One Produktion betrachtet, die Teil der Fabrik der nahen Zukunft sein könnte [152].

In einer solchen *Smart Factory* werden individuelle Produkte in kleinen Mengen hergestellt. Je Produkt ist dazu eine Reihe von Produktionsschritten erforderlich, die von unterschiedlichen Maschinen durchgeführt werden. Das Ziel ist, eine bestimmte Anzahl unterschiedlicher Produkte innerhalb eines vorgegebenen Zeitraums zu produzieren. Im *Software Engineering* (SE) ist dies eine *funktionale Anforderung*. In der Regel gibt es auch *nicht-funktionale Anforderungen*: Beispielsweise soll das System eine schnellere Produktion der Ausschöpfung der verfügbaren Zeit vorziehen; es soll die Beschädigung von Artikeln und Maschinen vermeiden; es soll widerstandsfähig gegenüber (menschlichen) Eingriffen und unerwarteten Ereignissen sein [9, 24]. Die Gesamtheit dieser funktionalen und nicht-funktionalen Anforderungen wird als *Spezifikation* eines Systems bezeichnet.

Die Erfüllung einer solchen Spezifikation könnte als direktes Ziel der Agenten festgelegt werden. Basierend darauf könnte ein binären Reward definiert werden, beispielsweise +1, wenn die Spezifikation erfüllt ist, und 0 ansonsten. Mit solch einem Reward ist es in der Praxis aber schwer, Agenten zu trainieren, da diese zu Beginn des Trainings nicht optimal handeln. In dieser Phase ist es

unwahrscheinlich, dass sie alle Anforderungen erfüllen, insbesondere, wenn die Anforderungen eine Hierarchie bilden, indem sie aufeinander aufbauen. Das ist beispielsweise der Fall, wenn eine Anforderung die Zusammenarbeit mehrerer Agenten erfordert. Da die dafür erforderliche Kooperation typischerweise erst im Laufe des Trainings erlernt wird, werden solche Anforderungen während der Exploration selten erfüllt. Der binäre Reward gibt den Agenten keinerlei Feedback darüber, wie nah sie der Lösung waren, hier also der Erfüllung der gesamten Spezifikation. Alternativ könnten einzelne Anforderungen einer Spezifikation als boolesche Funktionen dargestellt werden, die jeweils eine binäre Belohnung ergeben, wenn sie erfüllt werden. Jedoch kann eine solche Repräsentation eine unterschiedliche Gewichtung [83] verschiedener Anforderungen nicht abbilden. Dass Fehlen von zielgerichtetem Feedback kann den Lernprozess der Agenten verlangsamen und im schlimmsten Fall blockieren. Daher verwendet die vorliegende Arbeit kontinuierliche Rewards, die je eine Anforderung der Spezifikation gewichtet abbilden können.

Ein weiterer Aspekt ist die Interaktion lernender Agenten miteinander und ihrer Umgebung im Training. Wenn beispielsweise mehrere Agenten um dieselbe Position oder Maschine konkurrieren, kann das Verhindern einzelner Aktionen durch *Shielding* [5] oder ein *Action Elimination Network* [163] zwar unmittelbar unerwünschte Folgen wie Kollisionen verhindern, aber solche Ansätze erlauben den Agenten nicht, zu lernen, wie sie den zu Grunde liegenden Konflikt künftig vermeiden. Und auch lokale Konflikte um gemeinsame Ressourcen wie Platz oder Maschinen können globale Folgen haben, beispielsweise ein Leerlauf der nachfolgenden Produktionslinie. Um solche Konflikte mit a priori definierten Regeln zu lösen, müsste berücksichtigt werden, dass Agenten ihr Verhalten durch den Lernprozess fortlaufend verändern, sich also im Lauf der Zeit unterschiedliche Konflikte ergeben können. Ebenso müssten die langfristigen Folgen und Seiteneffekte von unmittelbaren Aktionen vorhergesehen werden, was sich in der Praxis als äußerst schwierig erweist [74]. Und letztlich würde damit ein (zunehmend größerer) Teil des Problems manuell gelöst werden, für das ursprünglich die autonomen Agenten vorgesehen waren. Daher schränkt die vorliegende Arbeit den Aktionsraum von RL-Agenten nicht situativ ein und sie verwendet keine Heuristiken zur Lösung lokaler Konflikte.

Die Ergebnisse des vorherigen Kapitels 3 zeigen, dass lernende Agenten selbstständig Koordination und Kooperation lernen können, selbst wenn sie mit einem sozialen Dilemma [78] konfrontiert sind, dass gieriges oder ängstliches Handeln verursachen kann. Anders als das Szenario des vorherigen Kapitels modelliert die Smart Factory eine kontrollierte Umgebung, in der per Design sichergestellt ist, dass den Agenten insgesamt ausreichend (gemeinsame) Ressourcen zur Verfügung stehen, beispielsweise Maschinen zum Durchführen der Produktionsschritte. Zwar können lokale Konflikte entstehen, aber kein Agent hat einen Vorteil davon, eine Ressource länger oder häufiger als erforderlich zu verwenden, womit die Handlungsmotive Angst und Gier entfallen. Daher kann hier kein soziales Dilemma auftreten, was wiederum die Entstehung emergenter

Kooperation begünstigt [8]. Ebenso anders als im vorherigen Kapitel liegt hier eine präzisere Anforderung an das Verhalten der Agenten in Form einer Spezifikation vor. Der Ansatz für diesen Abschnitt der Arbeit ist, alle Anforderungen der Spezifikation in das Feedback der Agenten einfließen zu lassen, dass im RL durch den Reward repräsentiert und durch die Rewardfunktion definiert wird. Dieses Vorgehen folgt dem Vorschlag von Leike et al. [79] und führt insbesondere keine Trennung durch, die nicht-funktionale Anforderungen in gesonderten Verhaltensregeln oder zusätzlichen Sicherungsschichten realisieren würde. Es werden eine Vielzahl kontinuierlicher anstelle eines einzelnen, binären Rewards verwendet und getestet, ob das den Lernprozess der Agenten so unterstützt, sodass sie letztlich alle Anforderungen einer Spezifikation vollständig erfüllen können Darüber hinaus wird untersucht, welches Verhalten Agenten lernen, wenn sie mit teilweise in Konflikt stehenden Anforderungen konfrontiert werden. In der Literatur [36] wurden solche Szenarien als *Multi-Objective* Probleme aufgefasst, wobei Agenten deutlich an Leistung verloren, wenn die Rewards auch die Erfüllung nicht-funktionaler Anforderungen berücksichtigten. Der Ansatz dieser Arbeit ist, die Gewichtung einzelner Rewardkomponenten während des Trainings zu variieren. Dabei wird getestet, ob Agenten so die Auswirkungen ihres Verhaltens auf die Erfüllung unterschiedlicher Anforderungen und basierend darauf eine entsprechend balancierte Strategie lernen.

4.3 Verwandte Arbeiten

In diesem Unterkapitel wird der Ansatz der vorliegenden Arbeit in insgesamt drei Richtungen abgegrenzt: *Safe RL* (Unterkapitel 4.3.1), *Reward Shaping* (Unterkapitel 4.3.2) und *Zielgerichtetes RL* (Unterkapitel 4.3.3).

4.3.1 Safe RL

Die erste Richtung der Abgrenzung sind Ansätze des *Safe* RL. Safe RL ist ein Teilbereich der *AI Safety* (siehe Kapitel 2.4), hat jedoch durch das Lernen aus Interaktion mit der Umgebung spezielle Schwerpunkte wie die sichere Exploration, die über *AI Safety Gridworlds* [80] getestet werden kann. Sicherheitsanforderungen werden im Verlauf des Kapitels in Form nicht-funktionaler Anforderungen innerhalb der Spezifikation berücksichtigt. Der vorliegende Ansatz ist jedoch nicht als Safe-RL-Ansatz konzipiert und hat weder sicheres Verhalten während des Trainings (die Agenten sollen die Folgen unsicherer Aktionen lernen) noch Garantien zur Laufzeit, wie beweisbar sicheres Verhalten zum Ziel. Der Ansatz erlaubt, dass Agenten lernen, funktionale und nicht-funktionale Anforderungen so (weit) zu erfüllen, wie es sich (mathematisch) im Training als optimal erweist.

Einen älteren, hinsichtlich der Kategorisierung zeitlosen Überblick über den Bereich des Safe RL geben Gervais et al. [44]. Darin unterscheiden sie zunächst zwischen Ansätzen, die bereits während der Exploration sicher sind und Ansätzen, die Sicherheit erst während der Optimierung des Verhaltens berücksichtigen. Die feinere Unterteilung nehmen sie anhand der Modellierung von Sicherheit und Risiko vor. Eine explizite Modellierung von Sicherheit oder Risiko kann beispielsweise verwendet werden, um einen MDP so einzuschränken, dass unsichere oder zu riskante Aktionen blockiert werden. Die vorliegende Arbeit modelliert weder Risiko noch Sicherheit explizit im Sinne des Safe RL. Im Sinne der Kategorisierung von Gervais et al. gehört das gewählte Vorgehen zu den Ansätzen, die während der Exploration unsicher agieren und Sicherheit als Kriterium während der Optimierung berücksichtigen: Die Agenten sollen lernen, ihr Verhalten an eine gegebene Spezifikation anzupassen, wobei dies an der Erfüllung von funktionalen und nicht-funktionalen Anforderungen gemessen wird.

Einen jüngeren Überblick über den Bereich des Safe RL geben Gu et al. in [51], in dem sie gegen Ende auf MARL-spezifische Herausforderungen eingehen, mit denen die vorliegende Arbeit ebenfalls konfrontiert ist. Relevant ist die Interaktion der Agenten, die gleichermaßen Ursache und Lösung für Sicherheitsprobleme auf unterschiedlichen Ebenen sein kann. Beispielsweise spielen sich Kollisionen lokal ab und können auf einzelne Agenten zurückgeführt werden. Das könnte zwar über Methoden des Safe RL, die für einzelne Agenten entwickelt wurden, verhindert werden, aber die globalen Folgen solcher lokalen Eingriffe sind schwer vorherzusehen, insbesondere hinsichtlich der Balance von Leistung und Sicherheit, die Arnold et al. [36] als *Multi-Objective* Problem auffassen. Ebenso kann Sicherheit über die lokale Ebene hinausgehen und die aktive Zusammenarbeit mehrerer Agenten erfordern, beispielsweise ein koordiniertes Abschirmen eines Verunglückten von der Umgebung. Der Vorteil des vorliegenden Ansatzes in solchen kooperativen Szenarien ist, dass das Credit-Assignment, dass es ohnehin zu lösen gilt, nicht nur den Beitrag einzelner Agenten zur Leistung, sondern auch zur Sicherheit des Gesamtsystems schätzen kann. Das ist aus Sicht des Safe RL keine ideale Lösung, denn dem vorliegenden Ansatz fehlen Garantien, jedoch kann er als Schritt in diese Richtung betrachtet werden.

Es gibt Ansätze des Safe RL, die Sicherheit erlernen, was mit dem vorliegenden Ansatz grundsätzlichen vergleichbar ist. Beispielsweise erweitern Seurin et al. [134] den MDP um eine Funktion, die Zustände und Aktionen auf ein binäres Signal abbildet, das die Gültigkeit der Aktion (erlaubt oder verboten) anzeigt. Dann trainieren sie ein zusätzliches neuronales Netz, um diese Gültigkeit vorherzusagen. Diese Vorhersagen berücksichtigen sie dann beim Training eines RL-Agenten mit DQN, dessen Trainingsziel so modifiziert wurde, dass die Q-Werte verbotener Aktionen niedriger als die Q-Werte erlaubter Aktionen sind. Konzeptuell ähnlich verwenden Zahavy et al. [163] ein *Action Elimination Network*, das mittels eines zusätzlichen Signals aus der Umgebung dazu trai-

niert wird, verbotene Aktionen vorherzusagen. Auch Zahavy et al. realisieren ihren RL-Agenten mit DQN, wobei das trainierte Action Elimination Network verbotene Aktionen unabhängig vom Q-Wert entfernt, bevor der Agent aus den verbleibenden Aktionen gemäß der Q-Werte wählt. Beide Ansätze können nur lokal auf einzelne Agenten angewandt werden und können Sicherheitsaspekte, die über das Vermeiden einzelner Aktionen hinaus gehen, nicht abbilden.

Hinsichtlich eines sicheren Lernprozesses in MAS, einschließlich der Exploration, verwenden Berducci et al. [13] adaptive *Control Barrier Functions*. Einerseits können diese verhindern, dass definierte Grenzwerte überschritten werden und andererseits erlauben sie eine Anpassung einzelner Koeffizienten, um beispielsweise eine Gleichgewicht zwischen Leistung und Sicherheit zu finden. Durch eine Anpassung dieser Koeffizienten können Berducci et al. eine Beeinflussung emergenten Verhaltens demonstrieren. Der Nachteil an diesem Vorgehen ist, dass das Framework immer nur einen Agenten trainiert und die anderen Agenten als Teil des (dynamischen) Problems betrachtet und skaliert daher schlecht mit steigender Anzahl von Agenten. Mangels Control Barrier Functions bietet der vorliegende Ansatz keine Garantien, aber er skaliert besser, da alle Agenten gleichzeitig lernen.

4.3.2 Reward Shaping

Die zweite Richtung der Abgrenzung ist der Einsatz von *Reward Shaping* [145, S. 470]. Ziele werden im RL durch die Rewardfunktion repräsentiert abgebildet und das (mathematische) Ziel von RL ist, die Summe der erwarteten Rewards zu maximieren. Daher ist es naheliegend, den Lernprozess von RL-Agenten über die Rewardfunktion zu beeinflussen. Allgemein ist das in Problemen relevant, in denen selten Feedback gewährt wird, beispielsweise weil das Ziel initial nur schwer zu erreichen ist.

In kooperativen MAS kann Reward Shaping auch genutzt werden, um das Credit-Assignment-Problem (Schätzung des Beitrags einzelner Agenten zum globalen Reward) zu lösen. Beispielsweise leiten Salimbeni et al. [129] individuelle Rewards basierend auf einer Kalman-Filterung ab und Wolpert et al. [159] verwenden dafür die Differenz zwischen dem tatsächlichen und einem alternativem Reward, der bei der Wahl der durchschnittlichen Aktion aller Agenten gewährt werden würde. Der alternative Reward drückt aus, ob die vom Agenten gewählte Aktionben für das Gesamtsystem vorteilhaft oder nachteilig ist. Foerster et al. [40] entwickeln diesen Ansatz innerhalb einer CTDE-Architektur weiter. Sie berechnen den alternativen Reward, indem sie die Aktion des jeweils betrachteten Agenten ausklammern und die Handlungen der anderen Agenten unverändert lassen. Dadurch lässt sich der Beitrag einzelner Agenten präziser messen. Die zuvor genannten Ansätze führen tiefgreifende Änderungen an den Rewards durch, um Kooperation zu fördern. Kooperation ist im vorliegenden Szenario relevant, aber das unmittelbare Ziel ist die Erfüllung einer Spezifika-

tion, wobei die Agenten die dafür optimale Form der Interaktion selbst lernen sollen. Das vorherige Kapitel 3 hat gezeigt, das dies möglich ist, und mit dem Verzicht auf solche Techniken werden mögliche Seiteneffekte minimiert.

Ein Ansatz, der ausgehend von einem RL-System mit einem Agenten weder die optimale Strategie verändert noch Seiteneffekte verursacht, die ein *Reward Hacking* [7, 79, 80] ermöglichen würden, ist das von Ng et al. entwickelte Potential Based Reward Shaping (PBRS) [99]. PBRS ergänzt den Reward $\mathcal{R}$ pro Zeitschritt um eine zusätzliche Komponente F, die als Differenz des Potenzials Φ eines Zustands und dessen Folgezustands definiert ist:

$$F(s_t, s_{t+1}) = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t) \tag{4.1}$$

$$r_t = \mathcal{R}(s_t, a_t) + F(s_t, s_{t+1}) \tag{4.2}$$

Das Potenzial einzelner Zustände wird meist in einer separaten Funktion definiert und enthält Expertenwissen über das zu lösende Problem, beispielsweise in Form näherungsweise optimaler Heuristiken. Devlin et al. [32] zeigten später, dass die ohne und mit PBRS erlernten Strategien eines Problems nicht nur bei statischen, sondern auch bei dynamischen Potenzialfunktionen invariant sind und Nash-Equilibria in MAS durch den Einsatz von PBRS nicht verändert werden. Im Stil alternativer Rewards erfassen sie den Beitrag einzelner Agenten zur Leistung des MAS und wenden diese im Stil des PBRS an, wobei sie eine Steigerung des Lernverhaltens der Agenten beobachteten. Für episodische Szenarien wie das vorliegende ist zudem die Studie von Grześ [50] relevant, die zu dem Ergebnis kommt, dass eine Anwendung von PBRS insbesondere in MAS unter Beibehaltung der theoretischen Garantien möglich ist, sofern terminierende Zustände (Ende einer Episode) neutrales Potenzial haben.

Der vorliegende Ansatz orientiert sich an der grundlegenden Funktionsweise von PBRS, um Agenten je Zeitschritt Feedback in Form eines Reward zu geben, auch wenn die globale Spezifikation nur partiell erfüllt ist. Da aussagekräftige, individuelle Rewards zur Verfügung stehen, wird das Credit-Assignment nicht als unmittelbares Problem betrachtet und strikt getrennt: Die Anforderungen der Spezifikation werden den Agenten über Reward Shaping zugänglich gemacht, dass Training erfolgt mit den bewährten Paradigmen IL und CTDE. Insbesondere wird Kooperation nicht über zusätzliches Reward Shaping forciert. Wenn die Spezifikation eine besondere Form der Interaktion wie Kooperation fordert, wird dies ohnehin in den Rewards abgebildet. Ist Kooperation aus Sicht der Spezifikation optional, sollten Agenten selbst lernen, wie sie die Spezifikation optimal erfüllen.

4.3.3 Zielgerichtetes RL

Die dritte Richtung der Abgrenzung sind Ansätze des RL, die das Training über Rewards in Richtung eines definierten Verhaltens lenken.

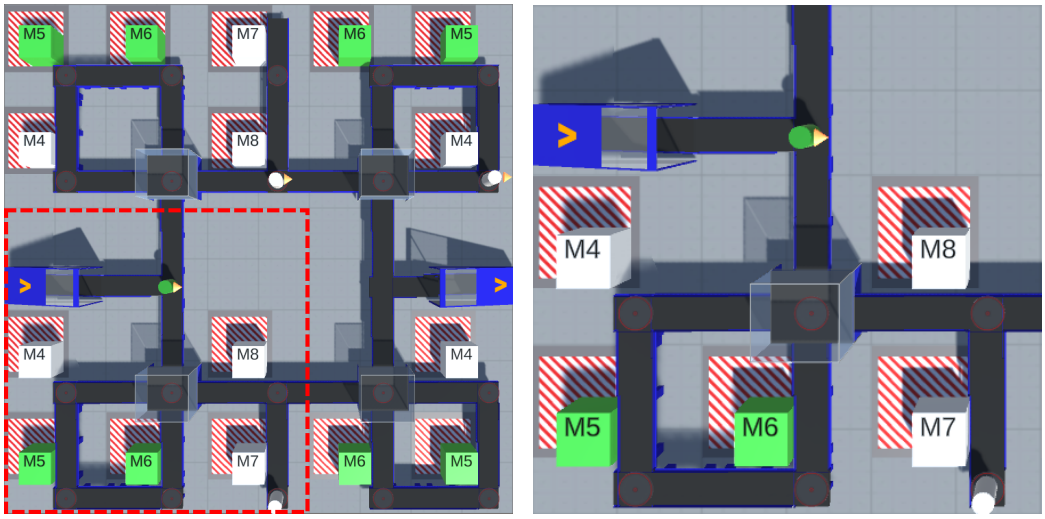
Das populationsbasierte Training von Jaderberg et al. [68] ergänzt den globalen Reward, der erst bei Beendigung der Aufgabe gewährt wird, um individuelle Rewards pro Agent und Zeitschritt. Diese individuellen Rewards sind an bestimmte Ereignisse gekoppelt, die jeder Agent im Training anders gewichtet. Zwischen zwei Generationen von Agenten wird die Gewichtung durch einen evolutionären Prozess verändert, der in der Selektion diejenigen Agenten begünstigt, deren individuelle Rewards im Training zu einem Verhalten führt, das den globalen Reward maximiert. Liu et al. [85] ergänzen den Ansatz von Jaderberg et al. [68] um separate Diskontierungsfaktoren je Rewardkomponente, was in Anbetracht der Ergebnisse in Kapitel 3.5 als wichtige Ergänzung erscheint. Diese populationsbasierten Ansätze haben in allgemeinen Szenarien (ein Team aus heterogenen Agenten trat in einem symmetrischen Szenario gegen ein anderes Team an) ein erfolgreiches Training ermöglicht und emergente Kooperation erzielt. Offen ist, ob diese Vorgehensweise auch in kooperativen Szenarien wie dem vorliegenden funktioniert, in denen Agenten nicht im Wettbewerb miteinander stehen, wodurch ein geringeres Maß an Diversität und eine statische Aufgabenschwierigkeit gegeben sind. Ebenso ist in industriellen Szenarien üblich, homogene Systeme (hier: Agenten) zu verwenden, um eine mehrfache Zertifizierung zu vermeiden und ein höheres Maß an Vorhersehbarkeit hinsichtlich des Verhaltens sicherzustellen. Abschließend ist zu bedenken, dass eine Population von Agenten am Ende des evolutionären Prozesses üblicherweise aus mehreren hundert Individuen besteht, deren Training insgesamt entsprechend rechenintensiv ist.

Einen Ansatz mit anderem Schwerpunkt stellen Berducci et al. in [12] vor. Ihre Ausgangslage sind Aufgaben, die als partiell geordnete Menge von Ziel-, Sicherheits- und Komfortanforderungen in einer formalen Sprache vorliegen. Eine Komfortanforderung ist beispielsweise eine gleichmäßige Kurvenfahrt eines autonomen Fahrzeugs, und die partielle Ordnung definiert beispielsweise, dass Sicherheitsanforderungen Vorrang vor Komfortanforderungen haben. Um diese Aufgaben zu erfüllen, trainieren Berducci et al. einzelne RL-Agenten mit multivariaten Rewards, die sie über einen hierarchischen, potenzialbasierten Ansatz (siehe vorheriges Kapitel 4.3.2) berechnen. Die formalisierten Anforderungen transferieren sie in eine Hierarchie, die im Reward Shaping berücksichtigt wird. Verglichen mit nicht spezialisierten Ansätzen erzielen ihre Agenten einen besseren Komfort und konvergieren schneller zur optimalen Strategie. Der Ansatz von Berducci et al. ist jünger als der vorliegende Ansatz. Der automatisierte Transfer einer Spezifikation in Rewards ist konzeptuell elegant, erfordert jedoch eine Repräsentation aller Anforderungen in einer formalen Sprache, was in Multi-Agenten-Szenarien komplex sein kann. Ungeachtet dessen wäre es spannend, diesen Ansatz für MAS weiterzuentwickeln und zu testen, ob er selbstständig Szenarien lösen kann, für die in der vorliegenden Arbeit eine dynamische Anpassung der Rewards während des Trainings von Hand erforderlich ist (siehe nachfolgendes Kapitel 4.5.3).

4.4 Fallstudie: Lot-Size One Produktion

Im Folgenden wird der Ansatz realisiert. Zuerst beschreibt Unterkapitel 4.4.1, wie die Smart Factory eine Lot-Size One Produktion als Multi-Agenten-System modelliert. Danach erklärt Unterkapitel 4.4.2, wie typische funktionale und nicht-funktionale Anforderungen einer solchen Produktion in Rewards abgebildet werden. Abschließend legt Unterkapitel 4.4.3 das experimentelle Setup für die nachfolgende Evaluation dar.

4.4.1 Simulation



(a) Gesamtansicht (b) Vergrößerung Bereich links unten

Abbildung 4.1: 3D Visualisierung der *Smart Factory* Simulation. Die Agenten (Zylinder) können sich nur über definierte Wege (schwarz) zu benachbarten Positionen bewegen. Mit Ausnahme der 4 Ausweichbereiche (transparente Würfel) kann sich nur ein Agent auf einer Position aufhalten. Die Agenten starten im Eingang (blauer Quader) auf der linken Seite und müssen ihre Produkte erst zu definierten Maschinen (grüne Würfel) und danach zum Ausgang (blauer Quader) auf der rechten Seite transportieren. Darstellungen übernommen aus [122].

Die verwendete Smart Factory Simulation hat hinsichtlich der Modellierung von Bestandteilen und Abläufen einer adaptiven Produktionsanlage ihren Ursprung in [107]. Diese Arbeit verwendet das in Abbildung 4.1 dargestellte Layout, das aus einem 5×5 Grid von *Maschinen* verschiedener Typen besteht. Jeder Agent i transportiert ein *Produkt*, das an den Maschinen gemäß einer Liste von Produktionsschritten $tasks_i = [\{a_{i,1}, b_{i,1}\}, \{a_{i,2}, b_{i,2}\}, ...]$ bearbeitet werden muss. Als Inspiration dient die Produktion individueller Werk-

stücke (*Lot-Size One*), daher wird die Liste von Produktionsschritten je Agent und Durchlauf zufällig erstellt. Jeder Produktionsschritt $a_{i,j}$,$b_{i,j}$ ist Teil einer *Gruppe*. Produktionsschritte innerhalb einer Gruppe dürfen in beliebiger Reihenfolge abgearbeitet werden, aber die Reihenfolge der Gruppen muss eingehalten werden. Damit werden mögliche Abhängigkeiten und Freiheitsgrade zwischen Produktionsschritten modelliert. Beispielsweise mag es egal sein, ob zuerst Motorhaube oder Heckklappe montiert werden, jedoch muss beides vor der Lackierung des Autos geschehen. Abbildung 4.1b zeigt beispielhaft einen Agenten i, dargestellt als grüner Zylinder, mit $tasks_i = [\{5, 6\}]$, dargestellt als grüne Kästen. Hier kann i zwischen verschiedenen Maschinen wählen, um die Produktionsschritte $a_{i,1} = 5$ und $b_{i,2} = 6$ durchzuführen.

Anders als im vorherigen Kapitel 3 wird die Interaktion der Agenten mit der Umgebung in diesem Kapitel auf abstrakterer Ebene modelliert. Der Fokus liegt hier nicht auf dem kontinuierlichen Ansteuern einzelner Aktuatore unter Berücksichtigung der Physik, sondern darauf, dass die Agenten ein dynamisches Routing-Problem dezentral lösen. Alle Agenten beginnen an einem Eingang auf der linken Seite der Smart Factory und verfügen über einen diskreten Aktionsraum: Pro Zeitschritt können sie entweder an Ort und Stelle bleiben (*no-op*), sich zu benachbarten Positionen begeben (*links, rechts, oben, unten*) oder ihr Produkt von der Maschine an ihrer aktuellen Position bearbeiten lassen (*enqueue*), was auch als Verwendung der Maschine beziehungsweise Produktionsschritt bezeichnet wird. Pro Zeitschritt kann eine Maschine nur einen Produktionsschritt gemäß ihrem Typ durchführen. Wenn ein passendes Produkt vorliegt, wird der entsprechende Produktionsschritt durchgeführt und aus der Liste des Produkts entfernt. Wenn eine Gruppe leer ist, wird sie ebenfalls aus der Liste des Produkts entfernt. Ein Produkt ist abgeschlossen, wenn seine Liste leer ist. Die letzte Aufgabe besteht immer darin, zum Ausgang zu navigieren, der die Spiegelposition des Eingangs ist (rechts).

Die Navigation innerhalb der Smart Factory unterliegt Einschränkungen. Ausgehend von einer Position führen definierte Verbindungen (Wege) zu den umliegenden Positionen. Jede Position kann nur eine begrenzte Anzahl Agenten aufnehmen. Ein Agent kann sich nur dann zu einer anderen Position bewegen, wenn ein Weg existiert und das Kapazitätslimit der Zielposition dadurch nicht überschritten wird. Die Ein- und Ausgänge können alle Agenten gleichzeitig beherbergen und es gibt vier *Ausweichbereiche*, die vollständig mit den Nachbarpositionen verbunden sind und die jeweils die Hälfte aller Agenten aufnehmen können. Ein solcher Ausweichbereich ist in Abbildung 4.1b südlich von Agent i in Form eines transparenten Kastens zu sehen. Alle übrigen Positionen können nur einen Agenten aufnehmen. Da sich mehrere Agenten in der Smart Factory befinden, ist Koordination erforderlich, um Konflikte bei der Wahl geeigneter Wege und Maschinen zu vermeiden: Agenten können ihre Positionen nur über die Ausweichbereiche tauschen und müssen ansonsten hintereinander her fahren.

Im Zuge der parallelen Forschung an Robustheit in MARL [108, 109] wurden verschiedene Varianten der Smart Factory Simulation veröffentlicht. Die vorliegende Variante unterscheidet sich von den anderen durch die explizite Modellierung zweier Arten von nicht-funktionalen Anforderungen. Die erste Art kann durch das Vermeiden bestimmter Aktionen erfüllt werden und betrifft das Verwenden von falschen Maschinen, das Abweichen von definierten Verbindungen sowie Kollisionen mit anderen Agenten. Wie bereits diskutiert können zusätzliche Sicherheitsmechanismen solche Aktionen verhindern, aber sie lösen das Routing-Problem nicht. Diese Anforderungen werden in allen nachfolgenden Experimenten berücksichtigt. Die zweite Art erfordert zeitweise ein Verhalten, das mit den funktionalen Anforderungen in Konflikt steht, und betrifft Notfallsituationen, in denen die Agenten entweder anhalten oder sich innerhalb begrenzter Zeit zu definierten Positionen begeben sollen. Die Motivation dafür ist, dass (teil-)autonome Systeme typischerweise über Fallback-Routinen für den Fall eines internen oder externen Versagens verfügen. Diese Arbeit geht von Letzterem aus, wobei das Anhalten dem häufig anzutreffenden Notaus nachempfunden ist. Dieses Szenario wird in Unterkapitel 4.5.3 betrachtet. Ein Notaus ist aber nicht immer ausreichend, beispielsweise bei einem Feueralarm. Die Agenten könnten FTFs steuern, die entweder wertvolle oder gefährliche, beispielsweise brennbare, Produkte transportieren. Selbst wenn FTFs und Produkte abkömmlich und ersetzbar sind, kann es dennoch wichtig sein, dass sie weder die Evakuierung von Menschen, noch die Feuerlöscharbeiten behindern. Diese Arbeit nimmt an, dass es besser ist, sich in begrenzter Zeit an definierte Positionen zu begeben, als einfach anzuhalten. Dieses Szenario wird in Unterkapitel 4.5.4 betrachtet.

4.4.2 Abbildung der Spezifikation

Der Ansatz dieser Arbeit ist, funktionale und nicht-funktionale Anforderungen einer Spezifikation als Rewards abzubilden, damit die Agenten lernen, ihr Verhalten an dieser Spezifikation auszurichten. Dafür wird im Stil von PBRS zunächst eine Potenzialfunktion Φ (siehe Gleichung 4.1) definiert. Jedoch wird ausschließlich die Differenz des Potenzials zweier aufeinanderfolgender Zeitschritte als Reward verwendet, also $r_t = F(s_t, s_{t+1})$. In ersten Tests wurde beobachtet, dass $\gamma < 1$ bei großen Beträgen des Potentials mathematische Seiteneffekte verursacht, die das Lernverhalten negativ beeinflussen. Beispielsweise müsste ein Agent mit $\Phi(s_t) = 10$ und $\gamma = 0,95$ sein künftiges Potenzial je Zeitschritt um mindestens $0,05$ erhöhen, um keinen negativen Reward zu erhalten. Teilweise führte das dazu, dass Agenten gezielt hohe Potentiale mieden und die Problemstellung nicht vollständig lösten Daher wird bei der Berechnung von F grundsätzlich $\gamma = 1,0$ verwendet, wohingegen die RL Algorithmen überwiegend mit $\gamma = 0,95$ parametrisiert sind. Folglich ist davon auszugehen, dass die theoretischen Garantien von PBRS für das vorliegende Szenario nicht gelten. Der Fokus dieses Kapitels liegt aber ohnehin darauf, ob ein System

eine gegebene Spezifikation erfüllt, was mittels Metriken unabhängig vom zu Grunde liegenden theoretischen Gerüst ermittelt wird. Und abschließend sei erwähnt, dass es wie in Kapitel 2.2.1 dargelegt derzeit keine Garantie für Optimalität gibt, wenn wertbasierte Algorithmen wie DQN eingesetzt werden, die NN zur Funktionsapproximation verwenden.

Rewardkomponente	Variable	Vorzeichen
Fertigstellung des Produkts	c_0	+
Abschluss eines Produktionsschritts	c_1	+
Zeitschritt mit unfertigem Produkt	c_2	-
Verwendung einer Maschine	c_3	-
Verlassen der erlaubten Wege	c_4	-
Kollision mit anderem Agenten	c_5	-
Missachten des Ausnahmesignals	c_6	-

(a) Rewardkomponenten

		Gewichtung der Komponenten						
		c_0	c_1	c_2	c_3	c_4	c_5	c_6
Rewardschema	rs_0	$5,0$		$0,1$				
	rs_1		$1,0$	$0,1$				
	rs_2		$1,0$	$0,1$	$0,2$			
	rs_3		$1,0$	$0,1$		$0,1$		
	rs_4		$1,0$	$0,1$			$0,4$	
	rs_5		$1,0$	$0,1$	$0,2$	$0,1$	$0,4$	$1,0$
	rs_x		$1,0$	$0,1$	$0-0,2$	$0,1$	$0-0,4$	$0-1,0$

(b) Rewardschemata

Tabelle 4.1: Übersicht über die Rewardschemata und -komponenten, die Anwendung erfolgt je Zeitschritt. Tabelle angepasst übernommen aus [122].

Inhaltlich wird zunächst davon ausgegangen, dass eine typische funktionale Anforderung an eine Smart Factory darin besteht, Produkte so schnell wie möglich fertigzustellen. Daher erhöht das erste Schema rs_0 das Potential um c_0, wenn ein Agent sein Produkt fertiggestellt hat, und verringert das Potential um c_2 pro Zeitschritt, den er bis dahin benötigt. Wenn mehrere Produktionsschritte an verschiedenen Maschinen in der richtigen Reihenfolge durchgeführt werden müssen, ist positives Feedback in rs_0 bei zufälliger Exploration sehr selten. Um das Lernverhalten zu verbessern, wird ein weiteres Schema rs_1 definiert, dass einem Agenten über c_1 jedes Mal positives Feedback gewährt, wenn er einen einzelnen Produktionsschritt an einer Maschine korrekt durchführt. Bei der Wahl der Werte von c_0 und c_1 wird die Anzahl der Produktionsschritte be-

rücksichtigt und sichergestellt, dass die Summe der Rewards bei Fertigstellung des Produkts gleich ist.

Industrielle Szenarien haben in der Regel auch eine Reihe von nicht-funktionalen Anforderungen, die von autonomen Agenten die Einhaltung bestimmter Regeln erfordern. Hier werden zwei Arten von Einschränkungen betrachtet. Die erste Art wird als *weiche* Einschränkung bezeichnet. Zu diesen gehört, dass die Agenten nur den für den Produktionsschritt erforderlichen Maschinentyp verwenden, auf den erlaubten Wegen bleiben und nicht mit anderen Agenten kollidieren. Die Simulation stellt unabhängig davon sicher, dass Agenten, die versuchen, miteinander zu kollidieren oder die erlaubten Wege verlassen, an Ort und Stelle bleiben, und Produkte nur von korrekten Maschinen verarbeitet werden. Daher stehen diese weichen Einschränkungen aus Sicht der Agenten nicht im Widerspruch zum Ziel der (schnellen) Fertigstellung von Produkten. Das Ziel ist hier, dass die Agenten selbst lernen, diese Einschränkungen bei der Optimierung zu berücksichtigen, beispielsweise, indem sie proaktiv zu einer nicht belegten Maschine navigieren. Die zweite Art von Einschränkungen wird als *harte* Einschränkungen bezeichnet. Zu diesen gehört, dass Agenten entweder stehen bleiben oder sich in sichere Positionen begeben, wenn ein Notfallsignal aktiv ist. Jedoch können Agenten den Notfall ignorieren, um ihre Aufgaben schneller zu erledigen. Daher verursachen die harten Einschränkungen einem Zielkonflikt. Hier wird untersucht, wie die Agenten diesen Zielkonflikt lösen. Da Missachtungen von Einschränkungen grundsätzlich minimiert werden sollen, werden diese in negative Terme übersetzt. c_3 stellt die Kosten für die Verwendung einer Maschine dar (praktisch wird $c_3 < c_1$ gewählt, sodass der Reward bei korrekter Benutzung einer Maschine positiv bleibt), c_4 die Strafe für das (versuchte) Verlassen der erlaubten Wege, c_5 die Strafe für eine (versuchte) Kollision mit einem anderen Agenten und c_6 die Strafe für das Missachten des Notfallsignals. Diese Terme sind in den Schemata rs_2, rs_3, rs_4 und rs_5 enthalten.

Inspiriert vom *Curriculum Learning* [11] und ähnlich dem Vorgehen in Kapitel 3.5 wird darüber hinaus ein dynamisches Schema rs_x definiert. Dieses enthält zu Beginn des Trainings nur die Terme c_1, c_2 und c_3, damit die Agenten zunächst nur einen Teil der Problemstellung lösen. c_4, c_5 und c_6 werden später im Training ergänzt, da die entsprechenden Anforderungen, beispielsweise die Vermeidung von Kollisionen, als schwer lernbar erachtet werden. Eine Bestrafung entsprechender Aktionen während der Exploration könnte den frühen Lernprozess behindern. Eine Zusammenfassung der Schemata, Komponenten und Werte befindet sich in Tabelle 4.1. Um die Schemata anwenden zu können, wurde je Komponente eine Metrik implementiert, beispielsweise $\mathit{AbgeschlosseneProduktionsschritte}(s_t)$, und mit der entsprechenden Gewichtung multipliziert, in diesem Beispiel c_1. Durch Aufsummierung all dieser Terme kann dann das Potential Φ jedes Zustands $s_t \in S$ berechnet werden:

$$\begin{aligned}\Phi(s_t) = c_0 \cdot \mathit{ProduktFertig}(s_t) + c_1 \cdot \mathit{AbgeschlosseneProduktionsschritte}(s_t) \\ + c_2 \cdot \mathit{VergangeneZeitschritte}(s_t) + c_3 \cdot \mathit{VerwendeteMaschinen}(s_t) \\ + c_4 \cdot \mathit{VerlasseneWege}(s_t) + c_5 \cdot \mathit{KollisionenMitAnderenAgenten}(s_t) \\ + c_6 \cdot \mathit{MissachtungenDesAusnahmesignals}(s_t)\end{aligned}$$

Abhängig vom RL-Algorithmus wird diese Potenzialfunktion je Agent (DQN) oder global (VDN, QMIX) angewendet. Wichtig ist, dass Agenten während des Trainings mit unterschiedlichen Rewardschemata nicht über den Return miteinander verglichen werden (können). Stattdessen wird auf die einzelnen Metriken zurückgegriffen, beispielsweise $\mathit{AbgeschlosseneProduktionsschritte}(s_t)$, die pro Episode über alle Agenten aufsummiert werden. Dieses Vorgehen wurde auch von Dulac-Arnold at al. [36] im Kontext der Berücksichtigung von Sicherheitsanforderungen einzelner RL-Agenten vorgeschlagen. Hier wird das (kooperative) MAS als Ganzes betrachtet, was den Einsatz der Technik *Parameter Sharing* (PS) erlaubt (Details im folgenden Kapitel 4.4.3). Eine Betrachtung einzelner Agenten würde nur Sinn machen, wenn diese unterschiedliches Verhalten entwickeln könnten, beispielsweise durch den Einsatz unterschiedlicher Techniken oder dem Training von individuellen Strategien.

4.4.3 Experimentelles Setup

Beim Testen der in Tabelle 4.1b dargestellten Schemata auf verschiedenen Layouts der Smart Factory erwies sich das Layout aus Abbildung 4.1 als guter Kompromiss hinsichtlich Komplexität und Lernbarkeit. Jeder Agent muss nach dem Start im Eingang (links) vier Produktionsschritte durchführen, die aus zwei Gruppen zu je zwei zufälligen Aufgaben bestehen. Um das Produkt fertigzustellen, muss er sich anschließend zum Ausgang (rechts, Spiegelposition des Eingangs) begeben. Wenn ein Agent sein Produkt fertiggestellt hat, wird er für den Rest der Episode aus der Simulation entfernt. Eine Episode endet, wenn alle Produkte fertiggestellt sind oder das Zeitlimit von 50 Schritten erreicht ist. Die Maschinen in der Smart Factory sind nicht nach Typ gruppiert, sondern gleichmäßig verteilt. So ist ausgeschlossen, dass ein einzelner nicht optimal handelnder Agent den Zugang zu allen Maschinen eines Typs blockiert. Dennoch können mehrere nicht optimal handelnde Agenten insbesondere in der Exploration Blockaden verursachen, die nur durch (emergente) Koordination gelöst werden können.

Die Basis bilden DQN-Agenten, die gemäß dem IL Paradigma unter Nutzung von PS trainiert werden (siehe Kapitel 2.2.3). Dies ist ein White-Box Test für den gewählten Ansatz: Die Agenten können das Feedback in Form individueller Rewards direkt mit ihren Aktionen in Verbindung bringen. Die Architektur und Parameter der Implementierung sind in Tabelle 4.2) aufgelistet. Mit Ausnahme der Experimente in Unterkapitel 4.5.3 gilt: $\gamma = 0,95$.

DQN	Parameter	Wert
Algorithmus		
	Aktionsraum	diskret
	Aktionswahl	ϵ-greedy
	Verringerung der Exploration	linear, pro Schritt
	Startwert	1,0
	Endwert	0,001
	Endwert erreicht bei Episode	1.000
	Diskontierungsfaktor γ	$0{,}95 - 1{,}0$
Architektur		
	Dimension innere MLP Schicht 1	64
	Dimension innere MLP Schicht 2	32
	Aktivierungsfunktion	ELU
	Loss Funktion	Huber
	Optimierer	Adam
	Max. Lernrate des Optimierers	0,005
Training		
	Dauer in Episoden	$5.000 - 6.000$
	Schritte pro Episode	50
	Bufferkapazität in Schritten	20.000
	Schritte bis Target Update	4.000
	Schritte pro Mini Batch	64

Tabelle 4.2: Details und Parameter der DQN Implementierung dieses Kapitels.

Zusätzlich werden Agenten mit VDN und QMIX gemäß dem CTDE Paradigma trainiert (siehe Kapitel 2.2.3). Dies ist ein Black-Box-Test für den gewählten Ansatz: Zu Beginn des Trainings können die Agenten das Feedback in Form der (globalen) Summe der Rewards nicht direkt mit ihren Aktionen in Verbindung bringen. Gemäß der Theorie ändert sich das mit zunehmendem Training durch das Erlernen der Faktorisierung, was die Koordination erleichtern könnte: Effekte, die das Handeln eines Agenten auf andere Agenten hat (beispielsweise die häufige Verwendung einer bestimmten Maschine), können dabei direkt berücksichtigt werden. Für die lokale Approximation der Q-Funktion verwenden VDN- und QMIX-Agenten die selbe Architektur und Parametrisierung wie die DQN-Agenten (siehe Tabelle 4.2). Für die erlernte, nichtlineare Faktorisierung der Wertfunktion verwendet die QMIX Implementierung ein neuronales Netz, das über eine einzelne innere Schicht mit 64 Einheiten und ELU-Aktivierungsfunktion verfügt.

In den Rewardschemata rs_0 bis rs_5 wird die Gewichtung der Komponenten während des Trainings nicht verändert. In Schema rs_x wird die Gewichtung

von c_4, c_5 und c_6 während des Trainings verändert, gegebenenfalls mehrfach. Dies wird in den jeweiligen Ergebnissen gesondert dargestellt. Um den Agenten die Anpassung auf die veränderte Rewardfunktion zu erleichtern, wird die Explorationsrate jedes mal auf 25% gesetzt und dann erneut linear verringert. Zusätzlich wird das Momentum des Optimierers vollständig zurückgesetzt. Für jede in den Ergebnissen dargestellte Kombination aus Rewardschema und RL-Algorithmus wird das Training der Agenten zehn Mal unabhängig voneinander durchgeführt. Berichtet werden jeweils den Mittelwert sowie das 95% Konfidenzintervall.

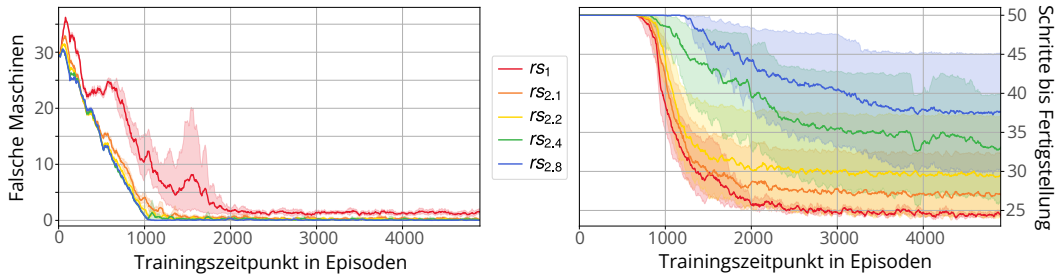
4.5 Auswertung

Nachfolgend wird die experimentelle Überprüfung des gewählten Ansatzes in der simulierten Smart Factory beschrieben. Zunächst untersucht Unterkapitel 4.5.1 den Einfluss isolierter Rewardkomponenten, die unterschiedliche Anforderungen einer Spezifikation abbilden, auf das Training von vier Agenten. Das folgende Unterkapitel 4.5.2 zeigt, wie der Ansatz auf acht Agenten skaliert werden kann, und Unterkapitel 4.5.3 zeigt, wie Agenten lernen können, zusätzliche Sicherheitsanforderungen zu berücksichtigen, die zeitweise im Konflikt mit anderen Zielen stehen. Abschließend demonstriert Unterkapitel 4.5.4, dass Agenten zur Laufzeit proaktiv sicher handeln, wenn sie im Training mit zusätzlichen Sicherheitsanforderungen konfrontiert wurden.

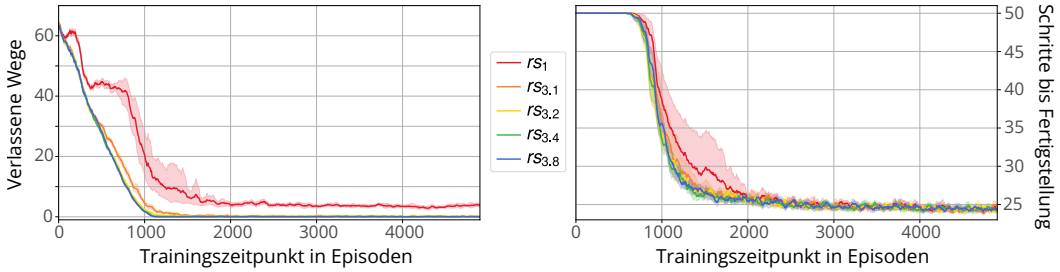
4.5.1 Analyse einzelner Komponenten

Insgesamt können die Rewards sieben verschiedene Aspekte der Spezifikation abbilden (siehe Tabelle 4.1), wobei die Aspekte unterschiedlich gewichtet werden können. Im nun folgenden ersten Szenario wird quantifiziert, wie die einzelnen Komponenten das Training der RL-Agenten beeinflussen. Als Basis für die Vergleiche werden vier DQN-Agenten verwendet, die mit Schema rs_1 trainiert werden, das lediglich funktionale Anforderungen enthält und häufig positives Feedback gewährt. Hiermit werden die Schemata rs_2, rs_3 und rs_4 verglichen, die zusätzlich jeweils eine nicht-funktionale Anforderung in unterschiedlicher Gewichtung enthalten. Die Gewichtung ist in der Legende angegeben, wobei beispielsweise $rs_{2.x}$ bedeutet, dass rs_2 mit $0,x$ gewichtet wird. Schema rs_5 enthält die funktionalen Anforderungen von rs_1 und die nicht-funktionalen Anforderungen von rs_2, rs_3 und rs_4 jeweils in der Gewichtung, die sich zuvor als praktikabel erwiesen hat. Mit Schema rs_0 wird getestet, ob Agenten auch mit seltenem positiven Feedback erfolgreich trainiert werden können.

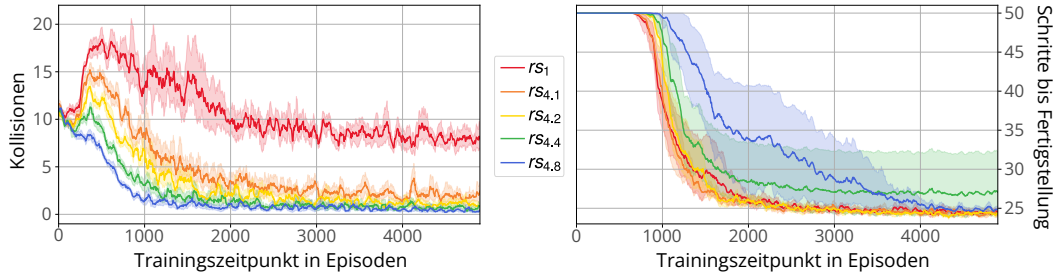
Als *Leistung* wird die Fähigkeit der Agenten bezeichnet, die funktionalen Anforderungen zu erfüllen. Hier wird globalen Zeitpunkt (Schritt innerhalb der



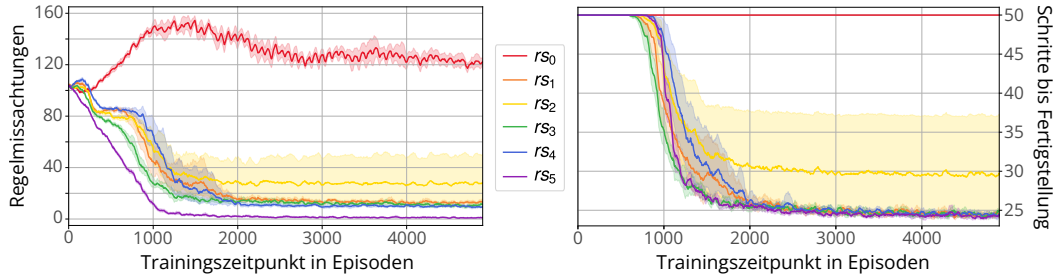
(a) Einfluss der Gewichtung von rs_2 auf Konformität (links) und Leistung (rechts)



(b) Einfluss der Gewichtung von rs_3 auf Konformität (links) und Leistung (rechts)



(c) Einfluss der Gewichtung von rs_4 auf Konformität (links) und Leistung (rechts)



(d) Einfluss statischer Schemata auf Konformität (links) und Leistung (rechts)

Abbildung 4.2: Einfluss einzelner Rewardkomponenten auf Leistung und Konformität von vier DQN-Agenten im Training. rs_0 und rs_1 bilden nur funktionale Anforderungen ab, wobei rs_1 häufiger positives Feedback gewährt. rs_2, rs_3 und rs_4 bilden zusätzlich je eine nicht-funktionale Anforderung mit variierender Gewichtung ab. rs_5 kombiniert die Anforderungen von rs_1 bis rs_4. Darstellungen aus [122], Layout angepasst, Beschriftungen übersetzt.

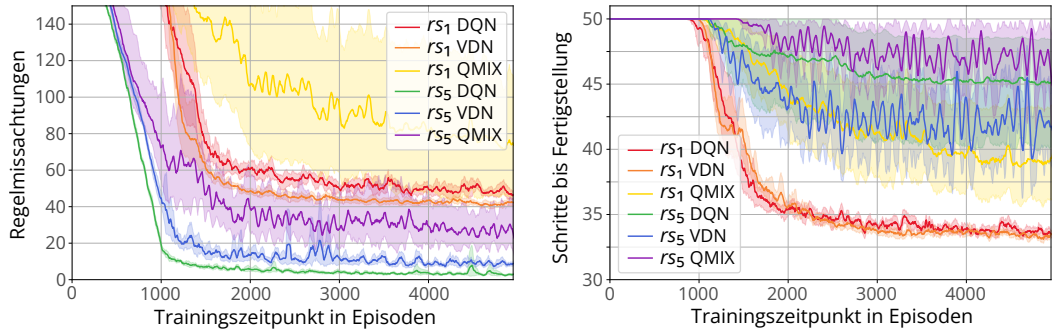
Episode) erfasst, zu dem alle Agenten ihr jeweiliges Produkt fertiggestellt haben, niedrigere Werte sind besser. Als *Konformität* wird die Fähigkeit der Agenten bezeichnet, die nicht-funktionalen Anforderungen zu erfüllen, die zusätzliche Bedingungen an das Verhalten stellen. Ein Beispiel ist die Vermeidung von Kollisionen mit anderen Agenten. Hier wird jegliche (versuchte) Missachtung erfasst und über alle Agenten und alle Schritte einer Episode aufsummiert, niedrigere Werte sind besser. Die Ergebnisse dieses ersten Szenarios sind in Abbildung 4.2 dargestellt.

Eine erste Beobachtung ist, dass Agenten mit allen Schemata außer rs_0 erfolgreich trainiert werden können. Bezüglich der Konformität war zu erwarten, dass die Agenten bei gezielter Bestrafung (negative Rewards) für bestimmte Aktionen in bestimmten Zuständen lernen, diese zu minimieren. In den Abbildung 4.2a und 4.2b ist zu sehen, dass Agenten die Verwendung falscher Maschinen und das Verlassen der Wege bereits bei geringer Gewichtung der entsprechenden Komponenten minimieren. Aus Abbildung 4.2c geht hervor, dass die Agenten Kollisionen mit anderen Agenten nur minimieren, wenn die entsprechenden Komponenten hoch gewichtet werden. Bezüglich der Leistung kann beobachtet werden, dass eine Bestrafung des Verlassens von Wegen keinen signifikanten Einfluss hat (Abbildung 4.2a) und das Bestrafen von Kollisionen mit anderen Agenten die Leistung nur bei hoher Gewichtung verringert (Abbildung 4.2c). Das Bestrafen der Verwendung falscher Maschinen hingegen führt dazu, dass die Agenten das Szenario umso langsamer lösen, je höher die Gewichtung ist (Abbildung 4.2b).

Die wichtigsten Ergebnisse dieses Experiments sind in Abbildung 4.2d dargestellt: Rewardschema rs_5, eine Kombination der zuvor getesteten Schemata, ist in der Lage, die Konformität signifikant zu erhöhen, ohne dabei einen signifikanten Verlust an Leistung zu verursachen.

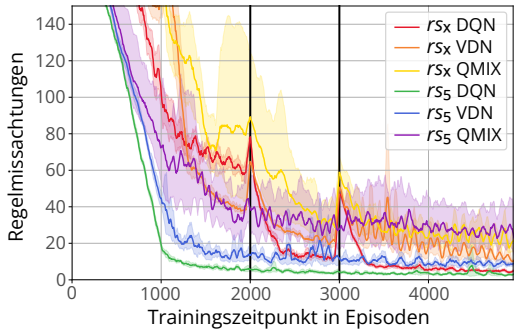
4.5.2 Skalierung auf acht Agenten

Mit dem Wissen, wie isolierte und kombinierte Rewardkomponenten das Training von RL-Agenten im vorliegenden Szenario beeinflussen, schwenkt der Fokus nun auf die Skalierung. Dazu werden acht Agenten mit den RL Algorithmen DQN, VDN und QMIX trainiert und mit den Rewardschemata rs_1 und rs_5 verglichen. Dabei kommen die bekannten Metriken *Leistung* (Schritte bis zur Fertigstellung aller Produkte, weniger ist besser) und *Konformität* (Missachtung von Verhaltensregeln, weniger ist besser) zum Einsatz. Die Ergebnisse sind in der oberen Zeile in Abbildung 4.3 dargestellt. Auch mit acht Agenten führt rs_5 zu einer höheren Konformität als rs_1 (siehe Abbildung 4.3b), wobei die RL Algorithmen unterschiedlich auf die Schemata regieren. VDN erzielt mit rs_1 eine höhere Konformität als DQN, mit rs_5 jedoch eine niedrigere. QMIX erzielt sowohl mit rs_1, als auch mit rs_5 die niedrigste Konformität der drei RL Algorithmen. Anders als im vorherigen Unterkapitel geht die höhere Konfor-

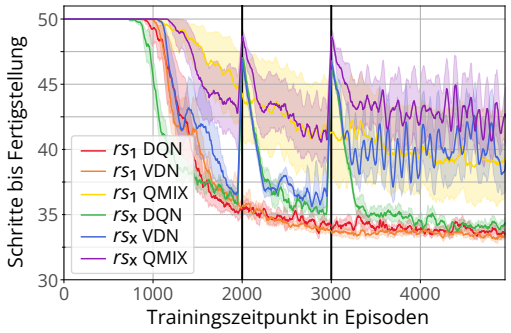


(a) Konformität: rs_1 vs rs_5

(b) Leistung: rs_1 vs rs_5



(c) Konformität: rs_5 vs rs_x



(d) Leistung: rs_1 vs rs_x

Abbildung 4.3: Skalierung des Trainings auf acht DQN-, VDN- und QMIX-Agenten mit unterschiedlichen Rewardschemata. rs_1 ist ein statisches Schema, das nur funktionale Aspekte der Spezifikation abbildet. rs_5 bildet zusätzlich nicht-funktionale Aspekte ab. Obere Reihe: Verglichen mit rs_1 führt rs_5 zu weniger Constraint-Verletzungen, aber zu mehr Schritten, bis die Umgebung gelöst ist. Untere Reihe: Das Hinzufügen von Rewardkomponenten in den Episoden 2.000 und 3.000 über rs_x erreicht fast die Konformität von rs_5 und die Leistung von rs_1. Darstellungen übernommen aus [122], Beschriftungen übersetzt.

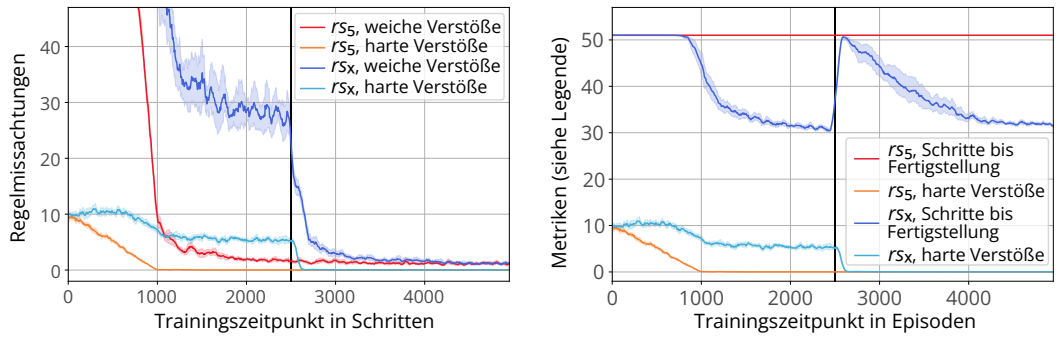
mität von rs_5 hier mit einer niedrigeren Leistung einher: In Abbildung 4.3b ist zu sehen, dass die Agenten unabhängig vom RL-Algorithmus mit rs_5 während des gesamten Trainings länger brauchen, bis alle Produkte fertiggestellt sind.

Das Ziel ist, die Leistung von Schema rs_1 und die Konformität von Schema rs_5 zu erreichen. Im vorherigen Unterkapitel wurde gezeigt, dass vier DQN-Agenten (auch) mit Schema rs_5 eine hohe Leistung erzielen können. Eine gleichzeitige Erfüllung funktionaler und nicht-funktionaler Anforderungen sollte im vorliegenden Szenario ebenso lernbar sein, für die doppelte Anzahl von acht Agenten könnte aber anderes Lernsignal erforderlich sein, insbesondere um den frühen Trainingsprozess (Exploration) zu unterstützen. Diese Hypothese wird überprüft, indem das Training der Agenten mit Belohnungen für die Erfüllung funktionaler Anforderungen der Spezifikation beginnt und die

Bestrafungen für das Missachten nicht-funktionaler Anforderungen nach und nach ergänzt wird. Dies wird mit Rewardschema rs_x realisiert. Die entsprechenden Ergebnisse sind in der unteren Zeile von Abbildung 4.3 dargestellt. Die senkrechten schwarzen Striche in den Datenreihen bei Episode 2.000 und 3.000 signalisieren, dass hier die Rewardkomponenten von rs_x verändert werden und anschließend die Exploration erhöht wird. In Abbildung 4.3c ist zu sehen, dass DQN und VDN mit rs_x fast die Konformität von rs_5 am Ende des Trainings erreichen. QMIX konvergiert langsamer, aber auch auch hier ermöglicht rs_x eine hohe Konformität. In Abbildung 4.3d ist zu sehen, dass DQN mit rs_x fast die Leistung rs_1 erzielt. Im Gegensatz dazu unterbricht rs_x die Konvergenz VDN und das Leistungsniveau von rs_1 wird nicht erreicht. QMIX zeigt mit rs_x das gleiche Verhalten. Dennoch führt rs_x hier insgesamt zu einer besseren Leistung als rs_5.

Die wichtigste Erkenntnis aus diesem Experiment ist, dass das mit vier Agenten erfolgreiche, statische Belohnungsschema rs_5 mit acht Agenten keine optimale Leistung mehr erzielt. Das dynamische Belohnungsschema rs_x verringert dieses Skalierungsproblem und ermöglicht sowohl das Erlernen von Konformität auf dem Niveau von rs_5, als auch (je nach RL-Algorithmus) Leistung auf dem Niveau von rs_1.

4.5.3 Abbildung von Sicherheitsaspekten



(a) Konformität (b) Leistung

Abbildung 4.4: Training von sechs DQN-Agenten mit zusätzlichen Sicherheitsanforderungen. Rewardschema rs_x ergänzt bei Episode 2.500 die entsprechenden Rewardkomponenten. Rewardschema rs_5 beinhaltet die selben Komponenten wie rs_x, gewichtet diese jedoch von Beginn des Trainings an vollständig. Die Agenten sollen bei aktivem Notfallsignal stehen bleiben (*harte Anforderung*). Alle übrigen nicht-funktionalen Anforderungen sind *weiche Anforderungen*. Darstellungen aus [122], Beschriftungen übersetzt.

Die gleichzeitige Maximierung von Leistung und Konformität bei steigender Anzahl Agenten im vorherigen Experiment war nicht trivial, erfordert aus Sicht der Agenten aber keine Kompromisse. Beispielsweise verliert ein Agent durch die Verwendung einer falschen Maschine einen Zeitschritt gegenüber der schnellstmöglichen Fertigstellung seines Produkts und hat dadurch eine inhärente Motivation, dies zu vermeiden. Und anstelle einer (versuchten) Kollision mit einem anderen Agenten kann er abwarten, bis dieser die Position freigibt, ohne einen Nachteil davon zu haben. In der Praxis gibt es jedoch auch Szenarien, in denen nicht alle Anforderungen gleichzeitig maximiert werden können. Daher wird nun ein Zielkonflikt eingeführt: Wenn das Notfallsignal aktiviert wird (zufällig, für einzelne Schritte), sollen die Agenten stehen bleiben, was der schnellstmöglichen Fertigstellung ihrer Produkte entgegensteht. Das Notfallsignal wird den Agenten über die Beobachtung zugänglich gemacht. Ziel ist, dass die Agenten lernen, die Einhaltung des Notfallsignal gegenüber allen anderen Zielen zu priorisieren. Um Probleme bei der Skalierung wie im vorherigen Szenario vorzubeugen, wird die Anzahl der Agenten auf sechs reduziert. Zudem werden diese mit DQN trainiert, da dieser Algorithmus in der Smart Factory bisher die besten Ergebnisse erzielt hat. Verglichen werden die Rewardschemata rs_5 und rs_x, die jeweils alle nicht-funktionalen Anforderungen abbilden. Die Modellierung des Notaus ist aus Sicht der RL Safety Community besonders relevant. Um Feedback hinsichtlich etwaiger theoretischer Garantien von PBRS nachzukommen, werden die Agenten hier einmalig mit $\gamma = 1,0$ parametrisiert und wie von Devlin et al. [33] vorgeschlagen am Ende jeder Episode in einen Zustand mit neutralem Potenzial versetzt, wodurch sich die Anzahl der Schritte auf 51 erhöht. Wenn zusätzlich eine theoretische Konvergenzgarantie zu einem lokalen Optimum gewünscht wäre, müsste wie in Kapitel 4.3.2 dargelegt ein Policy Gradient Algorithmus verwendet und den Agenten Zugriff auf sämtliche Informationen der Umgebung gegeben werden. Praktisch wird darauf verzichtet, da bei entsprechenden Tests eine Verschlechterung des Lernverhaltens beobachtetet wurde. Dies wurde auf den Beobachtungsraum zurückgeführt, der durch die Produktionslisten der anderen Agenten exponentiell anwuchs.

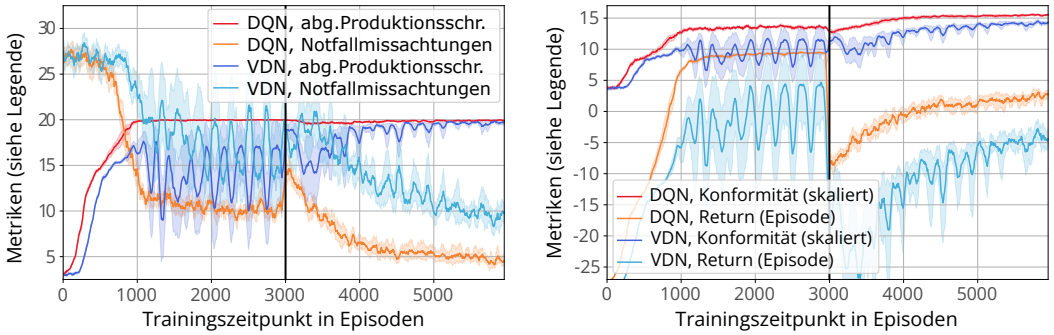
In Abbildung 4.4a ist zu sehen, dass DQN mit Rewardschema rs_5 lernt, Missachtungen weicher Einschränkungen zu minimieren und Missachtungen harter Einschränkungen vollständig zu vermeiden. Gemäß Abbildung 4.4b gelingt es DQN so jedoch nicht, das Szenario in weniger als 50 Schritten erfolgreich abzuschließen. Eine qualitative Überprüfung der Strategie der entsprechenden Agenten ergab, dass diese nach der Exploration unabhängig vom Notfallsignal an Ort und Stelle bleiben und sich nicht mehr bewegen. Die Verwendung des Rewardschemas rs_x führt bei Episode 2.500 zu einer charakteristischen Spitze in den Trainingsverläufen, die auf die Veränderung der Rewardkomponenten zurückzuführen ist. Zu diesem Zeitpunkt ergänzt rs_x eine Bestrafung für die Missachtung harter Einschränkungen. Dadurch kann rs_x den Zielkonflikt auflösen: Die entsprechend trainierten Agenten bleiben auf dem zuvor erreichten

Leistungsniveau und lernen dann, die harten Anforderungen einzuhalten, indem sie sich immer dann nicht bewegen, wenn das Notsignal aktiv ist.

4.5.4 Spezifikationskonformes Laufzeitverhalten

Aufbauend auf dem vorherigen Unterkapitel sollen die Agenten in den folgenden Experimenten nicht mehr stehen bleiben, sondern sich innerhalb einer begrenzten Zeit zu definierten Positionen bewegen, wenn das Notfallsignal aktiv ist. Das Notfallsignal bleibt eine entsprechende Anzahl von Schritten aktiv und wird den Agenten entsprechend über die Beobachtung signalisiert. In diesem Szenario werden die Notfallpositionen in einem zusätzlichen Kanal in der Beobachtung der Agenten ergänzt. Die Notfallpositionen sind unabhängig vom Status des Notfallsignals durchweg sichtbar. Grund dafür ist, dass die Notfallpositionen während des Trainings zufällig gesetzt werden und die Agenten in der Lage sein sollen, die Notfallpositionen beim Lernen der optimalen Route(n) durch die Produktionsstätte zu berücksichtigen. Erneut können die Agenten entscheiden, das Notfallsignal zu ignorieren. Die Motivation hierfür ist, dass es interessiert ist, zu sehen, welches Verhalten die Agenten lernen, um die funktionalen und nicht-funktionalen Anforderungen dieses Szenarios (mathematisch) zu optimieren. Während des Trainings werden die Agenten mit zufälligen Kombinationen von höchstens vier möglichen Notfallpositionen pro Episode konfrontiert. Die Kapazität der Notfallpositionen wird so angepasst, dass alle Agenten im Notfall Platz finden. Die möglichen Positionen sind in Abbildung 4.6a rot markiert. Ist das Notfallsignal inaktiv, verhalten sich die Notfallpositionen wie normale Ausweichbereiche. Insbesondere können die Agenten sie dann nicht nutzen, um einander auszuweichen, was das Routing schwieriger macht als zuvor.

Abbildung 4.5 stellt den Trainingsverlauf von vier Agenten mit den RL-Algorithmen DQN und VDN mittels Rewardschema rs_x dar, das eine Bestrafung für die Missachtung des Notfallsignals ab Episode 3.000 ergänzt. Die linke Abbildung zeigt die Entwicklung der Werte zweier globaler Metriken der Simulation: *Abgeschlossene Produktionsschritte* (tasks completed, mehr ist besser) und *Missachtungen des Notfallsignals* (emergency violations, weniger ist besser). Wie im vorhergegangenen Experiment lernen die Agenten, die Anzahl der Missachtungen des Notfallsignals zu verringern, ohne dass sich die Anzahl der abgeschlossenen Produktionsschritte signifikant verringert. VDN hat Schwierigkeiten, das Niveau von DQN zu erreichen, was auf eine (noch) nicht optimale Faktorisierung des globalen Rewards zurückzuführen ist. Es ist davon auszugehen, dass lokale Rewards für die Agenten ein leichter zu interpretierendes Feedback liefern. Die rechte Abbildung zeigt die Entwicklung zweier globaler Metriken der Agenten: *Return* (globale Summe aller Rewards, mehr ist besser) und *Konformität* (compliance, mehr ist besser). Abweichend von früheren Experimenten misst *Konformität* hier sowohl ein funktionales Ziel (alle Agenten



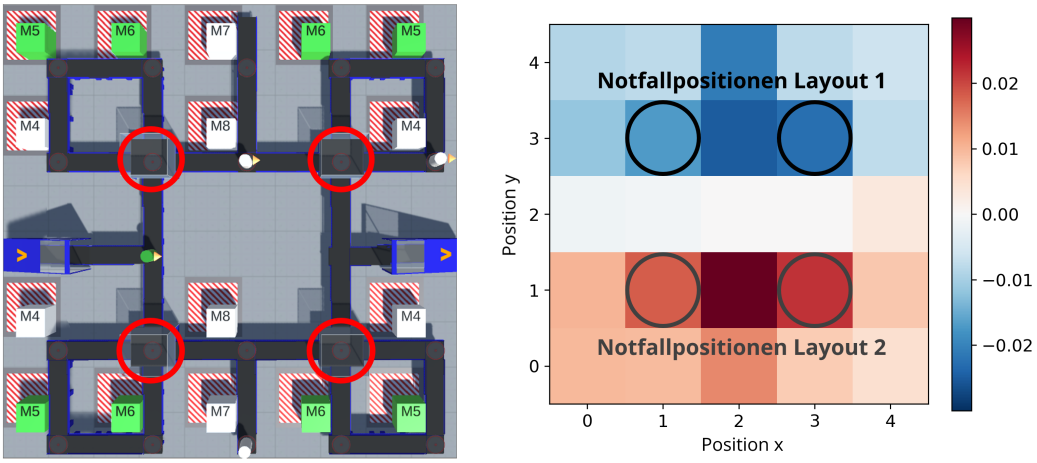
(a) Abgeschlossene Produktionsschritte und Missachtungen des Notfallsignals

(b) Return und Konformität

Abbildung 4.5: Trainingsverlauf von vier DQN- und VDN-Agenten mit Rewardschema rs_x und den Notfallpositionen aus Abbildung 4.6a. Ab Episode 3.000 bestraft rs_x die Missachtung des Notfallsignals. Links: Die Agenten lernen, die Anzahl Schritte außerhalb sicherer Positionen bei aktivem Notfallsignal zu verringern, ohne dass sich dies negativ auf die Anzahl der abgeschlossenen Produktionsschritte auswirkt. Rechts: Einbruch der Rewards nach der Ergänzung der Bestrafung für Missachtung der Notfallsignale. Darstellungen aus [122], Beschriftungen übersetzt.

sollen ihr Produkt innerhalb von 50 Zeitschritten fertig stellen), als auch ein nicht-funktionales Ziel (alle Agenten sollen innerhalb von vier Zeitschritten an einer Notfallposition sein, wenn das Notfallsignal aktiv wird) und gewichtet die Erfüllung beider Ziele gleich. Ein höherer Wert entspricht einem größeren Grad der Erfüllung der Anforderungen. Von Belang ist die Entwicklung über die Zeit. Für eine bessere Lesbarkeit im Diagramm wurde der Wert (linear) skaliert. Interessant ist, dass der Return einbricht, nachdem die Bestrafung für die Missachtung von Notfällen ergänzt wird, aber die Konformität weiter steigt. Dies ist ein starkes Indiz dafür, dass die Rewards ungeachtet des absoluten Wertes ermöglichen, dass die Agenten im Training das beabsichtigte, spezifikationskonforme Verhalten erlernen.

Um das Laufzeitverhalten zu untersuchen, werden die durchschnittlichen Positionen der Agenten mit zwei unterschiedlichen Kombinationen von Notfallpositionen verglichen. Diese befinden sich entweder in der oberen oder in der unteren Hälfte der Smart Factory. Gemessen werden die durchschnittlichen Agentenpositionen von zehn unabhängig trainierten Teams von Agenten über 100 Evaluierungsepisoden pro Team und Kombination aus Notfallpositionen. Abbildung 4.6b stellt den gemessenen Unterschied farblich dar. Hier ist zu erkennen, dass die Agenten häufiger diejenige Hälfte der Smart Factory durchqueren, die die jeweiligen sicheren Positionen enthält. Dies ist ein sehr interessantes Ergebnis, da Notfälle für diese Evaluation deaktiviert waren. Die Agenten haben also im Training gelernt, dass das Durchführen von Produkti-



(a) Mögliche Notfallpositionen (b) Aufenthaltswahrscheinlichkeiten

Abbildung 4.6: Links: Während des Trainings werden die Agenten mit zufälligen Kombinationen von maximal vier Notfallpositionen konfrontiert (rot markiert). Rechts: Differenz der Aufenthaltswahrscheinlichkeiten dieser Agenten während 100 Evaluierungsepisoden ausgehend von zwei unterschiedlichen Kombinationen von Notfallpositionen. Während der Evaluation gibt es keine Notfälle. Darstellungen aus [122], Beschriftungen übersetzt.

onsschritten in der Nähe der Notfallpositionen die durchschnittliche Zeit senkt, um im Notfall zu einer solchen Position zu gelangen. Dies führt hier zu einer Form von proaktiv sicherem Laufzeitverhalten.

Qualitativ wurde beobachtet, dass einzelne Agenten während einem Notfallsignal nicht zu sicheren Positionen, sondern zum Ausgang navigierten und damit ihre Produkt fertig stellten. Dies trat immer dann auf, wenn ein Agent nur noch zum Ausgang navigieren musste und die Notfallposition(en) weit entfernt war(en). Da Agenten nach der Fertigstellung ihrer Produkte für den Rest der Episode neutral behandelt werden, also keine Rewards bekommen, ist dieses Verhalten aus Optimierungssicht nachvollziehbar. In der Praxis müsste die Spezifikation hier definieren, ob ein Agent sich in solch einem Fall tatsächlich zum Ausgang bewegen soll. Qualitativ wurde ebenfalls beobachtet, dass Agenten von Zeit zu Zeit Maschinen verwendeten, die so weit von den Notfallpositionen entfernt sind, dass die Agenten bei Eintreten eines Notfalls die beispielhafte nicht-funktionale Anforderung verletzen würden. Die Agenten wägen dabei auf Basis der gesammelten Erfahrung ab, ob die zu erwartenden Belohnungen für den Abschluss von Produktionsschritten die zu erwartenden Strafen für das Missachten der Notfallsignale überwiegen, wobei die vorliegende Verteilung von Aufgaben, Maschinen und Notfallpositionen eine Rolle spielt. In der Praxis müsste die Spezifikation hier definieren, ob funktionale und nicht-funktionale Anforderungen Vorrang haben.

4.6 Diskussion

Zunächst zeigen die Ergebnisse, dass die gewählte Modellierung der Smart Factory geeignet ist, praxisrelevante Aspekte eines MAS, beispielsweise Koordination und Skalierung, zu untersuchen. Das verwendete Layout ist mit vier Agenten vergleichsweise leicht zu lösen und ermöglicht so aussagekräftige Experimente auch unter nicht optimalen Trainingsbedingungen. Jedoch scheitern naive Trainingsansätze an sechs und acht Agenten. Insbesondere hier sind deutliche (Leistungs-) Unterschiede zwischen verschiedenen RL-Techniken zu erkennen. Die Vermutung ist, dass die Agenten um Positionen konkurrieren: Einerseits, um über Wege zu Maschinen zu gelangen und andererseits um die Maschinen selbst, um dort die erforderlichen Produktionsschritte durchzuführen. Um gute Ergebnisse zu erzielen, müssen die dort entstehenden Konflikte gelöst werden, was ein koordiniertes Verhalten der Agenten erfordert. Das grundsätzlich gute Abschneiden von DQN gegenüber VDN und QMIX ist insofern nicht überraschend, als hier präzises individuelles Feedback pro Agent zur Verfügung steht und die homogenen Agenten den Einsatz von PS erlauben. Dies ist effizient, da die von allen Agenten generierten Daten für das Training einer gemeinsamen Strategie verwendet werden können. VDN und QMIX hingegen müssen die Zuordnung von globalem zu individuellem Feedback über die Faktorisierung erst lernen. Es ist möglich, dass die Faktorisierung im vorliegenden Szenario bei einer noch größeren Anzahl von Agenten einen Vorteil bringen würde, oder wenn kein individuelles Feedback verfügbar wäre. Analog zu den Ergebnissen aus Kapitel 3 konnte IL mit PS auch in diesem Kapitel erfolgreich eingesetzt und auf komplexere Ansätze verzichtet werden.

Hinsichtlich der Rewardschemata ist festzuhalten, dass seltenes Feedback wie in rs_0 den RL-Agenten nicht ermöglicht, das vorliegende Szenario in begrenzter Zeit zu lösen. Umso erfreulicher ist, dass das häufigere Feedback in rs_1 den Agenten in ausnahmslos allen Szenarien zu einer soliden Leistung hinsichtlich der Erfüllung funktionaler Anforderungen führt. Dass die Agenten mit rs_1 nicht lernen, Aktionen wie (versuchte) Kollisionen mit anderen Agenten zu minimieren (siehe Abbildung 4.2c), kann damit erklärt werden, dass dies aus Sicht der Agenten hier keinen Unterschied zum Warten macht. rs_1 bewertet Kollisionen mit anderen Agenten neutral und die Simulation führt solche Aktionen nicht aus. Bemerkenswert ist aber, dass Agenten mit rs_1 nicht lernen, ausschließlich Maschinen verwenden, die den aktuellen Produktionsschritt durchführen können (siehe Abbildung 4.2a). Hier verlieren sie einen Zeitschritt gegenüber der schnellstmöglichen Fertigstellung ihres Produktes. Gemäß der Ergebnisse benötigt es Schemata wie rs_2 oder rs_5, die detailliertere Informationen über Problemstellung und Ziel enthalten, hier repräsentiert durch die nicht-funktionalen Anforderungen der Spezifikation. Durch das Training mit solchen Schemata wurde in ausnahmslos allen Szenarien eine höhere Konformität der Agenten gegenüber der Spezifikation gemessen. Daraus kann gefolgert werden, dass lernenden Agenten die nicht-funktionale Anforderun-

gen, beispielsweise Präferenzen oder auch sicherheitsrelevante Aspekte, grundsätzlich zugänglich gemacht werden sollten, da nicht garantiert ist, dass sie dieses Verhalten durch Zufall entwickeln, selbst wenn die funktionalen Anforderungen die nicht-funktionalen Anforderungen subsummieren. Ebenso kann gefolgert werden, dass der Weg über das Feedback-Signal in Form der Rewards ein praktikabler Weg dafür ist.

Rewardschemata mit hoher Informationsdichte wie rs_5 haben aber auch Nachteile: Zunächst benötigen die Agenten in den durchgeführten Experimenten je nach Anzahl und RL-Algorithmus mehr Zeit, um das Problem zu lösen (siehe beispielsweise Abbildung 4.3c). Eine qualitative Auswertung von Stichproben ergab, dass die Agenten häufiger warten, als es erforderlich ist, also gewissermaßen übervorsichtig agieren. Je nach Szenario mag das jedoch einem unvorsichtigen Verhalten vorzuziehen sein. Gravierender ist, dass rs_5 den Zielkonflikt nicht lösen konnte, der in Abschnitt 4.5.2 eingeführt wurde. Daraus kann gefolgert werden, dass es nicht zwingend zum Erfolg führt, wenn die Agenten von Anfang an Zugang zur gesamten Spezifikation haben. Die Experimente mit adaptiven Rewardschemata wie rs_x (siehe beispielsweise Abbildung 4.4) zeigen aber, dass Leistung, Konformität und eine Einhaltung der gesamten Spezifikation erlernt werden können, wenn man mit einem reduzierten Rewardschema beginnt und schrittweise Komponenten ergänzt, sobald die Lernkurve der Agenten zu konvergieren beginnt. Manche Rewardkomponenten wirken sich demnach negativ auf die Exploration aus und sind ungeeignet, um das Training damit zu beginnen. Vergleichbare Nebeneffekte des Reward Shaping werden auch in der Literatur beschrieben [33]. Ausgehend von einem Schema wie rs_x ist die Wahl des RL-Algorithmus ein weiterer wichtiger Aspekt: Teilweise unterbricht rs_x die Konvergenz der CDTE-Algorithmen VDN und QMIX (siehe beispielsweise Abbildung 4.3d). Daraus kann geschlossen werden, dass auch der Zeitpunkt, zu dem Rewardkomponenten hinzugefügt werden, eine wichtige Rolle spielt und sich je nach Algorithmus unterscheiden kann.

Die Experimente der Unterkapitel 4.5.3 und 4.5.4 zeigen, dass Agenten mit RL lernen können, komplexe, nicht-funktionale Anforderungen einer Spezifikation wie Sicherheitsaspekte zu erfüllen. Als Beispiele dienten ein Notaus und eine sichere Notfallstrategie (Navigation zu dynamischen Sammelpunkten), was in Konflikt mit der schnellstmöglichen Fertigstellung ihrer Produkte steht. Die Agenten könnten entsprechende Notfallsignale ignorieren, jedoch zeigen die Ergebnisse, dass negative Rewards für die Missachtung der Notfallsignale ausreichen, dass sie im Notfall die Erfüllung der Sicherheitsanforderungen höher priorisieren als die Fertigstellung ihrer Produkte. Bemerkenswert ist, dass durch das Training mit dynamischen Sammelpunkten ein räumlicher Effekt zur Laufzeit messbar ist: Die Agenten steuern häufiger Maschinen an, die nahe an einem Sammelpunkt sind, auch wenn kein Notfall vorliegt. Dies wird als Form von proaktiv sicherem Laufzeitverhalten betrachtet, denn die Agenten werden nicht gezwungen, sich ausschließlich innerhalb eines bestimmten Abstands zu den Notfallpositionen zu bewegen. Jedoch entfernen sich die

Agenten je nach Verteilung der Aufgaben, Maschinen und anderen Agenten zeitweise weit von den Notfallpositionen. Sollte ein strikter räumlicher Effekt erwünscht sein, ist auf Basis der vorliegenden Ergebnisse anzunehmen, dass dieser mit einer stärkeren Gewichtung der entsprechenden Rewardkomponenten erzielt werden kann, wenn dieser mit fortlaufendem Training sukzessive erhöht wird. Hier besteht die Möglichkeit, die vorliegenden technische Realisierung mit Requirements-Engineering Methoden des *Goal Modeling* wie *i** [41, 83] zu kombinieren, die Abhängigkeiten zwischen einzelnen Anforderungen und den (gewünschten) Grad der Erfüllung präzise definieren. Die resultierenden Definitionen dürften gut zur Steuerung der iterativen Anpassung der Rewardkomponenten geeignet sein. Ein solches iteratives Vorgehen erfordert jedoch präzise Tests und ist nur praktikabel, wenn eine abgesicherte Trainingsumgebung, beispielsweise in Form einer Simulation, zur Verfügung steht, in der eine Verletzung der Spezifikation keine gefährlichen Folgen hat.

Eine Alternative zum vorliegenden Ansatz sind Fallback-Controller, die im Ausnahmefall die Kontrolle übernehmen [106]. Der Vorteil solcher Controller ist, dass sie je nach eingesetzter Technik bestimmte Eigenschaften oder Verhaltensweisen garantieren können und so Voraussetzungen für beweisbar sicheres Verhalten bieten [105]. Jedoch kann die Realisierung solcher Controller bei steigender Anzahl der Freiheitsgrade unpraktikabel werden. In der Smart Factory müssen willkürlich positionierte Agenten zu den nächstgelegenen sicheren Positionen unter Berücksichtigung der Kapazitätsgrenzen geleitet werden. Dabei ist zu beachten, dass die Sicherheitsanforderungen unerfüllbar werden können, wenn die Agenten im Normalfall nicht bereits proaktiv handeln: Auch ein optimaler Fallback-Controller kann die Agenten nicht schnell genug zu den Notfallpositionen leiten, wenn die Agenten zu weit entfernt sind. Also müsste die Strategie der RL-Agenten bereits im Normalbetrieb eingeschränkt werden. Eine solche Einschränkung müsste aber die Verteilung von Aufgaben, Maschinen und anderen Agenten berücksichtigen: Ein Stau würde jeder Abschätzung der Zeit bis zum Erreichen der Notfallpositionen auf Basis der Entfernung entgegenstehen. In der Smart Factory ist das Routing unter Berücksichtigung der Verteilung von Maschinen, Produktionsschritten und anderen Agenten ein Teil des Problems, das die RL-Agenten ohnehin lernen (sollen). Daher können die funktionalen und die nicht-funktionalen Anforderungen in Bezug auf das grundlegende Problem orthogonal sein. Es kann in solchen Fällen einfacher sein, die Fähigkeiten der Agenten auch zur Erfüllung der sicherheitsrelevanten Anforderungen zu verwenden, als eine Heuristik zu finden, die dies tut. Umgekehrt gilt, dass falls eine solche Heuristik oder eine exakte Definition der Einschränkungen zur Lösung der sicherheitsrelevanten Aspekte bekannt sind, diese ebenso zur Erfüllung der funktionalen Aspekte verwendet werden können, womit das Problem ohne RL gelöst werden kann.

4.7 Zusammenfassung

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf individueller und globaler Ebene lösen können. Diese Kapitel untersuchte, wie mehrere Agenten mittels RL lernen, ein zuverlässiges Gesamtsystem zu bilden. Besonderes Augenmerk lag auf der Erfüllung einer gegebenen Spezifikation, die performantes und gleichzeitig sicheres Verhalten in einem kooperativen Szenario fordert. Dafür wurden alle Anforderungen der Spezifikation in der Rewardfunktion abgebildet, um den Agenten die Spezifikation zugänglich zu machen und ihnen zu ermöglichen, ihr Verhalten daran selbst auszurichten. Die Agenten lernten, funktionale Aspekte wie die Durchführung mehrerer Produktionsschritte in begrenzter Zeit und nicht-funktionale Aspekte wie die Vermeidung von Kollisionen zu erfüllen. Zwei große Herausforderungen waren das Skalieren dieses Ansatzes auf acht Agenten und das Erlernen von Sicherheitsanforderungen, die ausgehend von einer Ausnahmesituation ein spezielles Verhalten von den Agenten erfordern, welches mit den funktionalen Anforderungen in Konflikt steht. Diese Herausforderungen wurden über ein Training mit adaptiven Rewardschemata gelöst, die initial eine reduzierte Menge funktionaler Anforderungen abbilden und weitere Anforderungen ergänzen, sobald die Lernkurve der Agenten zu konvergieren beginnt. Die so trainierten Agenten verhalten sich bereits im Normalfall so, dass sie im Ausnahmefall die Sicherheitsanforderungen besser erfüllen können, ohne dass die Performance im Normalfall signifikant einbricht, was gemäß Wooldrige et al. [160] als stark ausgeprägte Form proaktiven Verhaltens gewertet werden kann. Ein solches Verhalten ist über Ansätze wie Shielding [5] oder Fallback-Controller [105, 106] schwerer zu erzielen, die aufgrund der Trennung zwischen Normal- und Ausnahmefall entweder Performance oder Sicherheit auf Kosten des jeweils anderen Aspekts berücksichtigen. Wenn lernende Agenten Funktionsapproximation in Form von NN nutzen und dadurch kein Beweis für ein spezifisches Verhalten geführt werden kann, bietet der entwickelte Ansatz keine Sicherheitsgarantien. Aber er kann genutzt werden, um Anzahl und Komplexität zusätzlicher Sicherheitsschichten um autonome Agenten herum zu reduzieren, und trägt so dazu bei, das Problem der Ausrichtung [59] zu verringern.

5 Selbstanpassung

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf individueller und globaler Ebene lösen können. Dieses Kapitel beinhaltet den letzten von insgesamt drei Abschnitten, die verschiedenen Aspekten dieser These nachgehen. Es legt den Schwerpunkt auf die Beschreibung des gesamten Lebenszyklus von RL-Systemen (siehe Abbildung 1.1). Dieser umfasst mehrere Entwicklungszyklen, die aufeinander aufbauen und so eine iterative Anpassung ermöglichen, beispielsweise um veränderten Laufzeitbedingungen Rechnung zu tragen.

Dieses Kapitel beschäftigt sich mit der Frage, wie ein Entwicklungsprozess für ML-basierte Software definiert werden muss, um eine selbstständige Optimierung über mehrere Schritte hinweg zu ermöglichen. Dies wird am Beispiel eines formalen, grafischen Prozessmodells betrachtet. Eine Meta-Optimierung iteriert einzelne Entwicklungsschritte, beispielsweise die in Kapitel 3 betrachtete Definition der Umgebung oder die Konstruktion von Rewards aus Kapitel 4. An definierten Stellen des Entwicklungsprozesses können Methoden der Qualitätssicherung integriert werden, um den globalen Fortschritt zu messen und Feedbackschleifen gezielt anzusteuern. Besonderes Augenmerk liegt auf der Berücksichtigung von DL-Techniken, dem MLOps Paradigma, sowie den Abhängigkeiten zwischen Aktivitäten, beispielsweise Training und Test, sowie den beteiligten Artefakten, beispielsweise Spezifikation und ML-Modell.

Dieses Kapitel ist folgendermaßen strukturiert: Zunächst listet Unterkapitel 5.1 die Vorveröffentlichung auf und legt dar, welche Inhalte in die vorliegende Arbeit übernommen wurden und welche Beiträge die (Ko-)Autoren geleistet haben. Dann beschreibt Unterkapitel 5.2 Motivation und Zielsetzung des Prozessmodells und Unterkapitel 5.3 zeigt die entsprechende Lücke innerhalb der Literatur. Unterkapitel 5.4 beschreibt das entwickelte Prozessmodell. Es beginnt mit einer Schilderung der Prinzipien und Konzepte in Unterkapitel 5.4.1, gefolgt von einer formalen Definition in Unterkapitel 5.4.2. Anschließend erklärt Unterkapitel 5.4.3 die grafische Darstellung anhand eines beispielhaften ML-Entwicklungsprozesses, der den gesamten Lebenszyklus umfasst, und dann geht Unterkapitel 5.4.4 genauer auf die modellierten Aktivitäten ein. Das folgende Unterkapitel 5.4.5 bildet exemplarisch die in Kapitel 4 beschriebene Vorgehensweise mit der grafischen Darstellung ab. Anschließend diskutiert Unterkapitel 5.5 weitere Anwendungsmöglichkeiten des Prozessmodells und abschließend fasst Unterkapitel 5.6 die gewonnen Erkenntnisse zusammen.

5.1 Vorveröffentlichungen

Die zentralen Inhalte dieses Kapitels wurden im Rahmen einer Auftragsforschung für Siemens Technology erarbeitet und in [123] publiziert. Dieses Kapitel nimmt konzeptuell sowohl Bezug auf den in Kapitel 4 behandelten Ansatz SAT-MARL [121, 122], als auch auf das *Antagonist-Ratio Training Scheme* (ARTS) [109] sowie das *Scenario Co-Evolution* (SCoE) Paradigma [42]. Die Arbeiten an SCoE waren bereits im Gange, als der Autor der vorliegenden Arbeit zur Auftragsforschung hinzu kam, und ARTS wurde parallel zu SAT-MARL entwickelt. Bei SCoE und ARTS war der Autor der vorliegenden Arbeit nur unterstützend tätig, was die Reihenfolge der Autoren dieser Publikationen auch angemessen abbildet. SCoE und ARTS sind kein Beitrag (zu) dieser Dissertation.

Die Publikation mit dem Titel *Capturing Dependencies within Machine Learning via a Formal Process Mode* [123] schlägt ein formales Prozessmodell für die Entwicklung von ML-basierter Software dar, das die Mehrheit der in der Literatur beschriebenen Aktivitäten und Artefakte auf konsistente Weise vereint. Es berücksichtigt deren Zusammenwirken und Verwendung im Kontext der Qualitätssicherung – beispielsweise mit welchen Daten ein ML-Modell gegen welche Version der Spezifikation getestet werden muss, um den weiteren Verlauf der Entwicklung zu steuern. Ein wichtiges Merkmal ist eine Beschreibung der Zeit, die es erlaubt, den Fortschritt innerhalb von ML-Entwicklungsprozessen präzise darzustellen. Das Prozessmodell zielt darauf ab, ML-Modelle während des gesamten Lebenszyklus weiterzuentwickeln, insbesondere nach dem initialen Training und Testen, und skizziert, wie dieser Vorgang automatisiert werden könnte, um eine Meta-Optimierung zu realisieren.

Der Autor der vorliegenden Arbeit entwickelte das Konzept und die grafische Syntax des Prozessmodells in iterativer Vorgehensweise und verfasste den Großteil der Publikation. Thomas Gabor trug das Kapitel *Formalization* bei. Thomy Phan, Thomas Gabor, Andreas Sedlmeier und Philipp Altmann unterstützten bei der iterativen Weiterentwicklung der grafischen Syntax mit Feedback und Verbesserungsvorschlägen. Jan Wieghardt, Reiner Schmid, Horst Sauer und Cornel Klein gaben den Fokus der Auftragsforschung vor und halfen in regelmäßigen Diskussionen, das Prozessmodell hinsichtlich Darstellung und Nomenklatur leichter verständlich zu gestalten. Claudia Linnhoff-Popien stand für inhaltliche und strukturgebende Diskussionen der vorgestellten Konzepte zur Verfügung. Die Inhalte dieser Publikation fließen in die nachfolgenden Kapitel 5.2 bis 5.5 ein.

5.2 Anpassung durch automatisierte ML-Entwicklung

In den beiden vorhergegangenen Kapiteln verwendeten die lernenden Agenten DL zur Funktionsapproximation, eine ML-Technik. Dies war einer der Schlüssel, um *Selbstorganisation* und *Selbstausrichtung* in Multi-Agenten-Szenarien zu realisieren, und lässt hoffen, dass entsprechend entwickelte Systeme auch in unvorhergesehenen Situationen zuverlässig agieren. In der Praxis entstehen durch das iterative Vorgehen bei der Erstellung von ML-Komponenten wie lernender Agenten aber auch zusätzliche Herausforderungen, die nicht nur technisch geprägt sind, beispielsweise im Bereich der Entwurfsmethoden des *Software Engineering* (SE). Beispielsweise sollte berücksichtigt werden, dass Daten im laufenden Betrieb gesammelt werden, um eine *Anpassung* künftiger Iterationen dieser ML-Komponenten zu ermöglichen. Das kann wichtig sein, wenn das ursprüngliche Training die Laufzeitbedingungen nicht exakt abgebildet hat (*Sim2Real Gap*) oder sich diese langsam verändern (*Drift*).

Ein Schlüsselaspekt hierbei ist die Automatisierung. Fehlerbehebungen oder Verbesserungen werden häufig manuell von Entwicklern durchgeführt. Um die Nebeneffekte zu minimieren, die bei den entsprechenden Veränderungen entstehen, streben moderne Vorgehensweisen der ML-Entwicklung wie *MLOps* [75, 91] kleine, aber häufige Veränderungen mit Hilfe funktionsübergreifender Zusammenarbeit und der Automatisierung von Integration und Deployment an. Jegliche Software sollte Anwender zur Laufzeit bestmöglich unterstützen und so direkt oder indirekt für einen Effizienzgewinn sorgen. Ideal wäre, menschlichen Aufwand nicht nur hier, sondern über den gesamten Lebenszyklus der Software hinweg zu minimieren. Wie in Kapitel 2.5 skizziert, sind ML-Systeme in der Lage, sich durch Generalisierung an Veränderungen anzupassen und können mit geringem menschlichen Aufwand neu trainiert werden. *Auto-ML* [58] geht einen Schritt weiter und automatisiert einzelne Schritte des Entwicklungszyklus: die Suche nach einer geeigneten ML-Modellarchitektur und die Anpassung der entsprechenden Hyperparameter im Training. Konsequent wäre, künftig zunächst eine automatisierte Anpassung über mehrere Schritte eines Entwicklungszyklus und anschließend über mehrere Entwicklungszyklen hinweg anzustreben. Ein Beispiel wäre die autonome Entscheidung, wann ein ML-Modell, das bereits in Produktion ist, an veränderte Laufzeitbedingungen angepasst werden soll sowie die Auswahl und Durchführung der erforderlichen Schritte. Konzeptionell ähnliche Ansätze gibt es bereits im Bereich der *kollektiven autonomen Systeme* [157], die Wert darauf legen, formal zu verifizieren, dass eine (selbstständige) Anpassung von Systemkomponenten in der gegebenen Situation valide ist.

Im Rahmen einer automatisierten Anpassung über mehrere Entwicklungsschritte oder sogar -zyklen werden, verglichen mit der Frequenz, mit der ein ML-System im Rahmen seines Einsatzzwecks Vorhersagen trifft, wenige Ent-

scheidungen getroffen. Diese haben aber eine große Tragweite, denn eine Anpassung beinhaltet mit hoher Wahrscheinlichkeit einen oder mehrere Trainings-Test-Zyklen, die einen signifikanten Zeit- und Ressourcenbedarf haben können. Die Vorhersagequalität von ML-Modellen skaliert mit der Anzahl ihrer Parameter, und eine größere Anzahl an Parametern erfordert sowohl mehr Trainingsdaten, als auch mehr Rechenleistung zur Optimierung dieser Parameter (siehe beispielsweise [62] im Kontext großer Sprachmodelle). Die Komplexität von ML-Modellen und der für das Training dieser Modelle erforderliche Rechenaufwand sind in den letzten Jahren exponentiell gestiegen und das Training von Modellen, die dem neuesten Stand der Technik entsprechen, wurde erst durch die jüngsten Fortschritte in der Hardwareentwicklung und im algorithmischen Design möglich [60]. (Automatisierte) Anpassungen sollten daher zielgerichtet erfolgen, und es sollte Grund zur Annahme geben, dass eine den investierten Ressourcen angemessene Verbesserung möglich ist. Insbesondere sollten von vornherein die Entwicklung qualitativ hochwertiger ML-Modelle angestrebt werden, anstatt vermeidbare Fehler im Nachhinein auszubessern.

Ebenso ist zu beachten, dass es im Kontext der *Erklärbarkeit* (englisch *explainability* oder *X-AI*) derzeit an Methoden und Techniken mangelt, die es zumindest menschlichen Experten ermöglichen, die Entscheidungen der ML-Komponenten zu verstehen [84]. Es ist zwar möglich, post hoc Erkenntnisse über die Vorhersagen zu gewinnen [119], und es wurde wiederholt vorgeschlagen, Techniken zu entwickeln, die per Design erklärbar sind [127], aber die meisten aktuellen ML-Techniken erfüllen diese Anforderung (noch) nicht und auch ML-Ingenieure können nicht immer erklären, warum die verwendeten Modelle zu einem bestimmten Ergebnis kommen. Dabei ist zu beachten, dass zwar jeder Parameter innerhalb eines ML-Modells mathematisch analysiert werden kann, dies allein aber noch keine kausalen Zusammenhänge erklärt. Dies kann in Bereichen wie dem Gesundheitswesen oder der Strafjustiz zu rechtlichen Problemen führen, weshalb dort von einem Einsatz ML-basierter Verfahren abgesehen werden sollte. In den übrigen Bereichen muss es aber möglich sein, zumindest die Ergebnisse als solche nachvollziehen zu können, so lange das für die Regeln, nach denen ML-basierte Systeme zu diesen Ergebnissen kommen, noch nicht der Fall ist. Es muss also zweifelsfrei festgestellt werden können, dass die Ergebnisse korrekt im Sinne der Zielsetzung sind, die üblicherweise wie im vorherigen Kapitel 4 durch eine Spezifikation gegeben ist.

Hinsichtlich Effizienz und Erklärbarkeit strebt diese Arbeit eine direkte Integration von Methoden der *Verifikation und Validierung* (V&V) an, die per Definition [67] die Qualität eines Produktes über den gesamten Lebenszyklus gewährleisten sollen. Die Verifikation soll sicherstellen, dass das System korrekt entwickelt wurde in dem Sinne, dass die Ergebnisse den Anforderungen entsprechen, die zuvor spezifiziert wurden [20]. Die Validierung soll sicherstellen, dass das korrekte System entwickelt wird. Das heißt, dass das System seinen Verwendungszweck erfüllt, beispielsweise indem es einen Anwender effektiv unterstützt [20].

Um die Entwicklung ML-basierter Software über mehrere Entwicklungszyklen mit V&V-Methoden zu steuern, schlägt diese Arbeit ein formales Prozessmodell mit einer grafischen Syntax vor, das sowohl verschiedene ML-Techniken abbilden, als auch auf bestehende Entwicklungsprozesse abgebildet werden kann. Das Prozessmodell soll einerseits systematisch durch den ML-Lebenszyklus führen. Dabei soll die Mehrheit der in der Literatur beschriebenen, ML-spezifischen Aktivitäten und Artefakte berücksichtigt werden und es soll eine individuelle Definition von Feedbackschleifen möglich sein, die mit Aktivitäten und resultierenden Artefakten der Qualitätssicherung gesteuert werden können. Andererseits soll das Prozessmodell eine Meta-Optimierung über mehrere Entwicklungsschritte ermöglichen, die bei konsequenter Automatisierung eine Realisierung von selbstanpassenden, autonomen Systemen erlaubt, die auf ML-Methoden basieren.

5.3 Verwandte Arbeiten

Nachfolgend werden diese Ziele in den Kontext existierender Vorgehensweisen zur *Entwicklung selbstanpassender Systeme* (Unterkapitel 5.3.1) und *Entwicklung ML-basierter Systeme* (Unterkapitel 5.3.2) gesetzt.

5.3.1 Entwicklung selbstanpassender Systeme

Das *Autonomous Service Component Ensembles* (ASCENS) Projekt [157] beschäftigt sich mit der Entwicklung von *Ensembles*, unter denen die Autoren autonome Einheiten verstehen, die sich zur Laufzeit an Veränderungen in der Umgebung anpassen sowie neues Wissen in ihr Verhalten einfließen lassen. Das Projekt betrachtet sowohl traditionelle Aktivitäten der Softwareentwicklung, als auch Mechanismen der Selbstanpassung, Selbstüberwachung und Selbsterkenntnis der autonomen Einheiten. Ein zentraler Gedanke ist, dass das korrekte Verhalten der Ensembles durch geeignete Methoden und Werkzeuge sichergestellt werden sollte, wobei auch Einschränkungen der Umgebung beachtet werden müssen. Die Interaktion von menschlichen Entwicklern und autonomer Anpassung wird durch das *Ensemble Development Life Cyle* [64] Modell beschrieben, das den Softwarelebenszyklus abdeckt und Mechanismen skizziert, die Systemänderungen zur Laufzeit ermöglichen. Die Modellierungs- und die Programmierphase können durch Aktivitäten zur Konstruktion eines digitalen Zwillings erweitert werden. Der daraus resultierende *AIDL* Lebenszyklus [156] ermöglicht, Strategien (im Sinne der Definition aus Unterkapitel 2.1) für autonome Systeme unter Verwendung von SD-Algorithmen wie RL zu entwickeln und einzusetzen. Bedingt durch den hohen Abstraktionsgrad ist das Zusammenwirken der einzelnen Kontroll- und Feedbackschleifen lose definiert, aber flexibel hinsichtlich der (technischen) Ausgestaltung. Der Fokus der vorliegen-

den Arbeit liegt auf einer präzisen Darstellung des Zusammenspiels von iterativen Aktivitäten und Artefakten bei der Entwicklung ML-basierter Verfahren im Kontext einer Selbstanpassung über mehrere Entwicklungszyklen.

AIDL legt, wie das gesamte ASCENS-Projekt, großen Wert auf formale Verifikation, bei der korrektes Verhalten insbesondere für die angepassten Komponenten des Systems nachgewiesen werden soll. Mit Blick auf die Herausforderung der Erklärbarkeit beim Einsatz von ML-basierten Systemen sowie emergentem Verhalten in MAS skaliert die formale Verifikation möglicherweise nicht gut genug, um alle Eigenschaften zu überprüfen. Unbestreitbar ist, dass formale und empirische Tests ein zentraler Punkt sind, um die automatisierte Anpassung zu kontrollieren. Dafür könnten auch Techniken wie *Scenario Co-Evolution* [42] verwendet werden, die die Auswirkungen der laufenden Anpassung von ML-basierten Systemen im Training auf das SE beschreiben, wobei der Fokus auf der Qualitätssicherung durch Testen liegt. Dafür wird zunächst ein ML-Entwicklungsprozess formuliert und anschließend in einen formalen Rahmen zur Analyse der Anpassung eingebettet. Das zentrale Ergebnis ist: Das Testen muss sich anpassen, beispielsweise hinsichtlich der Schwierigkeit, um mit den Fähigkeiten des zu testenden Systems Schritt zu halten. Eine Beispiel dafür ist das *Antagonist-Ratio Training Scheme* [109], dass ein MAS als allgemeines Szenario mit einer Gruppe von Protagonisten, die reguläre Agenten des MAS darstellen, und einer Gruppe von Antagonisten, die Fehler im MAS darstellen, formuliert. Die Protagonisten müssen lernen, ihre Aufgabe in Gegenwart von ebenfalls lernenden Antagonisten zu erfüllen, deren Ziel es ist, die Protagonisten zu stören. Aus Sicht der Protagonisten wird die Schwierigkeit des Szenarios durch die Anzahl und den Trainingsstand der Antagonisten bestimmt. Solche adaptiven Tests sind in den Bereichen wichtig, in denen ML-Systeme Fähigkeiten entwickeln können, die denen menschlicher Experten überlegen sind. A priori definierte Tests sind dann möglicherweise nicht mehr in der Lage, die Fähigkeiten des Systems vollständig zu prüfen. Ungeachtet der exakten technischen Ausprägung berücksichtigt die vorliegende Arbeit eine Integration von V&V-Aktivitäten und entsprechender Feedback-Schleifen zu jedem Zeitpunkt des Lebenszyklus.

5.3.2 Entwicklung ML-basierter Systeme

Die Literatur wurde hinsichtlich der Entwicklung von ML-basierten Systemen im Kontext der jüngsten Fortschritte des DL mehrfach systematisch zusammengefasst [48, 93]. Eine gemeinsame Schlussfolgerung ist, dass dabei insbesondere die nicht-deterministische Natur von ML-Techniken beachtet werden muss, also die Auswirkungen einer abweichenden Datenverteilung oder einer veränderten ML-Modell-Architektur schlecht vorhersehbar sind, und es an ausgereiften Werkzeugen und Techniken fehlt, um damit umzugehen. Den Autoren zufolge lassen sich die Herausforderungen nur schwer anhand der etablier-

ten *SWEBOK*-Wissensbereiche [20] klassifizieren, da sie stark an das jeweilige Problem gebunden sind. Die vorliegende Arbeit führt unabhängig von der Problemdomäne systematisch durch den ML-Lebenszyklus und strebt eine Integration von V&V-Aktivitäten an, um Qualität sicherzustellen. Mangelhafte Datenqualität oder fehlende V&V-Methoden können damit nicht ersetzt werden, eine konsistente Qualität durch ein a priori definiertes und post hoc überprüfbares Vorgehen aber schon.

Um die größten Herausforderungen bei der Entwicklung von ML-Komponenten zu identifizieren, werteten Lwakatare et al. [92] eine Reihe von empirischen Fallstudien aus. Sie fassen in einer Taxonomie auf hohem Abstraktionsgrad die Verwendung von ML-Komponenten in industriellen Umgebungen zusammen und ordnen die Herausforderungen einer von vier Phasen zu: erstens den Datensatz zusammenzustellen, zweitens das Modell zu erstellen, drittens das Modell zu trainieren und evaluieren, und viertens das Modell einzusetzen. Das Erstellen des Modells wird im ML-Entwicklungszyklus in Abbildung 2.4 nicht explizit berücksichtigt, findet aber ebenso (zwangsläufig) statt. Eine entsprechende Aktivität wird im Prozessmodell der vorliegenden Arbeit berücksichtigt. Aufbauend auf dieser Taxonomie schlagen Bosch et al. eine Forschungsagenda [19] vor, die als letzten Schritt autonome ML-Komponenten nennt. Darunter verstehen sie die Selbstanpassung, auf die das Prozessmodell der vorliegenden Arbeit mit einer Meta-Optimierung über mehrere Entwicklungszyklen abzielt. Der Fokus der vorliegenden Arbeit liegt nicht auf dem Management der Daten, aber abseits davon werden die Aktivitäten der Forschungsagenda von Bosch et al. berücksichtigt. Eine weitere Literaturübersicht über Vorgehensweisen und Praktiken in der ML-Entwicklung findet sich bei Watanabe et al. [153]. Sie geben bei den häufig beschriebenen Praktiken explizit an, ob sie aus der klassischen Softwareentwicklung übernommenen wurden. Hinsichtlich dem Sammeln und Aufbereiten der Daten sind Watanabe et al. granularer als die vorliegende Arbeit. Die übrigen Phasen und Aktivitäten werden für die später folgende Ausgestaltung des Prozessmodells zur Kenntnis genommen.

Mit dem Aufkommen von MLOps [75] entwickelten Lwakatare et al. eine hinsichtlich der Aktivitäten und Feedbackschleifen präzise Darstellung des damit verbundenen Vorgehens [91]. Abbildung 5.1 stellt dar, wie Lwakatare et al. den ML-Entwicklungszyklus und DevOps [81] kombinieren, indem sie die Phasen *Data Management* und *ML Modeling* samt der zugehörigen Aktivitäten mit den Phasen *Development* und *System Operation* verknüpfen. Erwähnenswert ist insbesondere die explizit dargestellte Möglichkeit, ein ML-Modell direkt in den Betrieb zu übergeben, ohne das restliche System, in welches das ML-Modell eingebettet ist, zu ändern (diese Phase wird dabei übersprungen). Diese Modularität ermöglicht eine schnelle Anpassung des ML-Modells. Insgesamt wird der ML-Entwicklungszyklus so konsistent in das etablierte DevOps-Prozessmodell integriert.

Eine alternative Darstellung der ML-Entwicklung, die sich stärker auf die ML-

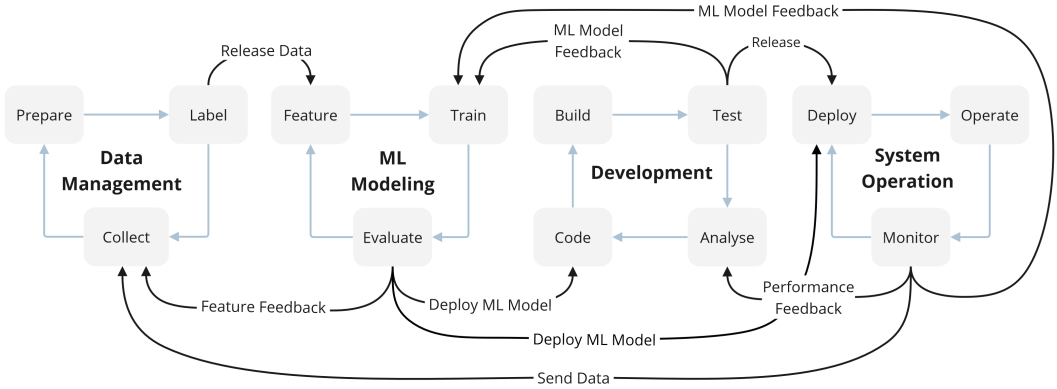


Abbildung 5.1: Phasen, Aktivitäten und Feedbackschleifen von MLOps, einer Kombination aus dem ML-Entwicklungszyklus (Zyklen links und mittig-links) und DevOps (Zyklen mittig-rechts und rechts). Darstellung aus [123] und basierend auf [91].

Komponenten fokussiert, bietet CRISP-ML(Q) [141]. Verglichen mit den vier Phasen von Lwakatare et al. [92] kommt in CRISP-ML(Q) eine Phase hinzu, um Geschäftsprozesse und Anwendungsfälle zu analysieren, was gemäß der Autoren eng mit dem Sammeln und Aufbereiten der Daten verbunden ist. Das Trainieren und Evaluieren sowie der Betrieb des ML-Modells von Lwakatare et al. [92] werden in CRISP-ML(Q) aufgeteilt in die Qualitätssicherung für ML-Anwendungen, die Übergabe in den Betrieb, die Überwachung und die Wartung. CRISP-ML(Q) schlägt vor, in jeder Phase initial Herausforderungen in Form von Risiken zu identifizieren und Gegenmaßnahmen zu treffen. Besonderes Augenmerk legen die Autoren auf die letzte Phase, in der sie das Modell in einer dynamischen Echtzeitumgebung wähnen und eine präzise Überwachung und Wartung vorschlagen, um einen potenziellen Leistungsabfall zu erkennen und zu verhindern. CRISP-ML(Q) bietet eine detaillierte Liste an Aktivitäten für jede Phase, stellt aber nicht dar, wie diese voneinander abhängen oder welche Artefakte beteiligt sind. Insgesamt fehlt MLOps und CRISP-ML(Q) eine präzise Sicht auf die Abhängigkeiten zwischen Aktivitäten und Artefakten. Die vorliegende Arbeit versucht, dies im Prozessmodell zu realisieren.

5.4 Prozessmodell für selbstanpassende, ML-basierte Systeme

In diesem Kapitel wird das Prozessmodell beschrieben. In der Softwareentwicklung gibt es unterschiedliche Vorgehensweisen mit spezifischen Notationen und die Meinung, welche Variante am besten geeignet ist, variiert je nach Personenkreis und Ära. Im Rahmen der vorliegenden Arbeit wird das nicht thematisiert und das Prozessmodell stützt sich in dieser Hinsicht nicht mehr

als erforderlich auf (noch) aktuelle Trends. Hinsichtlich der Darstellung der ML-Entwicklung wird hingegen eine möglichst präzise Darstellung der Besonderheiten angestrebt. Unterkapitel 5.4.1 beginnt mit der Schilderung der grundlegenden Konzepte, gefolgt von einer formalen Definition des Prozessmodells in Unterkapitel 5.4.2. Anschließend stellt Unterkapitel 5.4.3 eine grafische Darstellung des Prozessmodell anhand einer Blaupause vor, die den gesamten ML-Lebenszyklus umfasst. Danach beschreibt Unterkapitel 5.4.4 die einzelnen Aktivitäten der Blaupause. Das abschließenden Unterkapitel 5.4.5 bildet als Anwendungsbeispiel den Ansatz SAT-MARL des vorherigen Kapitels 4 mit der grafischen Darstellung ab.

5.4.1 Prinzipien

In diesem Unterkapitel werden sechs grundlegende Konzepte beschrieben, denen das Prozessmodell folgt. Erstens teilt es den Prozess in verschiedene Phasen. Jede Phase hat einen anderen Schwerpunkt, der sich in entsprechenden Aktivitäten widerspiegelt. Im Sinne agiler Ansätze müssen diese Phasen nicht strikt getrennt sein oder zwangsläufig in sequenzieller Reihenfolge ausgeführt werden und dürfen sich überschneiden. Dennoch sind *Planung* (original: Planning), *Entwicklung* (original: Development), *Auslieferung* (original: Deployment) und *Betrieb* (original: Operations) nützliche Kategorien beziehungsweise Phasen, um die Aktivitäten zu gruppieren.

Zweitens bildet das Prozessmodell sowohl menschliche Tätigkeiten, als auch maschinelle Routinen durch *Aktivitäten* (original: Activities) ab. Aktivitäten werden von unterschiedlichen Schlüsselfaktoren vorangetrieben oder limitiert, stützen sich auf unterschiedliche technische oder menschliche Ressourcen und erfordern unterschiedliche Fähigkeiten. Beispielsweise sollte die *Anwendungsfallanalyse* (original: Use Case Analysis) von Business-Analysten durchgeführt werden und in Anforderungen in Form einer Spezifikation resultieren. Mit dieser Spezifikation können Datenwissenschaftler dann geeignete Daten für das Training des ML-Modells auswählen oder sammeln. Business-Analysten und Datenwissenschaftler sollten dabei eng zusammenarbeiten.

Drittens modelliert das Prozessmodell Aktivitäten so, dass sie jeweils mindestens ein *Artefakt* (original: artifact) erzeugen. Ebenso können Aktivitäten Artefakte *voraussetzen*. Im obigen Beispiel hängt die Auswahl der Daten durch Datenwissenschaftler von der Problemdomäne ab, die durch die Spezifikation der Anforderungen beschrieben wird. Folglich muss die Anwendungsfallanalyse, die in einer Spezifikation der Anforderungen resultiert, zuerst abgeschlossen werden. Artefakte haben verschiedene Typen: *funktionale Beschreibung* (original: Functional Description), *Datensatz* (original: Data) oder *logische Aussage* (original: Logical Statement). Funktionale Beschreibungen sind die charakterisierenden Eigenschaften des ML-Modells, beispielsweise die Gewichte des zu Grunde liegenden neuronalen Netzes. Logische Aussagen sind beispielsweise

eine Spezifikation, ein Satz von Hyperparametern oder auch ein Testergebnis, das besagt, ob ein ML-Modell mit bestimmten Hyperparametern auf bestimmten Daten eine bestimmte Anforderung erfüllt.

Viertens setzt das Prozessmodell Aktivitäten und Artefakte über Assoziationen miteinander in Verbindung. Wie zuvor geschildert, kann eine Aktivität ein oder mehrere Artefakte voraussetzen sowie ein oder mehrere Artefakte erzeugen.

Fünftens bildet das Prozessmodell die Zeit über Assoziationen und Zustände ab. Zu einem Zeitpunkt ist eine Aktivität entweder abgeschlossen (*done*), oder sie wurde bereits gestartet und ist noch nicht abgeschlossen (*active*), oder sie wurde noch nicht gestartet (*inactive*). Ebenso verhält es sich mit Artefakten: entweder sie wurden noch nicht erzeugt (*inactive*), oder sie wurden erzeugt, werden genutzt und dabei gegebenenfalls verändert (*active*), oder sie wurden erzeugt und werden nicht mehr verändert (*done*). Kombiniert mit den Assoziationen sind die Zustände der Schlüssel, um das Zusammenspiel aus Aktivitäten und Artefakten präzise darstellen zu können. Beides wird sowohl in der formalen Repräsentation in Unterkapitel 5.4.2, als auch in der grafischen Syntax in Unterkapitel 5.4.3 genauer erläutert.

Sechstens modelliert das Prozessmodell Feedback und (automatisierte) Optimierung durch Schleifen, die zu früheren Aktivitäten des ML-Entwicklungsprozesses zurückführen. Auf diese Weise bildet es die iterative Vorgehensweise beim Entwickeln von ML-Modellen ab, die aus wiederholter stochastischer Optimierung und empirischer Validierung des ML-Modells in verschiedenen Entwicklungsstadien besteht. Dies ermöglicht ebenso die Abbildung verschiedener Vorgehensweisen und Techniken, nicht nur im Bezug auf die ML-Entwicklung, sondern auch auf V&V. Beispielsweise kann dadurch sowohl eine manuell durchgeführte Optimierung der Hyperparameter beschrieben werden, als auch die vollautomatische Fehlerkorrektur eines ML-Modells, das mit Daten aus dem Betrieb erneut trainiert wurde.

5.4.2 Formalisierung

In diesem Unterkapitel wird das Prozessmodell formal definiert. Ein Entwicklungsprozess $\mathfrak{D}$ ist durch eine Menge von Elementen E gegeben, die entweder Aktivitäten oder Artefakte sind. In beiden Fällen weist ein Element $e \in E$ zwei Arten von Assoziationen auf: erstens zu seinen *Voraussetzungen* $pre(e)$, also Elementen, die es benötigt, und zweitens zu den Elementen $post(e)$, für die es selbst eine Voraussetzung ist. Die Elemente und Assoziationen, die für den Entwicklungsprozess $\mathfrak{D}$ in dieser Arbeit verwendet werden, sind in Abbildung 5.2 zu sehen.

Bei der Ausführung von $\mathfrak{D}$ nimmt jedes seiner Elemente zu jedem Zeitpunkt t einen Zustand $s^{(t)} \in \{inactive, active, done\} = S$ an. Dadurch besteht eine Instanz $\mathcal{D}$ des Entwicklungsprozesses $\mathfrak{D}$ aus Mengen von Zuständen S entlang

einer Zeitlinie. Eine Zeitlinie ist eine Folge von Zeitpunkten $t \in [t_{start}, t_{end}] \subseteq \mathbb{N}$, wobei der Prozess zu jedem Zeitpunkt t jedes Element auf einen der zuvor genannten Zustände gemäß $\mathcal{D}^{(t)} : E \rightarrow S$ abbildet. Zu Beginn, also zum Zeitpunkt t_{start}, werden alle Elemente e in den Zustand *inactive* versetzt, formal $D^{(t_{start})}(e) = \textit{inactive}$. Anschließend wird der Prozess gestartet, indem diejenigen Elemente e in den Zustand *active* versetzt werden, formal $\mathcal{D}^{(t+1)}(e) = \textit{active}$, die eine Voraussetzung für Elemente im unmittelbar folgenden Zeitpunkt sind, formal $\forall e' \in \textit{pre}(e) : \mathcal{D}^{(t)}(e') = \textit{active}$. Ist eine Aktivität im Zustand *active*, so ist damit gemeint, dass diese zum gegebenen Zeitpunkt von Entwicklern oder technischen Routinen durchgeführt wird. Sind Artefakte im Zustand *active*, so ist damit gemeint, dass diese zum gegebenen Zeitpunkt von (mindestens) einer Aktivität verwendet werden. Eine Besonderheit ist: Artefakte dürfen sich zusammen mit den Aktivitäten, die sie verwenden, verändern. Ein Element muss mindestens einen Zeitschritt lang im Zustand *active* gewesen sein, um in den Zustand *done* wechseln zu können. Aktivitäten im Zustand *done* werden als abgeschlossen betrachtet. Artefakte im Zustand *done* existieren weiterhin und können lesend verwendet werden, aber sie werden als unveränderlich betrachtet. Wenn ein Element eine Voraussetzung für ein anderes Element ist, muss es entweder im Zustand *active* oder im Zustand*done* sein, aber anders als in anderen Prozessmodellen muss es nicht zwingend im Zustand *done* sein. Dennoch kann ein Artefakt unmittelbar nach dem Start der Elemente, für die es eine Voraussetzung ist, in den Zustand *done* wechseln.

Eine wichtiges Konzept des Prozessmodells ist, dass es beliebige Feedbackschleifen in die Vergangenheit erlaubt. Es ist somit möglich, *synchron* von jedem beliebigen Zeitpunkt zu einem früheren Zeitpunkt zurückzuspringen und dadurch eine beliebige Anzahl von Elementen des Prozesses zu wiederholen. Formal kann ein Element e mit $\mathcal{D}^{(t)}(e) \in \{\textit{active}, \textit{done}\}$ in die Zustände $\{\textit{active}, \textit{inactive}\}$ wechseln, solange alle Elemente $e' \in \textit{post}(e)$ ebenfalls entsprechend in die Zustände $\{\textit{active}, \textit{inactive}\}$ wechseln. Jedoch kann ein Element e zu keinem Zeitpunkt im Zustand *active* sein, ohne dass alle seine Voraussetzungen $\textit{pre}(e)$ entweder im Zustand *active* oder im Zustand *done* sind. Diese Art der Rückkopplungsschleifen ermöglicht, dynamische Entwicklungsprozesse unter Aufrechterhaltung einer Hierarchie zu definieren.

Dies erlaubt eine Prüfung jeder Instanz $\mathcal{D}$, ob sie dem Prozess $\mathfrak{D}$ folgt, formal $\mathfrak{D} \models \mathcal{D}$. Darüber hinaus wäre es denkbar, mit Hilfe temporaler Logiken wie CTL* [38] bei der Planung von (automatisierten) Feedbackschleifen Aussagen über den Prozessablauf zu formulieren und später während der Ausführung zu überprüfen. Ebenso können mit temporalen Logiken grundlegende Eigenschaften formuliert werden, beispielsweise, dass ein Prozess das gewünschte ML-Modell auf allen Pfaden künftig (irgendwann) hervorbringt, über den *All Finally* (AF) Operator: $AF\,\mathcal{D}(\textit{Trained ML Model}) = \textit{done}$. Trotz all der Möglichkeiten, die sich hier bieten, sind auch Elemente e mit $\textit{post}(e) = \emptyset$, beispielsweise Testergebnisse, wichtige Ergebnisse einer Instanz des Entwicklungsprozesses und sollten entsprechend berücksichtigt werden, was im nachfolgenden

Unterkapitel zur Geltung kommt.

5.4.3 Grafische Darstellung

Dieses Unterkapitel beinhaltet eine grafische Darstellung des Prozessmodells und beschreibt die verwendete Syntax und Semantik. Abbildung 5.2 enthält eine Blaupause für einen ML-Softwareentwicklungsprozess über den gesamten ML-Lebenszyklus. Die Elemente sind innerhalb einer zeitlichen Dimension (x-Achse) und einer organisatorischen Dimension (y-Achse) angeordnet. In der zeitlichen Dimension sind die Elemente je einer von vier Phasen zugeordnet. Dies ermöglicht eine unkomplizierte zeitliche Zuordnung zu und Synchronisation mit bestehenden Vorgehensweisen der Softwareentwicklung. Eine Position weiter rechts auf der x-Achse bedeutet gleichermaßen später im Entwicklungsprozess und weiter fortgeschritten hinsichtlich der Fertigstellung der ML-Software. In organisatorischer Dimension sind die Elemente je einer von sechs Kategorien zugeordnet, die eine Mischung aus Geschäftsbereichen und technischen Bereichen bilden. Damit sollen keine Rollen oder Verantwortlichkeiten definiert, sondern Schlüsselfaktoren skizziert werden, die den Prozess maßgeblich vorantreiben oder limitieren können. Manche Tätigkeiten oder Algorithmen lassen sich nicht (unbegrenzt) parallelisieren und auch Rechenleistung ist in der Praxis begrenzt. Wie bereits erwähnt ist die Kategorisierung der Elemente innerhalb beider Dimensionen nicht als strikt verbindlich anzusehen. Stattdessen sollten sie an die gegebenen Umstände angepasst werden.

Die grafische Darstellung bildet Aktivitäten, Artefakte und Assoziationen ab. Aktivitäten und Artefakte sind durch Rechtecke dargestellt. Die Farbe repräsentiert deren Typ: Artefakte sind blau, Datensätze grün, funktionale Beschreibungen gelb und logische Aussagen rot dargestellt. Assoziationen zwischen Artefakten und Aktivitäten sind durch Pfeile dargestellt. Wenn eine Aktivität ein Artefakt voraussetzt, verbindet ein Pfeil die beiden Elemente. Der Pfeil beginnt mit einem weißen Kreis im vorausgesetzten Artefakt und endet mit der Spitze an der betreffenden Aktivität. Wenn eine Aktivität ein Artefakt erzeugt, wird dies ebenfalls durch einen Pfeil gekennzeichnet. Dieser beginnt mit einem weißen Kreis in der betreffenden Aktivität und endet mit der Spitze im erzeugten Artefakt.

Gemäß der formalen Definition kann eine Instanz des Entwicklungsprozesses an einem beliebigen Zeitpunkt betrachtet werden, um Aussagen über die Instanz zu diesem Zeitpunkt zu treffen. In der grafischen Darstellung wird die Instanz dazu vertikal an der entsprechenden Stelle auf der x-Achse geschnitten. Werden dabei Aktivitäten oder Artefakte getroffen, sind diese aus formaler Sicht im Zustand *active*. Aktivitäten oder Artefakte, die links vom Schnittpunkt enden, sind formal im Zustand *done*. Analog dazu sind Aktivitäten oder Artefakte, die rechts des Schnittpunkts beginnen, formal im Zustand *inactive*. Die grafische Darstellung kann aber auch als Ablaufdiagramm interpretiert werden, das den

Prozess über die Reihenfolge der Aktivitäten beschreibt. Aktivitäten sind nicht direkt mit anderen Aktivitäten verbunden, sondern indirekt über Artefakte. Aus Sicht einer Aktivität A_x sind alle Aktivitäten, deren erzeugte Artefakte A_x voraussetzt, abgeschlossen und formal im Zustand *done*. Umgekehrt sind Aktivitäten, die das oder die von A_x erzeugten Artefakte voraussetzen, noch nicht gestartet und formal im Zustand *inactive*. Eine Aktivität wird automatisch gestartet, formal im Zustand *active*, wenn alle vorausgesetzten Artefakte erzeugt worden sind, formal in den Zuständen *active* oder *done*. Wie in der Formalisierung erläutert, ist es denkbar, dass sich Artefakte parallel zu den Aktivitäten, die sie verwenden, weiterentwickeln. Damit Aktivitäten starten können, müssen die vorausgesetzten Artefakte daher nicht zwingend im Zustand *done* sein, sondern können auch im Zustand *active* sein, was sich darin äußert, dass sie zum entsprechenden Zeitpunkt vertikal geschnitten werden würden. Jedoch ist nicht vorgesehen, dass eine Aktivität schon vor dem Wechsel in den Zustand *done* ein oder mehrere Artefakte erzeugt, die parallel von einer oder mehreren Aktivitäten genutzt werden.

Ursprünglich sollte die grafische Darstellung so konstruiert werden, dass bei einem vertikalen Schnitt der x-Achse immer (mindestens) eine Aktivität getroffen wird. Jedoch sollte die Darstellung ebenfalls leicht zu lesen und optisch ansprechend sein, was in Abständen zwischen allen Elementen auf der x-Achse resultierte. Folglich gibt es Bereiche auf der x-Achse, in denen ein vertikaler Schnitt keine Aktivität, aber zumindest Assoziationen von beziehungsweise zu Aktivitäten trifft, denen entsprechend gefolgt werden soll. Darüber hinaus sind die Assoziationen nicht multiplikativ. Syntaktisch gesehen spielt es keine Rolle, ob mehrere Artefakte durch einen gemeinsamen oder mehrere individuelle Pfeile mit einer Aktivität verbunden sind. Am wichtigsten ist jedoch, dass die Größe der Elemente – insbesondere ihre Länge auf der x-Achse – durch die Länge der Beschriftung und das Layout bestimmt wird und nichts über ihre relative oder absolute Dauer aussagt.

In der untersten Kategorie *Feedback* auf der y-Achse gibt es einige gestrichelte, annotierte Pfeile. Sie beginnen, sobald bestimmte V&V-Artefakte existieren, und führen zu früheren Zeitpunkten zurück. Dies sind Beispiele für verschiedene Feedback- respektive Optimierungsschleifen. Ein monolithisches Vorgehen kann auf diese Schleifen verzichten, ein iteratives Vorgehen ist zwingend darauf angewiesen. Ebenso sind gemischte Ansätze denkbar, die nur einige ausgewählte Schleifen nutzen. In der Praxis könnten *Qualitätsprüfungspunkte* (englisch: quality gates) auf Grundlage der V&V-Artefakte entscheiden, wie der Prozess zu bestimmten Zeitpunkten fortgesetzt wird. Die exakte Ausgestaltung hängt von organisatorischen und rechtlichen Anforderungen sowie den technischen Gegebenheiten ab.

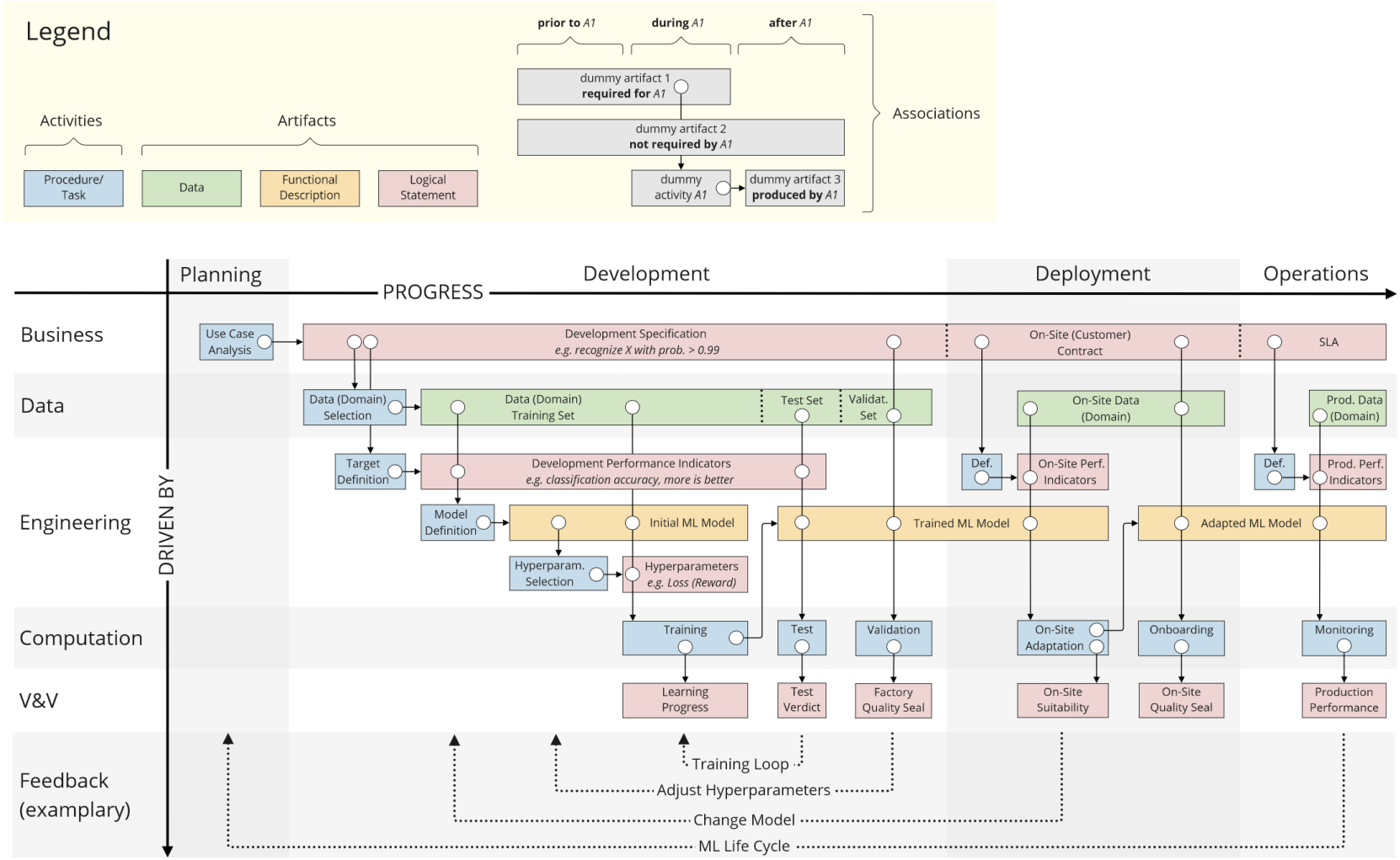


Abbildung 5.2: Visualisierung des Prozessmodells für die Entwicklung von ML-Software. Darstellung entnommen aus [123].

5.4.4 Beschreibung der Aktivitäten

Dieses Unterkapitel erläutert die Aktivitäten der Blaupause aus Abbildung 5.2 zeitlich aufsteigend sortiert und in Phasen gruppiert. Das Prozessmodell basiert auf Aktivitäten, und die grafische Darstellung kann ähnlich einem Aktivitätsdiagramm verwendet werden, das einen Prozess über die Reihenfolge seiner Aktivitäten beschreibt. Die Blaupause kann das Vorgehen verschiedener ML-Varianten abbilden, darunter nicht nur das in dieser Dissertation verwendete RL, sondern beispielsweise auch SL. Wie in Unterkapitel 2.5 dargelegt ist der grundlegende Ablauf der Entwicklung von DL-Modellen ungeachtet der Variante gleich und es gibt Ansätze, die Agenten [149] und LLMs [2, 147] zuerst mit SL und dann mit RL trainieren. Daher werden im Folgenden die Begriffe der zu Grunde liegenden Vorveröffentlichung [122] übernommen und die Beschreibungen der Aktivitäten nehmen sowohl auf SL, als auch auf RL Bezug. Ein praktisches Anwendungsbeispiel, das ausschließlich Begriffe des RL verwendet und konkrete Feedbackschleifen definiert, folgt in Unterkapitel 5.4.5.

Planung Die Planungsphase startet mit der *Use Case Analysis*, in der Business-Analysten in den Geschäftsbereichen Anwendungsfälle für ML-Software identifizieren und ausarbeiten. Die daraus resultierende *Entwicklungsspezifikation* (original: Development Specification) definiert die Ziele in natürlicher Sprache. Diese Spezifikation kann sich im Laufe des ML-Lebenszyklus noch ändern, beispielsweise auf Basis der ersten Erkenntnisse des Trainings, oder wenn das trainierte ML-Modell erstmalig in der späteren Laufzeitumgebung eingesetzt wird. In jedem Fall ist es wichtig, die stochastische Natur moderner DL-Verfahren bei der Formulierung der Spezifikation zu berücksichtigen [48, 93]. Beispielsweise sollte für die Erfüllung von Zielen eine (obere oder untere) Schranke, die erlaubte Streuung sowie die mindestens erforderliche Konfidenz beim Schätzen der Leistung des ML-Modells auf Basis empirischer Tests formuliert werden.

Entwicklung Die erste Aktivität der Entwicklungsphase ist die *Data Selection*, in der basierend auf der Entwicklungsspezifikation ein Datensatz ausgewählt oder zusammengestellt wird. Es ist üblich, diesen in drei Teile zu trennen: Training, Test und Validierung. Wenn nur wenige Daten verfügbar sind, werden diese Teile mitunter laufend neu zusammengestellt. In der grafischen Darstellung ist das durch gestrichelte Linien gekennzeichnet. Limitierender Faktor und treibende Kraft ist das *Data Management*. Das Prozessmodell definiert hier keine detaillierten Aktivitäten, sondern nimmt vereinfachend an, dass Daten in geeigneter Qualität und Quantität zugänglich sind. Das Data Management ist von großer Bedeutung, aber der Fokus des Prozessmodells liegt auf dem Lebenszyklus des ML-Modells. Kommt RL zum Einsatz, werden die Daten überwiegend ad hoc durch die Interaktion des oder der Agenten mit der Umgebung generiert. Dann ist an dieser Stelle eine Simulation oder abgesicherte Umgebung für den Agenten zu wählen und zu spezifizieren, wie der Agent diese Umgebung wahrnimmt und mit ihr interagiert.

Parallel werden im Rahmen der *Target Definition* die Trainingsziele für das ML-Modell definiert. Die resultierenden Entwicklungsmetriken spiegeln die Entwicklungsspezifikation in einer technischen Form wider und dienen dazu, verschiedene ML-Modelle oder verschiedene Versionen eines ML-Modells miteinander vergleichen und den Trainingsfortschritt messen zu können. Eine Anforderung kann beispielsweise sein, bestimmte Entitäten auf Bilddaten mit einer Wahrscheinlichkeit von mehr als 99% korrekt zu erkennen. Eine geeignete Entwicklungsmetrik dafür könnte die *Sensitivität* (englisch: recall) der Vorhersage des ML-Modells sein, wobei höhere Werte besser sind und das Ziel die Maximierung ist. Kommt RL zum Einsatz, könnte eine Anforderung sein, dass ein Agent eine Aufgabe in unter 50 Zeitschritten abschließen soll. Hier ist es intuitiv naheliegend, die Anzahl Zeitschritte zu messen, die der Agent benötigt, aber es gilt zu bedenken, dass ein Agent zu Beginn des Trainings die Aufgabe möglicherweise nicht in angemessener Zeit lösen kann und eine Messung des Fortschritts dann nicht möglich ist. Eine geeignete Metrik könnte stattdessen die Anzahl der abgeschlossenen Teilaufgaben sein, wobei auch hier höhere Werte besser sind und ein Ziel die Maximierung ist. Es ist aus zwei Gründen wichtig, klar zwischen der Entwicklungsspezifikation und dem Trainingsziel (mit den entsprechenden Metriken) zu trennen. Der erste Grund ist die Vermeidung von Missverständnissen zwischen Business-Analysten und ML-Experten. Der zweite Grund ist, dass so ein mehrstufiges Vorgehen möglich wird: Analog zu Kapitel 4 kann das Training mit einer Teilmenge der Anforderungen der Spezifikation beginnen und sobald dieses Trainingsziel erreicht ist, werden weitere Anforderungen ergänzt. Dieses Vorgehen wird so lange wiederholt, bis alle Anforderungen der Spezifikation enthalten sind und (hoffentlich) erfüllt werden.

In der folgenden *Model Definition* wird die Architektur des ML-Modells unter Berücksichtigung des Datensatzes und der Entwicklungsmetriken ausgewählt. Im Anschluss folgt die *Hyperparameter Definition*, in der die grundlegende Konfiguration des ML-Modells unter Berücksichtigung seiner Architektur und des Trainings vorgenommen wird. Die treibende Kraft hinter beiden Aktivitäten sind ML-Experten und Softwareingenieure. Die Hyperparameter können algorithmusspezifisch sein und umfassen auch das mathematische Optimierungsziel. Das ist beispielsweise der *Loss* im Fall von SL und der *Reward* im Fall von RL. Im RL wäre es technisch zwar auch möglich, den Loss zu messen. Jedoch ändern sich die gesammelten Daten mit jeder Anpassung der Gewichte des neuronalen Netzes, das die Strategie oder die Wertfunktion des Agenten repräsentiert, was im Training häufig der Fall ist. Daher ist in der Praxis selten eine eindeutige Tendenz im Verlauf des Loss zu erkennen und entsprechend selten kann damit eine verlässliche Aussage über das Verhalten des Agenten getroffen werden. Um die Übersichtlichkeit zu wahren, berücksichtigt das Prozessmodell keine dedizierte Aktivität zur Wahl des Algorithmus. Diese geht Hand in Hand mit der Wahl der Hyperparameter und kann ohne Verlust an Präzision im Rahmen dieser Aktivität geschehen. Im Übrigen ist es nicht un-

üblich, während des Trainings eines ML-Modells nicht nur die Hyperparameter anzupassen, sondern auch nacheinander unterschiedliche Algorithmen zu verwenden: beispielsweise wurde GPT-4 [2] initial mit *Self-Supervised Learning* und anschließend mit *Reinforcement Learning from Human Feedback* (RLHF) trainiert. Der Loss (im Fall von SL) beziehungsweise Reward (im Fall von RL) können als Teil der Hyperparameter aufgefasst werden, da sie das mathematische Optimierungsziel repräsentieren. Hier wird eine weitere Unterscheidung getroffen: Das mathematische Optimierungsziel wird von dem bereits beschriebenen Ziel der Entwicklungsspezifikation und dem Ziel der technischen Ausbildung unterschieden. Ein unmittelbarer Vorteil der Verwendung unterschiedlicher Metriken für die mathematische Optimierung und das Erreichen des Trainingsziels ist die Erkennung von *Reward-Hacking* [7, 79, 80], einer Herausforderung von RL im Bereich der Ausrichtung, die in Unterkapitel 2.5 beschrieben wurde.

Anschließend findet das *Training* des ML-Modells statt. Treibende Kraft und limitierender Faktor ist die verfügbare Rechenleistung. Der Trainingsfortschritt kann anhand des mathematischen Optimierungsziels beurteilt werden. Bei SL ist das beispielsweise der Loss auf den Trainingsdaten, bei RL der Reward in den Trainingsszenarien. Optional kann auch das Trainingsziel herangezogen werden, um den Lernfortschritt während des Trainings zu messen, denn bei korrekter Problemmodellierung führt eine Optimierung des mathematischen Ziels auch zu einer Optimierung des Trainingsziels. Unabhängig davon wird das Trainingsziel bei der nächsten Aktivität verwendet: dem *Test* des trainierten ML-Modells mit den Entwicklungsmetriken auf dem Testdatensatz (im Fall von SL) beziehungsweise den Testszenarien (im Fall von RL). Wie mehrfach erwähnt ist das Training von ML-Modellen ein iterativer, teils explorativer Prozess, bei dem verschiedene Hyperparameter und Algorithmen, seltener auch Architekturen, getestet werden. Entsprechend sind hier Feedbackschleifen von den Testergebnissen zu den entsprechenden Aktivitäten zu erwarten, die in der Praxis durch Auto-ML [58] automatisiert werden können. Die letzte Aktivität der Entwicklungsphase ist die *Validation*, in der geprüft wird, ob das trainierte ML-Modell die Entwicklungsspezifikation auf dem Validierungsdatensatz (im Fall von SL) respektive den entsprechenden Szenarien (im Fall von RL) erfüllt. Das resultierende Artefakt kann als eine Art Qualitätssiegel aufgefasst werden, das definiert, ob die Entwicklungsphase erfolgreich abgeschlossen ist. Hinsichtlich der Feedbackschleifen, die von Test und Validierung ausgehen, sollten die Experten der Qualitätssicherung eng mit den ML-Experten und Softwareingenieuren zusammenarbeiten, zu deren Aktivitäten die Feedbackschleifen zurückführen.

Auslieferung Nach erfolgreicher Validierung erfolgt der Wechsel in die Auslieferungsphase. Das trainierte ML-Modell wird nun mit weniger kontrollierten Bedingungen konfrontiert, beispielsweise durch eine Integration in das restliche (Software-) System, oder durch eine Integration in ein physisches System, wenn vorher eine Simulation verwendet wurde. Daher kann sich die Spezifi-

kation nun von der während der Entwicklung verwendeten unterscheiden und wird fortan als *Kundenspezifikation* (original: On-Site Contract) bezeichnet. Es ist möglich, dass die Kundenspezifikation eine Konkretisierung der Entwicklungsspezifikation ist, aber es können auch zusätzliche, kundenspezifische Anforderungen enthalten sein. Da die zusätzlichen Anforderungen und die Daten (im Fall von SL) beziehungsweise Einsatzszenarien (im Fall von RL) vom Kunden gestellt werden, besteht die erste Aktivität hier in einer *Definition* der Ziele und Metriken, die alle Aspekte der Kundenspezifikation berücksichtigen. Anschließend findet mit der *On-Site Adaptation* des bereits trainierten ML-Modells eine Anpassung an die neuen Gegebenheiten statt, üblicherweise in Form eines zusätzlichen Trainings. Dessen Ziel ist es, die kundenspezifischen Anforderungen besser zu erfüllen und die Auswirkungen eines eventuell auftretenden *Sim2Real Gap* zu verringern. Wenn eine größere Anpassung erforderlich ist, beispielsweise eine hochgradige Spezialisierung, könnte die Anpassung auf einen Zyklus ausgedehnt werden, der dem Trainings-Test-Zyklus der Entwicklungsphase ähnelt. Ebenso ist eine Feedbackschleife skizziert, die im Falle eines zu großen Sim2Real Gap zurück in die Entwicklungsphase führt, beispielsweise, um das Training auf Basis einer anderen Modellarchitektur erneut durchzuführen. Vorausgesetzt, die vorherigen Aktivitäten wurden gründlich durchgeführt, ist es wahrscheinlich, dass es sich bei der Anpassung an die spezifischen Gegebenheiten um eine schnelle, schlanke Aktivität handelt. Das anschließende *Onboarding* des angepassten ML-Modells, eine Mischung aus Abnahmetest und Schulung des Kunden, ist die letzte Aktivität während der Auslieferungsphase. Das Ergebnis des Onboarding kann erneut als eine Art Gütesiegel aufgefasst werden, das aussagt, zu welchem Grad das angepasste ML-Modell die Kundenspezifikation auf den entsprechenden Daten (im Fall von SL) respektive Einsatzszenarien (im Fall von RL) erfüllt.

Betrieb Nach erfolgreichem Onboarding wird das angepasste ML-Modell im regulären Betrieb eingesetzt. Hier ist die Annahme, dass sich die Spezifikation erneut von der vorhergegangenen Kundenspezifikation unterscheidet, beispielsweise um während des Onboardings erkannte Besonderheiten der Anwendungsfälle oder einem großangelegten Einsatz des ML-Modells Rechnung zu tragen. Dabei kann auch Zuverlässigkeit im Sinne der Reaktionszeit und Verfügbarkeit eine Rolle spielen, weshalb der Begriff *Service Level Agreement* (SLA) gewählt wurde. Die Spezifikation muss deshalb aber nicht zwangsläufig wie ein SLA ausgestaltet werden. Eine weitere Annahme ist, dass die Unterschiede zwischen SLA und Kundenspezifikation (wesentlich) kleiner ausfallen als die Unterschiede zwischen Entwicklungs- und Kundenspezifikation. Daher sieht das Prozessmodell hier keine erneute Anpassung des ML-Modells, aber eine erneute *Definition* der Ziele samt der entsprechenden Metriken vor. Diese sind für die laufende Überwachung des ML-Modells erforderlich, beispielsweise zur Erkennung von *Drift*, einer langsamen Veränderung der Daten (im Fall von SL) oder der Umgebung (im Fall von RL). Drift kann ansonsten im Laufe der Zeit zu einer langsamen Verschlechterung der Vorhersagequalität des ML-Modells

führen und in stillem Versagen enden [141]. Außerdem ist die Identifikation von Situationen, in denen das ML-Modell unzureichende Leistungen erbringt, der Schlüssel für präzises Feedback für das Training künftiger ML-Modelle, beispielsweise durch Bereitstellung von Daten oder Szenarien, die diese Situationen abbilden.

5.4.5 Anwendungsbeispiel

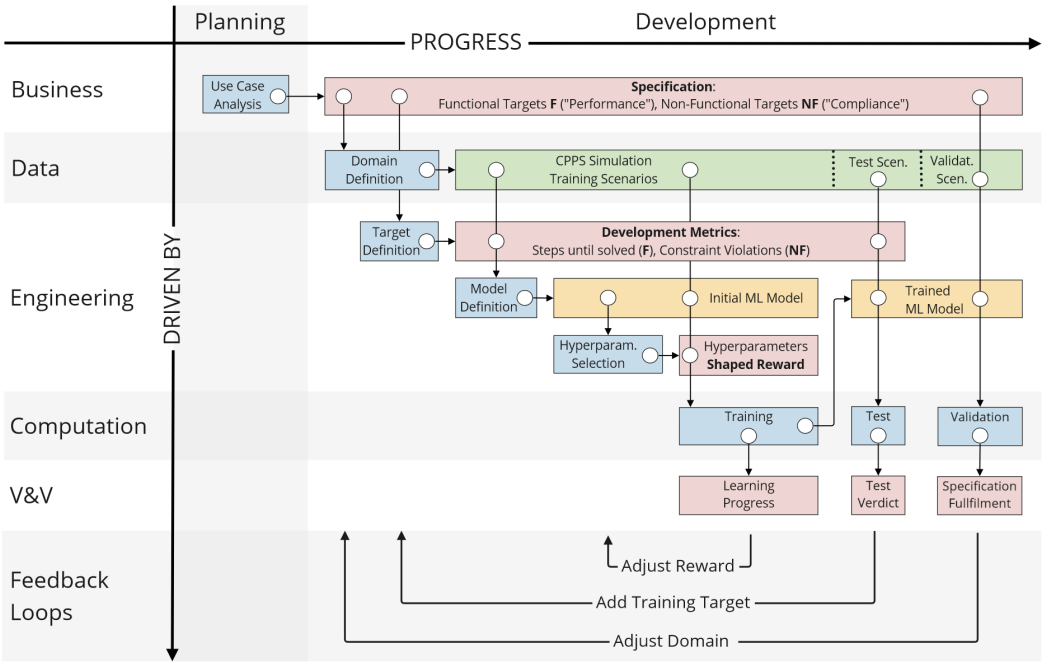


Abbildung 5.3: Darstellung des Vorgehens des im vorherigen Kapitel 3 behandelten Ansatzes SAT-MARL [121, 122] mit der grafischen Darstellung des Prozessmodells. Über die Feedbackschleifen könnten die Konfiguration der Simulation, das schrittweise Ergänzen von Zielen und deren Abbildung in Rewards künftig automatisiert und Bestandteil einer Meta-Optimierung werden. Darstellung entnommen aus [123].

Dieses Unterkapitel stellt ein konkretes Anwendungsbeispiel des Prozessmodells vor. Abbildung 5.3 zeigt das Vorgehen des im vorherigen Kapitel 3 behandelten Ansatzes SAT-MARL [121, 122] mit der grafischen Darstellung des Prozessmodells. Der Kern von SAT-MARL ist, den Agenten in einem kooperativen MAS funktionale und nicht-funktionale Anforderungen einer Spezifikation über Rewards zugänglich zu machen, sodass sie mit RL lernen können, die Spezifikation zu erfüllen. In Abbildung 5.3 ist die Spezifikation auf oberster Ebene mit funktionalen und nichtfunktionalen Zielen abgebildet, deren Erfüllung über

Leistung (original: Performance) und *Konformität* (original: Compliance) gemessen wird. Auf mittlerer Ebene sind entsprechende Entwicklungsmetriken vorgesehen, die messen, wie lange die Agenten benötigen, um ihre Aufgaben zu lösen (original: Steps until solved) und wie oft sie gegen Beschränkungen verstoßen (original: Constraint Violations). Auf unterster Ebene wird der im Stil von *Potential Based Reward Shaping* (PBRS) definierte Reward verwendet, um den Trainingsfortschritt zu messen. Ein wichtiger Aspekt von SAT-MARL ist die schrittweise Abbildung der Anforderungen in Rewards. Beispielsweise war das Ziel der ersten Trainingsphase, die Leistung zu maximieren. Die Maximierung der Konformität wurde in einer zweiten Trainingsphase hinzugenommen. Realisiert wurde das über eine Änderung der Rewards im Training. In Abbildung 5.3 wird dieses Vorgehen über zwei Feedbackschleifen dargestellt, die das Trainingsziel und den Reward anpassen. Eine dritte Feedbackschleife berücksichtigt, dass in initialen Experimenten auch die Simulation, beispielsweise hinsichtlich des Layouts und der Anzahl an Agenten, angepasst wurde. Das Erstellen einer geeigneten Simulation für eine gegebene Problemstellung ist eine praktische Herausforderung, die in der Wissenschaft meist keine große Beachtung findet. Es ist jedoch wichtig, dies im Kontext einer Selbstanpassung darzustellen, da auch dies eine der zeitintensiven, manuellen Tätigkeiten ist, die künftig automatisiert werden kann. Beispielsweise kann eine Problemstellung initial zu schwierig für die Agenten sein, wenn sich diese durch die Exploration gegenseitig behindern. In diesem Fall könnte das Training mit einer kleinen Anzahl gleichzeitig trainierender Agenten beginnen. Anschließend könnte die dritte Feedbackschleife die Anzahl der Agenten schrittweise erhöhen und den Schwierigkeitsgrad so langsam steigern. Die Smart Factory Simluation aus SAT-MARL modelliert Aspekte einer Produktion der nahen Zukunft, wurde in dieser Form aber noch nicht physisch realisiert. Daher endet Abbildung 5.3 mit der Entwicklungsphase und stellt die Auslieferungs- und Betriebsphase nicht dar.

5.5 Diskussion

Dieses Kapitel diskutiert das Prozessmodell hinsichtlich Stärken und Schwächen im Kontext der Zielsetzung, skizziert weitere Anwendungsmöglichkeiten und beschreibt, wie künftige Arbeiten diese umsetzen könnten.

Das Prozessmodell konzentriert sich auf den Lebenszyklus des ML-Modells, weshalb das Data Management nur stark vereinfacht dargestellt ist. Einen Datensatz oder eine Simulation so zu konstruieren, dass ML-Modelle bestimmte Merkmale oder Fähigkeiten erlernen können, ist eine nicht zu unterschätzende Herausforderung. Eine Ergänzung entsprechender Aktivitäten ist im entwickelten Prozessmodell grundsätzlich möglich. Hierfür können Feedbackschleifen und V&V-Aktivitäten zur Sicherstellung der Datenqualität in ähnlicher Weise eingesetzt werden, wie das Prozessmodell das zur Qualitätssicherung des

ML-Modells skizziert hat. Offen ist hingegen, inwieweit die Sicherstellung der Qualität von Daten beziehungsweise Datensätzen automatisiert werden kann. Je nach Anwendungsfall ist gesetzlich klar geregelt, welche Merkmale in ML-basierten Verfahren verwendet werden dürfen, und es benötigt eine präzise Überprüfung, ob solche Merkmale in den Daten enthalten sind. Derzeit geschieht das Kennzeichnen (englisch: Labeling) von Daten im besten Fall teil-, aber nicht voll-automatisiert [31]. Auch LLMs, deren erster Trainingsschritt mit *Semi-Supervised Learning* durchgeführt wird, das keine gekennzeichneten Daten benötigt, werden anschließend einer Feinjustierung unterzogen, beispielsweise mit von Menschen konstruierten Beispielen oder Bewertungen im Rahmen von RLHF [2, 28]. Für die Feinjustierung des quelloffenen LLM *Llama2* wurden laut den Entwicklern [147] mehr als eine Million von Menschen gekennzeichnete Beispiele verwendet. Das ist einen potentieller Flaschenhals.

Ebenso ist die grafische Darstellung nicht optimal, um ein synchronisiertes (paralleles) Training mehrerer ML-Modelle mit unterschiedlichen Aufgaben innerhalb desselben Prozesses übersichtlich darzustellen. Für eine Darstellung, in der sich Assoziationen seltener kreuzen, wäre es erforderlich, die Kategorisierung auf der y-Achse aufzubrechen, oder eine z-Achse hinzufügen. Ein Beispiel ist das abwechselnde Training von Teams aus Protagonisten und Antagonisten in ARTS [109]. Dieses würde von systematischer Automatisierung profitieren, wobei Feedbackschleifen die Häufigkeit und die Dauer des Trainings der Agenten des jeweiligen Teams so steuern, dass keines der Teams dauerhaft die Oberhand gewinnt.

Ein unmittelbares Ziel des Prozessmodells war, eine Brücke zwischen etablierten Vorgehensweisen der Softwareentwicklung und spezialisierten ML-Trainingsverfahren zu bauen. Das Prozessmodell ist nicht auf die Darstellung einer bestimmten ML-Technik beschränkt, sondern in der Lage, Verfahren des Supervised, Unsupervised und Reinforcement Learning darzustellen. Ebenso ist es nicht auf eine bestimmte Vorgehensweise bei der Softwareentwicklung beschränkt, jedoch ist die iterative Vorgehensweise bei der Entwicklung der ML-Modelle leichter mit agilen Vorgehensweisen zu synchronisieren und Modularität, also die Möglichkeit, das ML-Modell schnell und häufig auszutauschen, ist hilfreich.

Ein übergeordnetes Ziel des Prozessmodells war, die Entwicklung von ML-Modellen als Meta-Optimierungsproblem im Kontext des gesamten MLLebenszyklus darzustellen, auf dessen Basis eine Selbstoptimierung formuliert werden kann. Die Motivation ist insbesondere zu vermeiden, dass menschliche Experten den Entwicklungsprozess wiederholt an Probleme und Algorithmen anpassen, da dies nur bedingt skaliert, insbesondere in weniger gut verstandenen Problemen, da es hier an Erfahrungswerten fehlt. Ein Schlüsselelement ist eine Repräsentation von Zielen und Metriken in drei Ebenen: die Entwicklungsspezifikation auf oberster Ebene, die Trainingsziele auf mittlere Ebene und die Hyperparameter auf unterster Ebene, bei denen auch technische Metriken wie

der Loss oder der Reward verortet sind. Diese Repräsentation vereinfacht die Automatisierung, da diese stets auf einer Ebene erfolgt und ein konkretes und überprüfbares Ziel verfolgt. Mit DL Verfahren stehen skalierbare, leistungsstarke Optimierungstechniken zur Verfügung und in der Softwareentwicklung herrscht ohnehin der Trend zu Automatisierung, beispielsweise in Form von DevOps [81], MLOps [75] und Auto-ML [58]. Insofern ist es naheliegend, DL Techniken und Automatisierung auch auf der Ebene des Entwicklungsprozesses anzuwenden. Dafür eignen sich Algorithmen des SD, beispielsweise RL oder genetische Algorithmen. Unter Berücksichtigung der konzeptionellen Überschneidung von *Ensembles* [64] und MARL in kooperativen Szenarien [122] stehen damit die erforderlichen Bausteine zur Verfügung, um selbstanpassende, autonome Systeme zu realisieren. Dabei sollten bewährte Praktiken des Autonomic Computing [157] berücksichtigt werden, beispielsweise, wie mit vorhandenem Wissen umzugehen ist und wie dieses erweitert werden kann.

Die in der Blaupause vorgeschlagene Integration von V&V-Aktivitäten dient zur Qualitätskontrolle beim Übergang in die Auslieferungs- und in die Betriebsphase. Sie kann darüber hinaus genutzt werden, eine automatisierte Suche nach Modellarchitektur und Hyperparametern im Stil von Auto-ML [58] zu steuern. Je nach Anwendungsfall können evolutionäre Methoden [42], lernende Methoden [109] oder Monte-Carlo-basiertes, statistisches Model Checking [101] zum Testen eingesetzt werden. Werden numerische statt binärer Metriken verwendet [39] und die Spezifikation als Mindestanforderung betrachtet, ermöglicht das, Systeme so weiterzuentwickeln, dass sie die Spezifikation deutlich übertreffen, woraus eine erhöhte Robustheit erwachsen kann. Es ist möglich, dass die Entwicklungsspezifikationen, die Kundenspezifikation und das SLA in menschlicher Sprache formuliert sind und sich im Verlauf des Prozesses ändern. Basierend darauf stellt sich die Frage, wie die Entwicklungsspezifikation zu definieren ist, dass erstens die Lücke zur unmittelbar folgenden Kundenspezifikation und zur später folgenden SLA möglichst klein ist und zweitens ein ML-Modell, dass die Entwicklungsspezifikation erfüllt, im Idealfall ebenso die Kundenspezifikation und das SLA erfüllt. Ebenso gibt es Ansätze, die eine Spezifikation in einer formalen Repräsentation erfordern [12]. Im Sinne der Automation besteht hier das Potential, die Abbildung der Spezifikation in eine formale Repräsentation teilweise oder vollständig durch LLMs durchführen zu lassen [10].

Darüber hinaus sollte es möglich sein, Prozesse wie die Blaupause (mit ausdefinierten Feedbackschleifen) konsistent mathematisch zu formulieren, beispielsweise mit CTL* [38]. Dies würde eine Validierung von Prozessinstanzen ermöglichen, die beispielsweise durch eine Meta-Optimierung erzeugt werden. Aber auch ohne Meta-Optimierung erlaubt eine solche mathematische Formulierung, Abläufe zu überprüfen, beispielsweise „die Entwickler hielten sich immer an den definierten Prozess“, klare Prozessziele zu definieren, beispielsweise „eine Instanz der On-Site Adaptation soll vor $t = 100$ im Zustand *done* sein“, und ein Verständnis der Zusammenhänge zu schaffen, beispielsweise

„jedes Mal, wenn die Zieldefinition nach der Datenauswahl begann, endete die Entwicklung erfolgreich“. Weitere Möglichkeiten ergeben sich aus der Verknüpfung von Aussagen über den Entwicklungsprozess mit Aussagen über das resultierende ML-Modell. Werden Artefakte wie logische Aussagen betrachtet, können Anforderungen an den Entwicklungsprozess definiert werden, um im Gegenzug Garantien für die resultierende Software zu erhalten. Mit einer entsprechend ausgeprägten Logik könnten Qualitätssiegel wie „das ML-Modell hat drei Trainingszyklen hintereinander erfolgreich absolviert“ formuliert werden. Hier ist die Motivation, dass das Training von ML-Modellen ein stochastischer Prozess ist, der nicht nur zu Verbesserungen, sondern auch zu Verschlechterungen führen kann. Ein wiederholt positives Testergebnis zu mehreren, direkt aufeinanderfolgenden Zeitpunkten spricht für einen hohen Reifegrad des ML-Modells. Wenn künftig Rollen in das Prozessmodell einbezogen werden, könnte ebenso formuliert werden, dass die „Datenauswahl von mindestens zwei Entwicklern mit jeweils mindestens zehn Jahren Erfahrung“ durchgeführt werden muss. Dies könnte ebenfalls Teil der Spezifikation sein. Solche Varianten der Qualitätssicherung sind in Disziplinen üblich, die sich nicht auf formale Beweise oder umfangreiche Tests stützen können, beispielsweise der Medizin, in der kritische Diagnosen von mehreren erfahrenen Ärzten unabhängig voneinander durchgeführt werden müssen. Die Entwicklung von ML-Software ist diesen Disziplinen mitunter ähnlich, weshalb es wichtig ist, nicht nur das Produkt, sondern auch den Prozess zu betrachten.

Zu guter Letzt ist eine Schlüsselannahme, dass eine Validierung von ML-Entwicklungsprozessen zu besseren ML-Modellen führt. Bislang konnte in den zu Grunde liegenden Forschungsprojekten jedoch keine ML-Komponente über den gesamten Lebenszyklus begleitet werden, weshalb die Blaupause in dieser Hinsicht unerprobt ist. Dies kann gemeinsam mit einer konsistenten mathematischen Formulierung in einer künftigen Arbeit umgesetzt werden.

5.6 Zusammenfassung

Die zentrale These dieser Arbeit ist, dass lernende Agenten eigenständig Probleme auf individueller und globaler Ebene lösen können. In diesem Abschnitt wurde ein Prozessmodell beschrieben, das den gesamten Lebenszyklus von ML-Modellen umfasst.

Wichtige Erkenntnisse aus der Analyse verwandter Arbeiten sind die Bedeutung der Qualitätssicherung bei der Entwicklung selbstanpassender Systeme und das Fehlen einer präzisen Darstellung der Abhängigkeiten zwischen Aktivitäten und Artefakten bei der Entwicklung ML-basierter Systeme. Beides wurde in der Konzeption des Prozessmodells aufgegriffen. Dessen formale Definition beinhaltet eine Notation von Zeit, die erlaubt, zu jedem Zeitpunkt Aussagen über den Fortschritt des Prozesses und seinen Zustand zu treffen.

Anschließend wurde eine grafische Syntax für das Prozessmodell anhand einer Blaupause vorgestellt. Diese Blaupause kann um konkrete Feedbackschleifen ergänzt werden, um das Vorgehen bei der Verwendung verschiedener ML-Techniken abzubilden. Eine Stärke und ein Alleinstellungsmerkmal des Prozessmodells ist, dass sich nicht nur das ML-Modell, sondern auch Artefakte wie die Spezifikation im Lauf der Zeit verändern (dürfen). Dennoch wird klar dargestellt, wie Aktivitäten und Artefakte der Qualitätssicherung, beispielsweise Tests und Testergebnisse, eingesetzt werden können, um sicherzustellen, dass das ML-Modell letztlich die für den Betrieb relevante Spezifikation erfüllt. Ermöglicht wird das durch die Repräsentation von Zielen und Metriken in drei Ebenen, die in dieser Form noch nicht in der Literatur berichtet wurden. Verbesserungspotential besteht hinsichtlich einer ausführlicheren Betrachtung des Data Management.

Auf Basis der Formalisierung könnten Entwicklungsprozesse mathematisch formuliert werden. Dies würde eine Qualitätssicherung auf Prozessinstanzen ermöglichen, also einer Überprüfung, ob ein konkretes Vorgehen bei der ML-Entwicklung definierten Regeln folgt. Würden zusätzlich Rollen berücksichtigt, wäre es möglich, Anforderungen an Prozessinstanzen zu formulieren, beispielsweise hinsichtlich der Anzahl und Erfahrung der beteiligten Entwickler. Dies alleine ist kein Garant für Qualität, aber ein wichtiger Baustein: Es ist essentiell, sowohl die einzelnen Elemente des Entwicklungsprozesses (Aktivitäten und Artefakte), als auch den Ablauf des Entwicklungsprozess und die daran Beteiligten zu berücksichtigen.

Das Prozessmodell verknüpft die in der Praxis zunehmende Automatisierung einzelner ML-Entwicklungsschritte, beispielsweise durch *Auto-ML* [58] und moderne Vorgehensweisen, beispielsweise *MLOps* [75, 91], mit den in dieser Arbeit vorgestellten Ansätzen der Selbstorganisation und Selbstausrichtung. Dabei skizziert die Blaupause eines ML-Entwicklungsprozesses, wie mit diesen Bausteinen ein systematisches, iteratives Vorgehen über mehrere Entwicklungsschritte und -zyklen hinweg realisiert werden kann und wie die Integration von Qualitätssicherung eine gezielte Steuerung der Automatisierung ermöglicht. Künftige Arbeiten können mit entsprechenden Meta-Zielen eine selbstständige Anpassung ML-basierter Software auf Basis des Prozessmodells formulieren und darstellen.

6 Fazit

Autonome Systeme werden künftig häufiger aufeinander und auch auf Menschen treffen, beispielsweise im Straßenverkehr, der eine gleichermaßen offene und regulierte Umgebung darstellt. Wie eingangs erwähnt, fahren PKW Stand heute noch nicht durchweg autonom, da unter anderem die Fähigkeit der Steuersysteme, das Verhalten an dynamische Situationen und Probleme anzupassen, noch nicht ausreichend ausgeprägt ist. Ähnliche Herausforderungen gibt es beispielsweise in industriellen Produktionsstraßen, in denen FTF Stand heute meist noch durch zentrale Controller koordiniert werden. Diese Arbeit hat demonstriert, dass autonome Agenten mittels MARL lernen können, sowohl in offenen Systemen ohne Regeln, als auch in geschlossenen Systemen mit klar definierten Regeln reaktiv, proaktiv und sozial zu handeln, wobei sie insbesondere zu situativer Koordination und Kooperation fähig sind. Anschließend wurde dargelegt, wie solche Agenten unter Berücksichtigung aktueller Vorgehensweisen und Techniken zielgerichtet entwickelt werden können und wie sich solche Systeme künftig automatisch weiterentwickeln ließen. Dies eröffnet neue Perspektiven für künftige Arbeiten, denn angesichts der vielfältigen Einsatzmöglichkeiten autonomer Systeme bleibt noch ein weiter Weg, bis die Herausforderungen in dynamischen Multi-Agenten-Szenarien in der Praxis bewältigt sind. Das direkt folgende Unterkapitel 6.1 fasst die wichtigsten Erkenntnisse dieser Arbeit zusammen und das abschließende Unterkapitel 6.2 erläutert, wie künftige Arbeit daran anknüpfen können.

6.1 Zusammenfassung

Kapitel 3 betrachtete Agenten, die individuelle Ziele in einer offenen, unkontrollierten Umgebung verfolgen. Der Ansatz war, die Agenten im Rahmen einer *Selbstorganisation* lernen zu lassen, wann und wie sie miteinander interagieren. Die Agenten mussten Einigkeit über die Nutzung einer gemeinsamen Ressourcen erlangen, um vorübergehend zusammenarbeiten zu können. Besonderes Augenmerk lag auf der Analyse der Umstände, die die Entstehung von Koordination und Kooperation ermöglichten. Dafür wurde die Problemstellung, die im RL durch die Umgebung repräsentiert wird, systematisch variiert. Insbesondere wurde auf Mechanismen verzichtet, die in der Literatur genutzt werden, um Kooperation zu forcieren, beispielsweise Kommunikation [66] oder zentrale Koordination [88]. So wurde eine direkte Einflussnahme auf die Agenten vermieden. In den Experimenten wurden Bedingungen identifiziert, unter denen

bis drei RL-Agenten eine Tragik der Allmende vermieden (Koordination) sowie aktiv zusammenarbeiteten, um gemeinsam ein ad hoc gewähltes Ziel zu erreichen (Kooperation). Dies kann als *starke Selbstorganisation* [35] gewertet werden, da es in den untersuchten Szenarien keine explizite zentrale Kontrolle gab, weder intern noch extern. AMAS Ansätze können solchen Szenarien nur lösen, wenn (alle) Konfliktsituationen sowie erstrebenswerte Lösungen vorab bekannt sind, da sie auf a priori definierte Regeln [16] zurückgreifen. Die vorliegenden Ergebnisse zeigen, dass lernende Agenten auch ohne a priori definierte Regeln erfolgreich sein können.

Kapitel 4 betrachtete Agenten in Situationen, in denen eine klare Spezifikation des Problems und des gewünschten Verhaltens vorliegt. Der Ansatz war, die Agenten im Rahmen einer *Selbstausrichtung* mit der gesamten Spezifikation zu trainieren, sodass sie selbst lernen, diese einzuhalten und ein zuverlässiges Gesamtsystem zu bilden. Die Agenten mussten in einem kooperativen Szenario funktionale Aspekte wie die Durchführung mehrerer Produktionsschritte in begrenzter Zeit und nicht-funktionale Aspekte wie die Vermeidung von Kollisionen erfüllen. Besonderes Augenmerk lag auf gleichermaßen performantem und sicherem Verhalten. Die Spezifikation wurde in Rewards abgebildet, die den Agenten im RL Rückmeldung über ihr Verhalten geben. Zwei große Herausforderungen waren das Erlernen von Sicherheitsanforderungen, die gegeben einer Ausnahmesituation für deren Dauer mit den funktionalen Anforderungen in Konflikt stehen, sowie das Skalieren des Systems auf acht Agenten. Diese Herausforderungen wurden durch ein Training mit adaptiven Rewardschemata überwunden. Diese bilden initial nur eine reduzierte Menge funktionaler Anforderungen ab und ergänzen weitere Anforderungen, sobald die Lernkurve der Agenten zu konvergieren beginnt. Es wurde gezeigt, dass so trainierte Agenten ihr Verhalten bereits im Normalfall so anpassen, dass sie die Sicherheitsanforderungen im Ausnahmefall besser erfüllen können, ohne dass die Performance im Normalfall signifikant einbricht. Dies kann als stark ausgeprägte Form *proaktiven Verhaltens* [160] gewertet werden. Mit Ansätzen wie Fallback-Controllern [105, 106] oder Shielding [5] ist ein solches Verhalten schwerer zu erzielen, da diese Ansätze entweder Leistung oder Sicherheit auf Kosten des jeweils anderen Aspekts optimieren, da sie strikt zwischen Normal- und Ausnahmefall trennen.

Kapitel 5 betrachtete die vorhergegangenen Ansätze auf abstrakterer Ebene im Kontext eines ML-Entwicklungszyklus und ging der Frage nach, wie auf dieser Basis mit Hilfe von Meta-Zielen *selbstanpassende* Systeme realisiert werden können. Dazu wurde ein formales Prozessmodell mit grafischer Darstellung für die Entwicklung ML-basierter Systeme vorgeschlagen, das systematisch durch deren Lebenszyklus führt und eine gezielte Kontrolle von (automatisierter) Anpassung ermöglicht. Die Konzeption des Prozessmodells zielt auf eine klare Darstellung des Zusammenwirkens von Aktivitäten und Artefakten, insbesondere hinsichtlich der Qualitätssicherung. Die formale Definition des Prozessmodells enthält eine Notation von Zeit und erlaubt, Aussa-

gen über den Zustand und Fortschritt des Prozesses zu treffen. Die grafische Darstellung beinhaltet eine Blaupause, die den gesamten Lebenszyklus von ML-Komponenten erfasst und das Vorgehen bei der Entwicklung verschiedene ML-Techniken abbilden kann. Ein Alleinstellungsmerkmal des Prozessmodells ist, dass sich auch Artefakte wie die Spezifikation im Lauf der Zeit verändern (dürfen). Insgesamt verknüpft das Prozessmodell die vorgestellten Ansätze zur Selbstorganisation und Selbstausrichtung mit der Automatisierung einzelner ML-Entwicklungsschritte, beispielsweise durch *Auto-ML* [58], und moderne Vorgehensweisen, beispielsweise *MLOps* [75, 91]. Dabei skizziert die Blaupause, wie mit diesen Bausteinen ein systematisches, iteratives Vorgehen über mehrere Entwicklungszyklen hinweg realisiert werden kann und wie dessen Ablauf automatisiert und durch Qualitätssicherung kontrolliert werden kann.

6.2 Ausblick

Hinsichtlich der *Selbstorganisation* ist die wichtigste Frage für künftige Arbeiten, wie sich das in Kapitel 3 vorgestellte Prinzip am besten in die Praxis übertragen lässt. Bei Einsatz der Technik *Parameter Sharing*, die mit drei Agenten erfolgreich angewandt wurde, ist es hinsichtlich der erforderlichen Rechenleistung vorteilhaft, wenn eine möglichst große Anzahl Agenten die gleiche Strategie mit ihrer Erfahrung verbessert. Jedoch ist es aus algorithmischer Sicht auch möglich, dass aufgrund der Nichtstationarität in MAS eine gemeinsame Strategie leichter mit einer kleinen Anzahl von Agenten gelernt werden kann. Diese Strategie kann anschließend auf weitere Agenten übertragen werden, die hinsichtlich ihrer Wahrnehmung und ihres Aktionsraums kompatibel sind. Dabei ist dann zu analysieren, ob die wesentlich größere Anzahl Agenten zur Laufzeit ohne unerwünschte Folgen bleibt. Grund zur Annahme, dass dieses Vorgehen funktionieren kann, bieten beispielsweise die *Boids-Regeln* für Schwarmsysteme [118], die auf Basis einer begrenzten Anzahl lokaler Informationen (n nächste Nachbarn) formuliert sind und eine Schwarmformation mit beliebig vielen Individuen erzeugen. In diesem Kontext ist ebenso relevant, dass Konventionen der alltäglichen Interaktion häufig universell, also auf andere Bereiche übertragbar sind. Beispielsweise weichen sich Fußgänger in einem Land mit Rechtsverkehr auch auf dem Bürgersteig überwiegend rechts aus. Im Bereich autonomer Systeme wurde noch nicht untersucht, ob solche universellen Konventionen gelernt werden (können), oder ob diese Konventionen mit von anderen Agenten gelernten Konventionen kompatibel sind.

Im Bereich der *Selbstausrichtung* fehlt dem in Kapitel 4 vorgestellten Ansatz ein automatisiertes Verfahren, um zu bestimmen, wann welche Anforderung in das Training einfließt und wie stark diese gegenüber den anderen Anforderungen gewichtet wird. Hier bietet sich für künftige Arbeiten an, genetische Algorithmen [71] zu verwenden, die nicht die Strategien der lernenden Agenten verändern, sondern die Gewichtung der einzelnen Rewardkomponenten.

Dies hätte gegenüber bekannten Anwendungen von populationsbasiertem Training [68, 149] einen Effizienzvorteil, da die Agenten nicht jedes mal von Grund auf trainiert werden müssen. Vorausgesetzt, dass diese Agenten auch künftig Funktionsapproximation in Form von NN nutzen und kein Beweis für ein spezifisches Verhalten geführt werden kann, kann der vorgestellte Ansatz nicht ohne zusätzliche Maßnahmen in sicherheitskritischen Bereichen eingesetzt werden. Jedoch ist das manuelle Design von Mechanismen, die beispielsweise zu beweisbarem Verhalten führen, mit zunehmend autonomen Agenten problematisch, insbesondere in komplexen Szenarien. Hier besteht die Möglichkeit für künftige Arbeiten darin, den vorgestellten Ansatz für unkritische Anforderungen zu nutzen und so die Anzahl und Komplexität manuell zu erstellender Sicherheitsschichten auf die Anforderungen zu reduzieren, bei denen eine formale Verifikation strikt erforderlich ist.

Das Prozessmodell zur systematischen Entwicklung ML-basierter Systeme aus Kapitel 5, auf dessen Basis eine *Selbstanpassung* formuliert werden kann, ist durch die zu Grunde liegenden Forschungsprojekte nur zur Hälfte durch praktische Erfahrungswerte gestützt. Ein erster Schritt für künftige Arbeiten ist, die Phasen *Auslieferung* und *Betrieb* an praktischen Erfahrungen zu messen, die zum jetzigen Zeitpunkt noch fehlen. Ein weiterer Schritt ist, Entwicklungsprozesse auf Basis der Formalisierung unter Berücksichtigung von Rollen mathematisch zu formulieren, um eine Qualitätssicherung für Entwicklungsprozesse zu ermöglichen. Es ist wichtig, nicht nur die einzelnen Elemente des Entwicklungsprozesses (Aktivitäten und Artefakte), sondern auch den Ablauf des Entwicklungsprozesses und die daran Beteiligten bei der Qualitätssicherung zu berücksichtigen. Dabei kann gleichermaßen überprüft werden, ob ein entsprechend verifizierter Entwicklungsprozess tatsächlich zu besserer ML-Software führt. Eine weitere Möglichkeit für künftige Arbeiten ist, auf Basis des Prozessmodells eine selbstständige Anpassung für MAS zu realisieren, die auf die vorgestellten Ansätze zur Selbstorganisation und Selbstausrichtung zurückgreift. Die Komplexität der Interaktion ist in beiden Fällen auf die Agenten ausgelagert, daher müsste die Optimierung auf Prozessebene primär nach geeigneten Trainingsbedingungen für die Agenten suchen.

Abkürzungsverzeichnis

AMAS Adaptive Multi-Agent Systems 20

CTDE Centralized Training for Dezentralized Execution 17

CRISP-ML(Q) Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance Framework . . 27

DL Deep Learning 2

DQN Deep-Q-Networks 13

FTF Fahrerlose Transportfahrzeuge 1

IL Independent Learning 17

MARL Multi-Agent Reinforcement Learning 1

MAS Multi-Agenten-System 7

ML Machine Learning 1

MLOps Machine Learning Operations 27

PBRS Potential Based Reward Shaping 70

POMDP Partially Observable Markov Decision Process 8

POSG Partially Observable Stochastic Game 9

PPO Proximal Policy Optimization 15

Abbildungsverzeichnis

Literatur

[1] S. Abdallah und V. Lesser. „Multiagent Reinforcement Learning and Self-Organization in a Network of Agents“. In: *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2007, S. 1–8. DOI: 10.1145/1329125.1329172.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat et al. „GPT-4 Technical Report“. In: *preprint* arxiv:2303.08774 (2024). DOI: 10.48550/arXiv.2303.08774.

[3] R. Akkiraju, V. Sinha, A. Xu, J. Mahmud, P. Gundecha, Z. Liu, X. Liu und J. Schumacher. „Characterizing Machine Learning Processes: A Maturity Framework“. In: *Business Process Management*. Springer, 2020, S. 17–31. DOI: 10.1007/978-3-030-58666-9_2.

[4] S. V. Albrecht, F. Christianos und L. Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2023. URL: https://www.marl-book.com.

[5] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum und U. Topcu. „Safe Reinforcement Learning via Shielding“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2018, S. 2669–2678. URL: https://dl.acm.org/doi/10.5555/3504035.3504361.

[6] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi und T. Zimmermann. „Software Engineering for Machine Learning: A Case Study“. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2019, S. 291–300. DOI: 10.1109/ICSE-SEIP.2019.00042.

[7] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman und D. Mané. „Concrete Problems in AI Safety“. In: *preprint* arxiv:1606.06565 (2016). DOI: 10.48550/arXiv.1606.06565.

[8] R. Axelrod und D. Dion. „The Further Evolution of Cooperation“. In: *Science* 242.4884 (1988), S. 1385–1390. DOI: 10.1126/science.242.4884.1385.

[9] L. Belzner, M. T. Beck, T. Gabor, H. Roelle und H. Sauer. „Software engineering for distributed autonomous real-time systems“. In: *2016 IEEE/ACM 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*. 2016, S. 54–57. DOI: 10.1145/2897035.2897040.

[10] L. Belzner, T. Gabor und M. Wirsing. „Large Language Model Assisted Software Engineering: Prospects, Challenges, and a Case Study“. In: *Bridging the Gap Between AI and Reality*. Springer, 2024, S. 355–374. DOI: 10.1007/978-3-031-46002-9_23.

[11] Y. Bengio, J. Louradour, R. Collobert und J. Weston. „Curriculum Learning“. In: *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*. 2009, S. 41–48. DOI: https://doi.org/10.1145/1553374.1553380.

[12] L. Berducci, E. A. Aguilar, D. Ničković und R. Grosu. „Hierarchical Potential-based Reward Shaping from Task Specifications“. In: *preprint* arxiv:2110.02792 (2022). DOI: 10.48550/arXiv.2110.02792.

[13] L. Berducci, S. Yang, R. Mangharam und R. Grosu. *Learning Adaptive Safety for Multi-Agent Systems*. 2023. DOI: 10.48550/arXiv.2309.10657.

[14] C. Bernon, M.-P. Gleizes, S. Peyruqueou und G. Picard. „ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering“. In: *Engineering Societies in the Agents World III*. Springer, 2003, S. 156–169. DOI: 10.1007/3-540-39173-8_12.

[15] M. B. Bezcioglu, B. Lennox und F. Arvin. „Self-Organised Swarm Flocking with Deep Reinforcement Learning“. In: *7th International Conference on Automation, Robotics and Applications (ICARA)*. 2021, S. 226–230. DOI: 10.1109/ICARA51699.2021.9376509.

[16] J. Boes und F. Migeon. „Self-organizing multi-agent systems for the control of complex systems“. In: *Journal of Systems and Software* 134 (2017), S. 12–28. DOI: 10.1016/j.jss.2017.08.038.

[17] O. Boissier, J. F. Hübner und J. S. Sichman. „Organization Oriented Programming: From Closed to Open Organizations“. In: *Engineering Societies in the Agents World (ESAW) VII. Revised Selected and Invited Papers*. Lecture Notes in Computer Science 4457 (2007), S. 86–105. DOI: 10.1007/978-3-540-75524-1_5.

[18] N. Bonjean, W. Mefteh, M. P. Gleizes, C. Maurel und F. Migeon. „ADELFE 2.0“. In: *Handbook on Agent-Oriented Design Processes*. Springer, 2014, S. 19–63. DOI: 10.1007/978-3-642-39975-6_3.

[19] J. Bosch, H. Holmström Olsson und I. Crnkovic. „Engineering AI Systems“. In: *Accelerating Digital Transformation*. Springer, 2022, S. 407–425. DOI: 10.1007/978-3-031-10873-0_18.

[20] P. Bourque und R. E. Fairley, Hrsg. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Version 3.0. IEEE Computer Society, 2014. ISBN: 978-0-7695-5166-1. URL: https://www.swebok.org.

[21] C. Boutilier. „Planning, Learning and Coordination in Multiagent Decision Processes“. In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*. 1996, S. 195–210. URL: https://dl.acm.org/doi/10.5555/1029693.1029710.

[22] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis und S. Colton. „A Survey of Monte Carlo Tree Search Methods“. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), S. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

[23] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro und Y. Zhang. „Sparks of Artificial General Intelligence: Early experiments with GPT-4“. In: *preprint* arxiv:2303.12712 (2023). DOI: 10.48550/arXiv.2303.12712.

[24] T. Bures, D. Weyns, B. Schmer, E. Tovar, E. Boden, T. Gabor, I. Gerostathopoulos, P. Gupta, E. Kang, A. Knauss et al. „Software engineering for smart cyber-physical systems: Challenges and promising solutions“. In: *ACM SIGSOFT Software Engineering Notes* 42.2 (2017), S. 19–24. DOI: 10.1145/3089649.3089656.

[25] M. S. Burtsev. „Artificial life meets anthropology: A case of aggression in primitive societies“. In: Lecture Notes in Computer Science 3630 (2005), S. 655–664. DOI: 10.1007/11553090_66.

[26] C. Castelfranchi. „Modelling social action for AI agents“. In: *Artificial Intelligence* 103.1 (1998), S. 157–182. DOI: 10.1016/S0004-3702(98)00056-3.

[27] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic et al. „Software engineering for self-adaptive systems: A research roadmap“. In: *Software engineering for self-adaptive systems*. Springer, 2009, S. 1–26. DOI: 10.1007/978-3-642-02161-9_1.

[28] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg und D. Amodei. „Deep Reinforcement Learning from Human Preferences“. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Bd. 30. 2017, S. 4302–4310. URL: https://dl.acm.org/doi/10.5555/3294996.3295184.

[29] A. Consoli, J. Tweedale und L. Jain. „The Link between Agent Coordination and Cooperation“. In: *Intelligent Information Processing III*. 2007, S. 11–19. DOI: 10.1007/978-0-387-44641-7_2.

[30] A. Dafoe, E. Hughes, Y. Bachrach, T. Collins, K. R. McKee, J. Z. Leibo, K. Larson und T. Graepel. „Open Problems in Cooperative AI“. In: *preprint* arxiv:2012.08630 (2020). DOI: 10.48550/arXiv.2012.08630.

[31] M. Desmond, E. Duesterwald, K. Brimijoin, M. Brachman und Q. Pan. „Semi-Automated Data Labeling“. In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. Bd. 133. 2021, S. 156–169. URL: https://proceedings.mlr.press/v133/desmond21a.html.

[32] S. Devlin und D. Kudenko. „Dynamic Potential-based Reward Shaping“. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012, S. 433–440. URL: https://dl.acm.org/doi/10.5555/2343576.2343638.

[33] S. Devlin, L. Yliniemi, D. Kudenko und K. Tumer. „Potential-based Difference Rewards for Multiagent Reinforcement Learning“. In: *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*. 2014, S. 165–172. URL: https://dl.acm.org/doi/10.5555/2615731.2615761.

[34] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu und P. Zhokhov. „OpenAI Baselines“. In: *GitHub repository* (2017). URL: ttps://github.com/openai/baselines.

[35] G. Di Marzo Serugendo, M.-P. Gleizes und A. Karageorgos. „Self-organization in multi-agent systems“. In: *The Knowledge Engineering Review* 20.2 (2005), S. 165–189. DOI: 10.1017/S0269888905000494.

[36] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal und T. Hester. „Challenges of real-world reinforcement learning: definitions, benchmarks and analysis“. In: *Machine Learning* 110.9 (Sep. 2021), S. 2419–2468. DOI: 10.1007/s10994-021-05961-4.

[37] W. Eddy. *Transmission Control Protocol (TCP)*. Aug. 2022. DOI: 10.17487/RFC9293.

[38] E. A. Emerson und J. Y. Halpern. „SSometimesänd "not never"revisited: on branching versus linear time (preliminary report)“. In: *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. 1983, S. 127–140. DOI: 10.1145/567067.567081.

[39] G. Fainekos, B. Hoxha und S. Sankaranarayanan. „Robustness of Specifications and Its Applications to Falsification, Parameter Mining, and Runtime Monitoring with S-TaLiRo“. In: *Proceedings of the International Conference on Runtime Verification (RV)*. 2019, S. 27–47. DOI: 10.1007/978-3-030-32079-9_3.

[40] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli und S. Whiteson. „Counterfactual multi-agent policy gradients“. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. Feb. 2018, S. 2974–2982. URL: https://dl.acm.org/doi/10.5555/3504035.3504398.

[41] X. Franch, L. López, C. Cares und D. Colomer. „The i* Framework for Goal-Oriented Modeling“. In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*. Springer, 2016, S. 485–506. DOI: 10.1007/978-3-319-39417-6_22.

[42] T. Gabor, A. Sedlmeier, T. Phan, F. Ritz, M. Kiermeier, L. Belzner, B. Kempter, C. Klein, H. Sauer, R. Schmid et al. „The scenario coevolution paradigm: adaptive quality assurance for adaptive systems“. In: *International Journal on Software Tools for Technology Transfer* 22 (2020), S. 457–476. DOI: 10.1007/s10009-020-00560-5.

[43] I. Gabriel. „Artificial Intelligence, Values, and Alignment“. In: *Minds and Machines* 30.3 (Sep. 2020), S. 411–437. DOI: 10.1007/s11023-020-09539-2.

[44] J. García und F. Fernández. „A Comprehensive Survey on Safe Reinforcement Learning“. In: *Journal of Machine Learning Research* 16.42 (2015), S. 1437–1480. URL: https://dl.acm.org/doi/10.5555/2789272.2886795.

[45] D. Garlan, B. Schmerl und S.-W. Cheng. „Software Architecture Based Self Adaptation“. In: *Autonomic Computing and Networking*. Springer, 2009, S. 31–55. DOI: 10.1007/978-0-387-89828-5_2.

[46] K. Geihs. „Selbst-adaptive Software“. In: *Informatik-Spektrum* 31 (2008), S. 133–145. DOI: 10.1007/s00287-007-0198-9.

[47] J.-P. Georgé, B. Edmonds und P. Glize. „Making Self-Organising Adaptive Multiagent Systems Work“. In: *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Springer, 2004, S. 321–340. DOI: 10.1007/1-4020-8058-1_20.

[48] G. Giray. „A software engineering perspective on engineering machine learning systems: State of the art and challenges“. In: *Journal of Systems and Software* 180 (2021), S. 111031. DOI: 10.1016/j.jss.2021.111031.

[49] I. Grondman, L. Busoniu, G. A. D. Lopes und R. Babuska. „A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), S. 1291–1307. DOI: 10.1109/TSMCC.2012.2218595.

[50] M. Grześ. „Reward Shaping in Episodic Reinforcement Learning“. In: *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2017, S. 565–573. URL: https://dl.acm.org/doi/10.5555/3091125.3091208.

[51] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang und A. Knoll. „A Review of Safe Reinforcement Learning: Methods, Theory and Applications“. In: *preprint* arxiv:2205.10330 (2023). DOI: 10.48550/arXiv.2205.10330.

[52] D. Ha und J. Schmidhuber. „Recurrent World Models Facilitate Policy Evolution“. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018, S. 2451–2463. URL: https://dl.acm.org/doi/10.5555/3327144.3327171.

[53] D. Hafner, T. Lillicrap, J. Ba und M. Norouzi. „Dream to Control: Learning Behaviors by Latent Imagination“. In: *International Conference on Learning Representations (ICLR*. 2020. URL: https://openreview.net/forum?id=S1lOTC4tDS.

[54] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee und J. Davidson. „Learning Latent Dynamics for Planning from Pixels“. In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, S. 2555–2565. URL: https://proceedings.mlr.press/v97/hafner19a.html.

[55] C. Hahn, T. Phan, S. Feld, C. Roch, F. Ritz, A. Sedlmeier, T. Gabor und C. Linnhoff-Popien. „Nash Equilibria in Multi-Agent Swarms“. In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART)*. Feb. 2020, S. 234–241. DOI: 10.5220/0008990802340241.

[56] C. Hahn, T. Phan, T. Gabor, L. Belzner und C. Linnhoff-Popien. „Emergent escape-based flocking behavior using multi-agent reinforcement learning“. In: *Proceedings of the 2019 Conference on Artificial Life (ALIFE)*. Juli 2019, S. 598–605. DOI: 10.1162/isal_a_00226.

[57] G. Hardin. „The Tragedy of the Commons“. In: *Science* 162.3859 (1968), S. 1243–1248. DOI: 10.1126/science.162.3859.1243.

[58] X. He, K. Zhao und X. Chu. „AutoML: A survey of the state-of-the-art“. In: *Knowledge-Based Systems* 212 (2021). DOI: 10.1016/j.knosys.2020.106622.

[59] D. Hendrycks, N. Carlini, J. Schulman und J. Steinhardt. „Unsolved Problems in ML Safety“. In: *preprint* arxiv:2109.13916 (2021). URL: 10.48550/arXiv.2109.13916.

[60] D. Hernandez und T. B. Brown. „Measuring the Algorithmic Efficiency of Neural Networks“. In: *preprint* arxiv:2005.04305 (2020). DOI: 10.48550/arXiv.2005.04305.

[61] S. Hochreiter und J. Schmidhuber. „Long short-term memory“. In: *Neural computation* 9.8 (1997), S. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[62] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de las Casas, L. A. Hendricks, J. Welbl et al. „An empirical analysis of compute-optimal large language model training“. In: *Advances in Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=iBBcRUlOAPR.

[63] M. Hölzl, N. Koch, M. Puviani, M. Wirsing und F. Zambonelli. „The Ensemble Development Life Cycle and Best Practices for Collective Autonomic Systems“. In: *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. Springer, 2015, S. 325–354. DOI: 10.1007/978-3-319-16310-9_9.

[64] M. Hölzl, N. Koch, M. Puviani, M. Wirsing und F. Zambonelli. „The Ensemble Development Life Cycle and Best Practices for Collective Autonomic Systems“. In: *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. 2015, S. 325–354. DOI: 10.1007/978-3-319-16310-9_9.

[65] M. Hüttenrauch, A. Šošić und G. Neumann. „Deep Reinforcement Learning for Swarm Systems“. In: *Journal of Machine Learning Research* 20.54 (2019), S. 1–31. URL: https://dl.acm.org/doi/10.5555/3322706.3361995.

[66] M. Hüttenrauch, A. Šošić und G. Neumann. „Guided Deep Reinforcement Learning for Swarm Systems“. In: *arXiv preprint* abs/1709.06011 (2017). DOI: 10.48550/arXiv.1709.06011.

[67] IEEE. „IEEE Standard for System and Software Verification and Validation“. In: *Std. 1012-2012* (2012), S. 1–223. DOI: 10.1109/IEEESTD.2012.6204026.

[68] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman et al. „Human-level performance in 3D multiplayer games with population-based reinforcement learning“. In: *Science* 364.6443 (2019), S. 859–865. DOI: 10.1126/science.aau6249.

[69] L. P. Kaelbling und A. R. Littman Michael L. Cassandra. „Planning and acting in partially observable stochastic domains“. In: *Artificial Intelligence* 101.1 (1998), S. 99–134. DOI: 10.1016/S0004-3702(98)00023-X.

[70] U. Kannengiesser, R. Heininger, L. Billy, P. Terpak, M. Neubauer, C. Stary, D. Majoe, A. Totter und D. Bonaldi. „Lot-Size One Production“. In: *S-BPM in the Production Industry: A Stakeholder Approach*. Springer, 2017, S. 69–111. DOI: 10.1007/978-3-319-48466-2_4.

[71] S. Katoch, S. S. Chauhan und V. Kumar. „A review on genetic algorithm: past, present, and future“. In: *Multimedia Tools and Applications* 80.5 (Okt. 2020), S. 8091–8126. DOI: 10.1007/s11042-020-10139-6.

[72] M. M. Khan, K. Kasmarik und M. Barlow. „Toward Computational Motivation for Multi-Agent Systems and Swarms“. In: *Frontiers in Robotics and AI* 5 (2018), S. 134. DOI: 10.3389/frobt.2018.00134.

[73] R. Kirk, A. Zhang, E. Grefenstette und T. Rocktäschel. „A Survey of Zero-Shot Generalisation in Deep Reinforcement Learning“. In: *Journal of Artificial Intelligence Research* 76 (2023). DOI: 10.1613/jair.1.14174.

[74] V. Krakovna, L. Orseau, R. Ngo, M. Martic und S. Legg. „Avoiding Side Effects by Considering Future Tasks“. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS)*. 2020. DOI: 10.5555/3495724.3497324.

[75] D. Kreuzberger, N. Kühl und S. Hirschl. „Machine Learning Operations (MLOps): Overview, Definition, and Architecture“. In: *IEEE Access* 11 (2023), S. 31866–31879. DOI: 10.1109/ACCESS.2023.3262138.

[76] P. Kruchten. *The Rational Unified Process–An Introduction*. Jan. 2000, S. 255. ISBN: 0321197704. URL: https://dl.acm.org/doi/10.5555/940320.

[77] J. Z. Leibo, J. Perolat, E. Hughes, S. Wheelwright, A. H. Marblestone, E. Duéñez-Guzmán, P. Sunehag, I. Dunning und T. Graepel. „Malthusian Reinforcement Learning“. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2019, S. 1099–1107. URL: https://dl.acm.org/doi/10.5555/3306127.3331809.

[78] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki und T. Graepel. „Multi-Agent Reinforcement Learning in Sequential Social Dilemmas“. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2017, S. 464–473. URL: https://dl.acm.org/doi/abs/10.5555/3091125.3091194.

[79] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini und S. Legg. „Scalable agent alignment via reward modeling: a research direction“. In: *preprint* arxiv:1811.07871 (2018). DOI: 10.48550/arXiv.1811.07871.

[80] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau und S. Legg. „AI Safety Gridworlds“. In: *preprint* arxiv:1711.0988 (2017). DOI: 10.48550/arXiv.1711.09883.

[81] L. Leite, C. Rocha, F. Kon, D. Milojicic und P. Meirelles. „A Survey of DevOps Concepts and Challenges“. In: *ACM Computing Surveys* 52.6 (Nov. 2019). DOI: 10.1145/3359981.

[82] Z. Li, C. H. Sim und M. Y. Hean Low. „A Survey of Emergent Behavior and Its Impacts in Agent-based Systems“. In: *2006 4th IEEE International Conference on Industrial Informatics*. Aug. 2006, S. 1295–1300. DOI: 10.1109/INDIN.2006.275846.

[83] S. Liaskos, S. A. McIlraith, S. Sohrabi und J. Mylopoulos. „Integrating Preferences into Goal Models for Requirements Engineering“. In: *2010 18th IEEE International Requirements Engineering Conference*. 2010, S. 135–144. DOI: 10.1109/RE.2010.26.

[84] P. Linardatos, V. Papastefanopoulos und S. Kotsiantis. „Explainable AI: A Review of Machine Learning Interpretability Methods“. In: *Entropy* 23.1 (2021). DOI: 10.3390/e23010018.

[85] S. Liu, G. Lever, N. Heess, J. Merel, S. Tunyasuvunakool und T. Graepel. „Emergent Coordination Through Competition“. In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://openreview.net/forum?id=BkG8sjR5Km.

[86] W. F. Lloyd. „W. F. Lloyd on the Checks to Population“. In: *Population and Development Review* 6.3 (1980), S. 473–496. DOI: 10.2307/1972412.

[87] A. J. Lotka. *Analytical Theory of Biological Populations*. 1. Aufl. Springer, 1998. DOI: 10.1007/978-1-4757-9176-1.

[88] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel und I. Mordatch. „Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments“. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Bd. 30. 2017, S. 6379–6390. URL: https://dl.acm.org/doi/10.5555/3295222.3295385.

[89] R. Luna und K. E. Bekris. „Efficient and complete centralized multi-robot path planning“. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, S. 3268–3275. DOI: 10.1109/IROS.2011.6095085.

[90] M. J. Lutz, C. R. Reid, C. J. Lustri, A. B. Kao, S. Garnier und I. D. Couzin. „Individual error correction drives responsive self-assembly of army ant scaffolds“. In: *Proceedings of the National Academy of Sciences* 118.17 (2021), e2013741118. DOI: 10.1073/pnas.2013741118.

[91] L. E. Lwakatare, I. Crnkovic und J. Bosch. „DevOps for AI–Challenges in Development of AI-enabled Applications“. In: *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. Sep. 2020, S. 1–6. DOI: 10.23919/SoftCOM50211.2020.9238323.

[92] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson und I. Crnkovic. „A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation“. In: *Agile Processes in Software Engineering and Extreme Programming*. Springer, 2019, S. 227–243. DOI: 10.1007/978-3-030-19034-7_14.

[93] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer und S. Wagner. „Software Engineering for AI-Based Systems: A Survey“. In: *ACM Transactions on Software Engineering and Methodology* 31.2 (Apr. 2022). DOI: 10.1145/3487043.

[94] P. McKinley, S. Sadjadi, E. Kasten und B. Cheng. „Composing adaptive software“. In: *Computer* 37.7 (2004), S. 56–64. DOI: 10.1109/MC.2004.48.

[95] T. M. Mitchell. *Machine Learning*. 1. Aufl. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077. URL: https://www.cs.cmu.edu/~tom/mlbook.html.

[96] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver und K. Kavukcuoglu. „Asynchronous Methods for Deep Reinforcement Learning“. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML)*. 2016, S. 1928–1937. URL: https://dl.acm.org/doi/10.5555/3045390.3045594.

[97] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra und M. Riedmiller. „Playing Atari with Deep Reinforcement Learning“. In: *arXiv preprint* abs/1312.5602 (2013). DOI: 10.48550/arXiv.1312.5602.

[98] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al. „Human-level control through deep reinforcement learning“. In: *Nature* 518.7540 (2015), S. 529–533. DOI: 10.1038/nature14236.

[99] A. Ng, D. Harada und S. Russell. „Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping“. In: *Proceedings of the Sixteenth International Conference on Machine Learning (ICML)*. 1999, S. 278–287. URL: https://dl.acm.org/doi/10.5555/645528.657613.

[100] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike und R. Lowe. „Training language models to follow instructions with human feedback“. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Bd. 35. 2022, S. 27730–27744. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

[101] A. Pappagallo, A. Massini und E. Tronci. „Monte Carlo Based Statistical Model Checking of Cyber-Physical Systems: A Review“. In: *Information* 11.12 (2020). DOI: 10.3390/info11120588.

[102] J. K. Parrish, S. V. Viscido und D. Grünbaum. „Self-Organized Fish Schools: An Examination of Emergent Properties“. In: *Biological Bulletin* 202.3 (2002), S. 296–305. DOI: 10.2307/1543482.

[103] E. Pennisi. „How did cooperative behavior evolve?“ In: *Science* 309.5731 (2005), S. 93–93. DOI: 10.1126/science.309.5731.93.

[104] J. Perolat, J. Z. Leibo, V. Zambaldi, C. Beattie, K. Tuyls und T. Graepel. „A multi-agent reinforcement learning model of common-pool resource appropriation“. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, S. 3643–3652. URL: https://dl.acm.org/doi/10.5555/3294996.3295122.

[105] D. Phan, J. Yang, M. Clark, R. Grosu, J. Schierman, S. Smolka und S. Stoller. „A Component-Based Simplex Architecture for High-Assurance Cyber-Physical Systems“. In: *17th International Conference on Application of Concurrency to System Design (ACSD)*. 2017, S. 49–58. DOI: 10.1109/ACSD.2017.23.

[106] D. T. Phan, R. Grosu, N. Jansen, N. Paoletti, S. A. Smolka und S. D. Stoller. „Neural Simplex Architecture“. In: *NASA Formal Methods*. 2020, S. 97–114. DOI: 10.1007/978-3-030-55754-6_6.

[107] T. Phan, L. Belzner, T. Gabor und K. Schmid. „Leveraging Statistical Multi-Agent Online Planning with Emergent Value Function Approximation“. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018, S. 730–738. URL: https://dl.acm.org/doi/10.5555/3237383.3237491.

[108] T. Phan, L. Belzner, T. Gabor, A. Sedlmeier, F. Ritz und C. Linnhoff-Popien. „Resilient Multi-Agent Reinforcement Learning with Adversarial Value Decomposition“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Mai 2021, S. 11308–11316. DOI: 10.1609/aaai.v35i13.17348.

[109] T. Phan, T. Gabor, A. Sedlmeier, R. Fabian, B. Kempter, C. Klein, H. Sauer, R. Schmid, J. Wieghardt, M. Zeller und C. Linnhoff-Popien. „Learning and Testing Resilience in Cooperative Multi-Agent Systems“. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. Mai 2020, S. 1055–1063. URL: https://dl.acm.org/doi/10.5555/3398761.3398884.

[110] T. Phan, F. Ritz, P. Altmann, M. Zorn, J. Nüßlein, M. Kölle, T. Gabor und C. Linnhoff-Popien. „Attention-based recurrence for multi-agent reinforcement learning under stochastic partial observability“. In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. 2023. DOI: 10.5555/3618408.3619565.

[111] T. Phan, F. Ritz, L. Belzner, P. Altmann, T. Gabor und C. Linnhoff-Popien. „VAST: Value Function Factorization with Variable Agent Sub-Teams“. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021, S. 24018–24032. URL: https://openreview.net/forum?id=hyJKKIhfxxT.

[112] T. Phan, F. Sommer, P. Altmann, F. Ritz, L. Belzner und C. Linnhoff-Popien. „Emergent Cooperation from Mutual Acknowledgment Exchange“. In: *Proceedings of the 21st International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. Mai 2022, S. 1047–1055. URL: https://dl.acm.org/doi/abs/10.5555/3535850.3535967.

[113] G. Picard, J. F. Hübner, O. Boissier und M.-P. Gleizes. „Reorganisation and Self-organisation in Multi-Agent Systems“. In: *International Workshop on Organizational Modeling (OrgMod 2009)*. Juni 2009, S. 66–80. URL: https://hal-emse.ccsd.cnrs.fr/emse-00675584.

[114] R. Pinsler, M. Maag, O. Arenz und G. Neumann. „Inverse Reinforcement Learning of Bird Flocking Behavior“. In: *ICRA 2018 Workshop on Swarms: From Biology to Robotics and Back*. 2018. URL: https://www.ias.informatik.tu-darmstadt.de/uploads/Team/OlegArenz/PinslerEtAl_ICRA2018swarms.pdf.

[115] M. S. Rahman, E. Rivera, F. Khomh, Y. Guéhéneuc und B. Lehnert. „Machine Learning Software Engineering in Practice: An Industrial Case Study“. In: *preprint* arXiv:1906.07154 (2019). DOI: 10.48550/arXiv.1906.07154.

[116] V. Rao und D. Bernstein. „Naive control of the double integrator“. In: *IEEE Control Systems Magazine* 21.5 (2001), S. 86–97. DOI: 10.1109/37.954521.

[117] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster und S. Whiteson. „Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning“. In: *Journal of Machine Learning Research* 21.178 (2020), S. 1–51. URL: http://jmlr.org/papers/v21/20-081.html.

[118] C. W. Reynolds. „Flocks, Herds and Schools: A Distributed Behavioral Model“. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), S. 25–34. DOI: 10.1145/37402.37406.

[119] M. T. Ribeiro, S. Singh und C. Guestrin. „“Why Should I Trust You?”: Explaining the Predictions of Any Classifier“. In: *Proceedings of the 22nd ACM SIGKDD*. 2016, S. 1135–1144. DOI: 10.1145/2939672.2939778.

[120] F. Ritz, F. Hohnstein, R. Müller, T. Phan, T. Gabor, C. Hahn und C. Linnhoff-Popien. „Towards Ecosystem Management from Greedy Reinforcement Learning in a Predator-Prey Setting“. In: *Proceedings of the 2020 Conference on Artificial Life (ALIFE)*. Juli 2020, S. 518–525. DOI: 10.1162/isal_a_00273.

[121] F. Ritz, T. Phan, R. Müller, T. Gabor, A. Sedlmeier, M. Zeller, J. Wieghardt, R. Schmid, H. Sauer, C. Klein und C. Linnhoff-Popien. „SAT-MARL: Specification Aware Training in Multi-Agent Reinforcement Learning“. In: *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART)*. Feb. 2021, S. 28–37. DOI: 10.5220/0010189500280037.

[122] F. Ritz, T. Phan, R. Müller, T. Gabor, A. Sedlmeier, M. Zeller, J. Wieghardt, R. Schmid, H. Sauer, C. Klein und C. Linnhoff-Popien. „Specification Aware Multi-Agent Reinforcement Learning“. In: *Agents and Artificial Intelligene: ICAART 2021. Revised Selected Papers*. Lecture Notes in Computer Science 13251 (2022), S. 3–21. DOI: 10.1007/978-3-031-10161-8_1.

[123] F. Ritz, T. Phan, A. Sedlmeier, P. Altmann, J. Wieghardt, R. Schmid, H. Sauer, C. Klein, C. Linnhoff-Popien und T. Gabor. „Capturing Dependencies Within Machine Learning via a Formal Process Model“. In: Lecture Notes in Computer Science 13703 (2022), S. 249–265. DOI: 10.1007/978-3-031-19759-8_16.

[124] F. Ritz, D. Ratke, T. Phan, L. Belzner und C. Linnhoff-Popien. „A Sustainable Ecosystem through Emergent Cooperation in Multi-Agent Reinforcement Learning“. In: *Proceedings of the 2021 Conference on Artificial Life (ALIFE)*. Juli 2021, S. 74–84. DOI: 10.1162/isal_a_00399.

[125] A. Rodriguez, R. Parr und D. Koller. „Reinforcement learning using approximate belief states“. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS)*. 1999, S. 1036–1042. URL: https://dl.acm.org/doi/abs/10.5555/3009657.3009803.

[126] Á. L. Rodríguez, B. Cheeran, G. Koch, T. Hortobágyi und M. Fernandez-del-Olmo. „The role of mirror neurons in observational motor learning: an integrative review“. In: *European Journal of Human Movement* 32 (2014), S. 82–103. URL: https://www.eurjhm.com/index.php/eurjhm/article/view/320.

[127] C. Rudin. „Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead“. In: *Nature Machine Intelligence* 1.5 (Mai 2019), S. 206–215. DOI: 10.1038/s42256-019-0048-x.

[128] S. Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Allen Lane, 2019. ISBN: 9780241335208. URL: https://books.google.de/books?id=VMq_wwEACAAJ.

[129] M. Salimibeni, A. Mohammadi, P. Malekzadeh und K. N. Plataniotis. „Multi-Agent Reinforcement Learning via Adaptive Kalman Temporal Difference and Successor Representation“. In: *Sensors* 22.4 (2022). DOI: 10.3390/s22041393.

[130] I. H. Sarker. „Machine Learning: Algorithms, Real-World Applications and Research Directions“. In: *SN Computer Science* 2.3 (März 2021). DOI: 10.1007/s42979-021-00592-x.

[131] J. Schulman, P. Moritz, S. Levine, M. Jordan und P. Abbeel. „High-Dimensional Continuous Control Using Generalized Advantage Estimation“. In: *4th International Conference on Learning Representations (ICLR) Poster*. 2016. DOI: 10.48550/arXiv.1506.02438.

[132] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov. „Proximal policy optimization algorithms“. In: *preprint* arxiv:1707.06347 (2017). DOI: 10.48550/arXiv.1707.06347.

[133] W. Scott. „Organizations: Rational, Natural and Open Systems“. In: *Canadian Journal of Sociology / Cahiers canadiens de sociologie* 29 (Jan. 1998). DOI: 10.2307/2393090.

[134] M. Seurin, P. Preux und O. Pietquin. „I'm Sorry Dave, I'm Afraid I Can't Do That: Deep Q-Learning from Forbidden Actions“. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, S. 1–8. DOI: 10.1109/IJCNN48605.2020.9207496.

[135] G. Sharon, R. Stern, A. Felner und N. R. Sturtevant. „Conflict-based search for optimal multi-agent pathfinding“. In: *Artificial Intelligence* 219 (2015), S. 40–66. DOI: 10.1016/j.artint.2014.11.006.

[136] O. M. Shehory, K. Sycara und S. Jha. „Multi-agent coordination through coalition formation“. In: *Intelligent Agents IV. Agent Theories, Architectures, and Languages. ATAL 1997*. Lecture Notes in Computer Science 1365 (1998), S. 143–154. DOI: 10.1007/BFb0026756.

[137] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan und D. Hassabis. „A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play“. In: *Science* 362.6419 (2018), S. 1140–1144. DOI: 10.1126/science.aar6404.

[138] D. Sinreich. „An architectural blueprint for autonomic computing“. In: *IBM Whitepaper* (2006). URL: https://citeseerx.ist.psu.edu/document?doi=0e99837d9b1e70bb35d516e32ecfc345cd30e795.

[139] K. Son, D. Kim, W. J. Kang, D. E. Hostallero und Y. Yi. „QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning“. In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Bd. 97. Juni 2019, S. 5887–5896. URL: https://proceedings.mlr.press/v97/son19a.html.

[140] L. Steels. „When are robots intelligent autonomous agents?“ In: *Robotics and Autonomous Systems* 15.1 (1995), S. 3–9. DOI: 10.1016/0921-8890(95)00011-4.

[141] S. Studer, T. B. Bui, C. Drescher, A. Hanuschkin, L. Winkler, S. Peters und K.-R. Müller. „Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology“. In: *Machine Learning and Knowledge Extraction* 3.2 (2021), S. 392–413. DOI: 10.3390/make3020020.

[142] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls und T. Graepel. „Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward“. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018, S. 2085–2087. URL: https://dl.acm.org/doi/10.5555/3237383.3238080.

[143] P. Sunehag, G. Lever, S. Liu, J. Merel, N. Heess, J. Z. Leibo, E. Hughes, T. Eccles und T. Graepel. „Reinforcement Learning Agents acquire Flocking and Symbiotic Behaviour in Simulated Ecosystems“. In: *ALIFE 2019: The 2019 Conference on Artificial Life*. Juli 2019, S. 103–110. DOI: 10.1162/isal_a_00148.

[144] R. S. Sutton, D. McAllester, S. Singh und Y. Mansour. „Policy Gradient Methods for Reinforcement Learning with Function Approximation“. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Bd. 12. 1999. URL: https://dl.acm.org/doi/10.5555/3009657.3009806.

[145] R. S. Sutton und A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249. URL: http://incompleteideas.net/book/the-book.html.

[146] S. Tanabe und N. Masuda. „Evolution of cooperation facilitated by reinforcement learning with adaptive aspiration levels“. In: *Journal of Theoretical Biology* 293 (2012), S. 151–160. DOI: 10.1016/j.jtbi.2011.10.020.

[147] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale et al. „Llama 2: Open Foundation and Fine-Tuned Chat Models“. In: *preprint* arxiv:2307.09288 (2023). DOI: 10.48550/arXiv.2307.09288.

[148] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser und I. Polosukhin. „Attention is all you need“. In: *Advances in neural information processing systems (NeurIPS)*. 2017, S. 5998–6008. URL: https://dl.acm.org/doi/10.5555/3295222.3295349.

[149] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev et al. „Grandmaster level in StarCraft II using multi-agent reinforcement learning“. In: *Nature* 575.7782 (2019), S. 350–354. DOI: 10.1038/s41586-019-1724-z.

[150] O. Vinyals, M. Fortunato und N. Jaitly. „Pointer networks“. In: *Advances in neural information processing systems (NeurIPS)*. 2015, S. 2692–2700. URL: https://dl.acm.org/doi/10.5555/2969442.2969540.

[151] L. Wang, Q. Cai, Z. Yang und Z. Wang. „Neural Policy Gradient Methods: Global Optimality and Rates of Convergence“. In: *International Conference on Learning Representations (ICLR)*. 2020. URL: https://openreview.net/forum?id=BJgQfkSYDS.

[152] S. Wang, J. Wan, D. Zhang, D. Li und C. Zhang. „Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination“. In: *Computer Networks* 101 (2016), S. 158–168. DOI: https://doi.org/10.1016/j.comnet.2015.12.017.

[153] Y. Watanabe, H. Washizaki, K. Sakamoto, D. Saito, K. Honda, N. Tsuda, Y. Fukazawa und N. Yoshioka. „Preliminary Literature Review of Machine Learning System Development Practices“. In: *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2021, S. 1407–1408. DOI: 10.1109/COMPSAC51774.2021.00207.

[154] C. J. C. H. Watkins und P. Dayan. „Q-learning“. In: *Machine Learning* 8.3 (Mai 1992), S. 279–292. DOI: 10.1007/BF00992698.

[155] R. J. Williams. „Simple statistical gradient-following algorithms for connectionist reinforcement learning“. In: *Machine Learning* 8.3 (Mai 1992), S. 229–256. DOI: 10.1007/BF00992696.

[156] M. Wirsing und L. Belzner. „Towards Systematically Engineering Autonomous Systems Using Reinforcement Learning and Planning“. In: *Analysis, Verification and Transformation for Declarative Programming and Intelligent Systems*. Springer, 2023, S. 281–306. DOI: 10.1007/978-3-031-31476-6_16.

[157] M. Wirsing, M. M. Hölzl, N. Koch und P. Mayer, Hrsg. *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. Bd. 8998. Lecture Notes in Computer Science. Springer, 2015. DOI: https://doi.org/10.1007/978-3-319-16310-9.

[158] C. S. de Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. S. Torr, M. Sun und S. Whiteson. „Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge?“ In: *preprint* arxiv:2011.09533 (2020). DOI: 10.48550/arXiv.2011.09533.

[159] D. H. Wolpert und K. Tumer. „Optimal Payoff Functions for Members of Collectives“. In: *Advances in Complex Systems* 04.02n03 (2001). DOI: 10.1142/s0219525901000188.

[160] M. Wooldridge und N. R. Jennings. „Intelligent agents: theory and practice“. In: *The Knowledge Engineering Review* 10.2 (1995), S. 115–152. DOI: 10.1017/S0269888900008122.

[161] Y. Yang, L. Yu, Y. Bai, Y. Wen, W. Zhang und J. Wang. „A Study of AI Population Dynamics with Million-Agent Reinforcement Learning“. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. Juli 2018, S. 2133–2135. URL: https://dl.acm.org/doi/10.5555/3237383.3238096.

[162] D. Ye, M. Zhang und A. V. Vasilakos. „A Survey of Self-Organization Mechanisms in Multiagent Systems“. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.3 (2017), S. 441–461. DOI: 10.1109/TSMC.2015.2504350.

[163] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz und S. Mannor. „Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning“. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, S. 3562–3573. URL: https://dl.acm.org/doi/10.5555/3327144.3327274.

[164] C. Zhang, S. Abdallah und V. Lesser. „Integrating Organizational Control into Multi-Agent Learning“. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009, S. 757–764. URL: https://dl.acm.org/doi/10.5555/1558109.1558116.

[165] C. Zhang, V. Lesser und S. Abdallah. „Self-Organization for Coordinating Decentralized Reinforcement Learning“. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2010, S. 739–746. URL: https://dl.acm.org/doi/10.5555/1838206.1838304.