Supporting Architectural Decision Making on Training Strategies in Reinforcement Learning Architectures

Evangelos Ntentos, Stephen John Warnett, Uwe Zdun
Faculty of Computer Science, Research Group Software Architecture
University of Vienna
Vienna, Austria
firstname.lastname@univie.ac.at

***Abstract*—In the dynamic landscape of artificial intelligence and machine learning, Reinforcement Learning (RL) has emerged as a powerful paradigm for training intelligent agents in sequential decision-making. As RL architectures progress in complexity, the need for informed decision-making regarding training strategies and related consequences on the software architecture becomes increasingly intricate. This work addresses this challenge by presenting the outcomes of a qualitative, in-depth study focused on best practices and patterns within training strategies for RL architectures, as articulated by practitioners. Leveraging a model-based qualitative research method, we introduce a formal architecture decision model to bridge the gap between scientific insights and practical implementation. We aim to enhance the understanding of practitioners' approaches in RL architecture. The paper analyzes 33 knowledge sources to discern established industrial practices, patterns, relationships, and decision drivers. Based on this knowledge, we introduce a formal Architectural Design Decision (ADD) model, encapsulating 6 decisions, 29 decision options, and 19 decision drivers, providing robust decision-making support for this critical facet of RL-based software architectures.**

***Index Terms*—Machine Learning, Reinforcement Learning, Grounded Theory, Software Architecture, Design Decisions**

I. Introduction

In the fast-evolving realm of artificial intelligence and machine learning, Reinforcement Learning (RL) has emerged as a potent paradigm, enabling the training of intelligent agents to make sequential decisions [1], [2]. As RL architectures progress in complexity and scale, making pivotal decisions regarding training strategies and related consequences on the software architecture becomes progressively intricate. Several authors have attempted to capture patterns and best practices in training strategies within reinforcement learning architectures [3], [4], [5], [6]. However, these works predominantly concentrate on applying published patterns or scientific findings. In contrast, we find prevalent industry practices predominantly in grey literature, such as blogs, experience reports, and system documentation. While these sources offer insights into existing practices, they often need more systematic architectural guidance. The reported practices exhibit variation and reliance on personal experiences, contributing to uncertainty and risk in RL design. Addressing this necessitates extensive experience or a comprehensive study of knowledge sources. We aim to present a more comprehensive and consistent perspective on current industrial practices, complementing existing knowledge.

To achieve this, we conducted an in-depth qualitative study of RL descriptions provided by practitioners, extracting informal information about established practices and patterns in RL training strategies. Employing a model-based qualitative research method[7], we systematically analyzed practitioner sources using coding and constant comparison methods [8], followed by precise software modeling. This rigorous process enabled the development of a detailed software model illustrating established practices, patterns, and their interrelationships.

This paper aims to study the following research questions:

- **RQ1** Which patterns and practices do practitioners currently use for supporting training strategies in RL architectures?
- **RQ2** What are the relations between existing training patterns and practices? Specifically, which Architectural Design Decisions (ADDs) are pertinent when designing RL systems?
- **RQ3** What are the influencing factors (i.e., decision drivers) in architecting RL systems in the eye of the practitioner today?

This paper makes three primary contributions. Firstly, we conducted a qualitative study on reinforcement learning architectures, analyzing 33 knowledge sources to discern established industrial practices, patterns, relationships, and decision drivers. Secondly, we developed a formal Architectural Design Decision (ADD) model, encapsulating 6 decisions, 29 decision options, and 19 decision drivers.

This paper is organized as follows: In Section II, we describe RL and its role in software architecture. Section III compares our work with related studies. Section IV presents the research methods applied in our study and summarizes the knowledge sources. Section V details our reusable ADD model on RL architectures. Section VI evaluates and Section VII discusses our results. Subsequently, Section VIII considers potential threats to the validity of our study, and Section IX summarizes our findings.

II. Background: RL and its Role in Software Architecture

RL, a paradigm for training intelligent agents via trial and error in sequential decision-making, diverges from supervised learning by enabling agents to learn optimal strategies through interactions with an environment and receiving feedback as rewards or penalties. This process involves formulating the problem, defining state and action spaces, and establishing a reward structure. Environment design is crucial, as it shapes decision-making, actions, and feedback. The RL training pipeline encompasses environment setup, policy definition, algorithmic training, hyperparameter tuning, and policy evaluation, emphasizing an iterative approach for continuous performance enhancement via parameter adjustments and repeated training iterations.

Incorporating RL into software architecture is pivotal, as RL training strategy decisions significantly influence system design. For instance, choosing between single-agent and multi-agent RL models impacts system design by shaping communication channels and coordination mechanisms. Similarly, the use of model checkpoints and multiple model versions affects architecture, requiring a checkpoint management system that influences storage, retrieval, and deployment processes.

III. Related Work

In this section, we provide details on related works and compare them. We discuss related studies for RL patterns, practices, and approaches for decision documentation.

Several approaches that study RL patterns and practices exist: Lee et al. [3] discuss the evolution of RL algorithms, progressing from single-agent to multi-agent systems. The focus is on a distributed optimization perspective. Canese et al. [4] delineate multi-agent algorithms, comparing them based on critical characteristics for multi-agent reinforcement learning applications, including non-stationarity, scalability, and observability. The description also encompasses the most prevalent benchmark environments utilized to assess the performance of the discussed methods. Zhu et al. [5] present a framework for classifying state-of-the-art transfer learning approaches, through which they scrutinize their objectives, methodologies, compatibility with reinforcement learning backbones, and practical applications. Additionally, they establish links between transfer learning and other pertinent topics from the reinforcement learning standpoint, delving into potential challenges that await further research progress. Eimer et al. [9] propose adopting best practices from Automated Machine Learning in RL, including separating tuning and testing seeds and employing principled Hyperparameter Optimization (HPO) across a broad search space. While these works focus on specific aspects of RL, our work contributes by conducting a comprehensive qualitative study to extract best practices and patterns within training strategies for RL architectures. Furthermore, the formal ADD model introduced in our work provides a structured framework and a focus on software architecture beyond these related works.

Washizaki et al. [10] present a comprehensive literature review, revealing 15 software engineering design patterns tailored to machine learning applications. The work by Sharma and Bavuluri. [11] involves the identification and analysis of design patterns and architectural patterns in two software applications utilizing Machine Learning (ML) and Deep Learning techniques, respectively. While these works focus on design patterns in machine learning applications, the presented work specifically addresses RL architectures, which are not yet considered in the form of design patterns or ADDs.

Furthermore, numerous approaches exist for decision documentation, ranging from service-oriented solutions [12] and service-based platform integration [13] to considerations of REST vs. SOAP [14] and big data repositories [15]. However, these studies are not focussed on training strategies in RL architectures or similar topics. Warnett and Zdun [16] introduce a Grounded Theory-based approach, delving into practitioners' current understanding and architectural concepts of machine learning solution deployment. They formulate seven ADDs and various relationships, modeling twenty-six decision options and forty-four decision drivers in machine learning deployment. While other authors have combined decision models with formal view models [17], our contribution builds on these techniques by incorporating a formal modeling approach rooted in a qualitative research methodology.

Our study scrutinizes practitioner methods and techniques, aiming to bridge the gap between theory and practice in training strategies in RL. The resulting formal model encompasses ADDs, decision options, practices, drivers, and relationships, providing valuable insights to aid practitioners in making informed training strategy decisions in RL.

IV. Research Method

This section discusses the research method followed in this study and the modeling tool we used to create and visualize the decision model.

A. Grounded Theory

This paper aims to systematically study the established practices for training strategies in RL architectures. We follow the model-based qualitative research method described in [7]. It is based on the established Grounded Theory (GT) [8] research method in combination with methods for studying established practices like pattern mining (see e.g. [18]) and their combination with GT [19]. It involves iterative steps of interpreting data, aiming to construct a theory rooted in the collected data. Data analysis is conducted during data collection rather than after.

Constant comparison is vital in GT, where researchers continuously compare existing data and concepts with new data to identify emerging abstract concepts. These new concepts are then compared with pre-existing ones and the collected data. We organized concepts into categories (or codes) and linked them with properties and relationships, guiding subsequent research iterations.

In iterative cycles, we conducted knowledge-mining procedures, searching for new sources and applying open and axial coding to identify candidate categories. Continuously comparing codes with the evolving model allowed for incremental enhancements. The decision of when to conclude this process is critical in qualitative methods, with theoretical saturation [8] widely accepted as a stopping criterion. In our study, we ceased analysis when an additional seven knowledge sources failed to contribute new insights, demonstrating theoretical saturation. Although this approach was conservative, our study had already achieved convergence after twenty-six knowledge sources. Source selection, detailed in Table I, drew upon our experience with tools, methods, patterns, and practices encountered or studied previously. Our methodology encompassed three coding activities: (1) Open coding involved developing concepts from data with specific questioning, precise coding, and minimal assumptions. (2) Axial coding focused on developing categories and linking data, concepts, categories, and properties. (3) Selective coding aimed to integrate developed categories in a core category.

B. Methodology

In Figure 1, we present an overview of our research method steps followed in this study. To gather practitioner sources, we utilized standard search engines like Google, StartPage, Bing, and DuckDuckGo and topic portals such as InfoQ and DZone. We utilized relevant initial search terms to align with our work's focus, such as "training strategies in RL", "Transfer Learning in RL," "Checkpoints in RL," etc. We have opted against a Multivocal Literature Review (MLR) where scientific sources are included, as our research goals focus on exploring practitioner views specifically. Our initial data are from the keywords. After the coding process, further data sources emerged in terms of new keywords or new referenced sources. The data collection process is repeatable with scanning and skimming techniques. During open and axial coding, we studied each included source line by line in-depth during open coding – for most sources in many iterations. We chose this method over manually browsing selected grey literature initially because it is replicable [20]. We employed GT coding practices and constant iterative comparison to identify concepts, categories, properties, and relationships. We developed our decision model using our Python-based modeling tool, CodeableModels[1].

For subsequent iterations, we conducted searches using relevant terms based on the identified topics from previous iterations, focusing on areas requiring coding and their potential contribution to the model. Practitioner articles needed to be relevant to the topic to be considered candidate sources rather than primarily promotional. The authors reviewed and approved each other's selection of sources for suitability.

Open coding was utilized to translate conceptual details into conceptual labels, while axial coding facilitated the identification of categories based on recurring, synonymous, and related

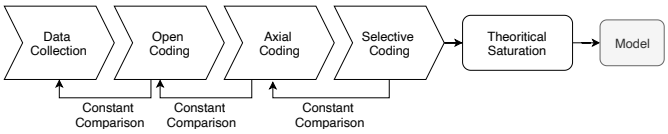

Fig. 1: Research Method Steps

concepts. Each source underwent line-by-line examination during both open and axial coding phases, with our thought processes, conceptual understanding, interpretation, and reasoning documented in memos linked to the respective sources, ensuring the traceability of codes to their origins. Selective coding then focused on extracting the main ideas of the theory, refining previous sources as necessary. Formal UML-based modeling was employed for axial and selective coding, resulting in a precise and consistent theory represented as a formal UML model. Notably, all relevant lines in the knowledge source were coded, providing an initial interpretation through open coding, followed by categorization and formalization via axial coding. Python source code models[2] were generated from the information derived from the sources, facilitating further analysis and refinement. This iterative process, summarized in Table I, culminated in the identification of errors and performance improvements during selective coding, ensuring the accuracy and robustness of our findings.

V. Reusable ADD Model for Training Strategies in Reinforcement Learning Architectures

In this section, we introduce the reusable Architectural Design Decision (ADD) model derived in our study (see the replication package[3]). Figure 2 shows the metamodel for ADD models. A decision has a name, a type, and a description. The metamodel contains the *Decisions* of the ADD model. The decision has a *Context*, which is described by a domain object that denotes the system part or aspect in which the decision is applied. Each *Decision* has *Options*. All *Optios* are *Solutions*. An *Option* has *Forces*, which can have a force impact. Finally, decisions, solutions, and options can have *Relations*. A solution can relate to another solution, but the relation's source or target must be an option (relations such as *is-a, uses, can be combined with, typically realized with*). All solutions in the decision must be linked directly or via other options to a decision. Decisions and options can have next-decision relations, too. Decisions, contexts, solutions (and options), forces, and relations are all named elements, meaning they can have an optional name and an optional description.

The reusable ADD model consists of a single decision *Category*, the *Training Efficiency and Optimization Strategies Category*, which has six top-level decisions, as depicted in Fig. 3. A *Category* can have one or more decisions. All elements of our model are instances of the metamodel, with meta-classes such as *Decision*, *Category*, etc. We denote the meta-classes as stereotypes in the model descriptions below.

[1]https://github.com/uzdun/CodeableModels

[2]See Data/Decisions/Generated Python Models from Sources directory in https://doi.org/10.5281/zenodo.10624377

[3]https://doi.org/10.5281/zenodo.10624377

TABLE I: Knowledge Sources Included in the Study

ID	Description	Reference
S1	What is Model-Based Reinforcement Learning?	https://bit.ly/3MNg3iR
S2	Reinforcement Learning Meets Large Language Models (LLMs): Aligning Human Preferences in LLMs	https://bit.ly/47EbKP3
S3	What Is Better: One General Model or Many Specialized Models?	https://bit.ly/47gHtpJ
S4	Multi-Agent Reinforcement Learning with Coordination Graphs	https://bit.ly/3sAHkhA
S5	Model-Based RL for Decentralized Multi-agent Navigation	https://bit.ly/3sF42VV
S6	EFlow — Racing towards millions of ML flows	https://bit.ly/49BARnj
S7	L33T M10y	https://sforce.co/3MNfKom
S8	Generalists vs (Micro)Specialists in AI architectures	https://bit.ly/49FxXhu
S9	Hierarchical Reinforcement Learning for Robustness, Performance and Explainability	https://bit.ly/40CLNx3
S10	Hierarchical Reinforcement Learning	https://bit.ly/3R17wLO
S11	Single agent vs multi agent system in AI	https://bit.ly/40DwcNJ
S12	Train Agents Using Parallel Computing and GPUs	https://bit.ly/49GQNVi
S13	Centralized Training and Decentralized Execution in Multi-Agent Reinforcement Learning	https://shorturl.at/fpEN1
S14	Multi-Agent Reinforcement Learning (MARL) and Cooperative AI	https://shorturl.at/bciM3
S15	Decentralized Multi-Agent Reinforcement Learning and Game Theory	https://shorturl.at/fANQW
S16	What is Hierarchical Reinforcement Learning?	https://shorturl.at/tGVW1
S17	The Promise of Hierarchical Reinforcement Learning	https://shorturl.at/ekBN1
S18	Saving and Loading your RL Algorithms and Policies	https://t.ly/8BaaS
S19	Running RLlib Experiments	https://t.ly/DOnRn
S20	Accelerate Training in RL Using Distributed Reinforcement Learning Architectures	https://t.ly/JmhJZ
S21	Parallelism Strategies for Distributed Training	https://t.ly/Y9P7Q
S22	Deep Reinforcement Learning and Hyperparameter Tuning	https://t.ly/FQe2V
S23	Hyperparameter Tuning in Reinforcement Learning is Easy, Actually	https://rb.gy/3kj2hq
S24	Transfer Learning in Reinforcement Learning	https://rb.gy/xi6q56
S25	RL — Transfer Learning	https://rb.gy/a37rhn
S26	Introduction to Experience Replay for Off-Policy Deep Reinforcement Learning	https://rb.gy/dh0l5d
S27	Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem)	https://rb.gy/2rdzuk
S28	How to Improve Your Network Performance by Using Curriculum Learning	https://rb.gy/hnhh4d
S29	Curriculum Learning	https://rb.gy/wgjs6p
S30	Parameter Sharing and Tying	https://rb.gy/enucvc
S31	A Complete Guide to Data Augmentation	https://rb.gy/so60h0
S32	Model Compression Techniques for Edge AI	https://rb.gy/84ih5l
S33	Distributed training with TensorFlow	https://urlis.net/myucb96s

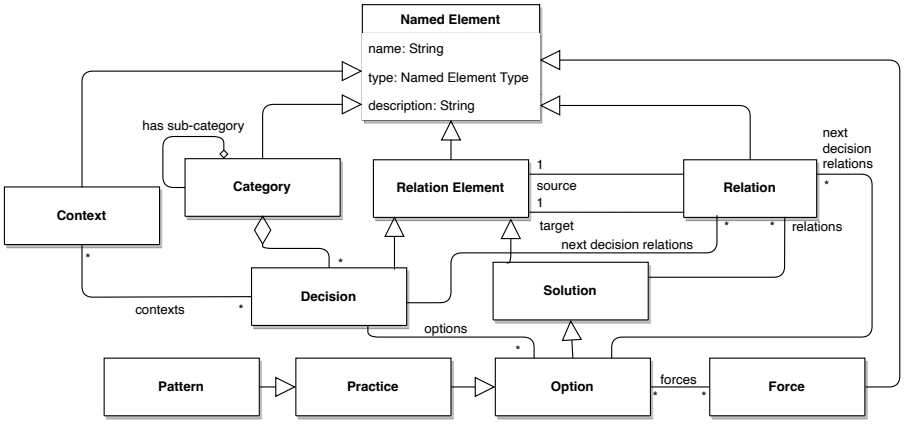

Fig. 2: Meta-model for ADD Models

A. ADD: Model Architecture

Reinforcement learning model architectures are pivotal in RL-based software architectures. There are various approaches to model architecture (Fig. 4) [S1-S8, S10, S16, S17], each with advantages and trade-offs.

One prominent option is the *Monolithic Model* where a single, comprehensive RL model is trained centrally to make all predictions. This approach is akin to having a centralized decision-maker that handles all aspects of an operation. This model can capture complex interdependencies effectively, ensuring a holistic understanding of the environment [1]. This pattern uses *Single-Agent Reinforcement Learning*.

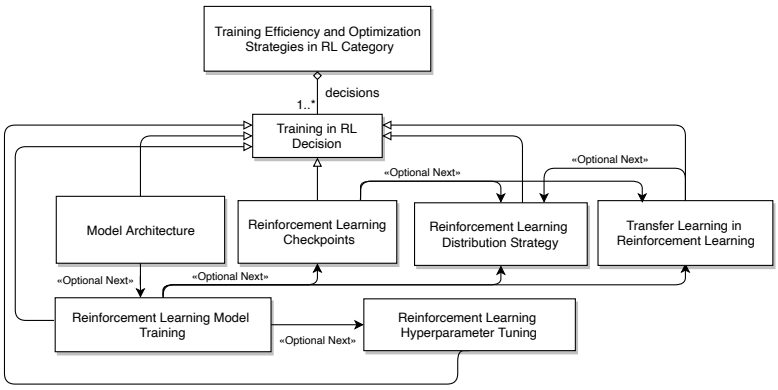

Fig. 3: Reusable ADD Model on Training Strategies in Reinforcement Learning Architectures: Overview

Specialized Models with Multi-Agent-Based Coordination present an alternative perspective, using *Multi-Agent Reinforcement Learning*. This pattern involves multiple independent agents that collaborate to achieve a common goal. Coordination can be facilitated through shared rewards or market-based systems, allowing for decentralized decision-making and resource allocation [21].

The Specialized Models with a Coordinator Specialist introduce a hybrid solution, combining the benefits of specialized agents with the oversight of a central coordinator. This coordinator specialist, often an AI system, possesses a broader view of the entire process and makes high-level decisions based on information collected from specialized models. This architecture aims to strike a balance between distributed coordination and centralized control [2]. This pattern uses *Multi-Agent Reinforcement Learning* for training specialized models.

Hierarchical models offer another dimension to RL architectures, featuring multiple levels of control. A top-level controller manages the broader process, while lower-level controllers handle specific sections or processes. This approach provides a structured framework for decision-making, enabling more efficient handling of complex systems [21].

The monolithic model gains its strength from the advantage of lower *complexity* as it tackles the intricate challenges of building and maintaining a comprehensive RL system. However, a monolithic model's *flexibility* may be limited, especially in dynamic environments requiring rapid adjustments. Adapting to changes may be more challenging due to the model's overarching nature. *Specialized Models with Multi-Agent-Based Coordination* can enhance *performance* and *adaptability* Specialized models can excel at their designated tasks, leading to efficient overall system performance and can lead to more efficient *resource utilization*. In contrast, the *Specialized Models with a Coordinator Specialist* can also enhance *performance*. Specialized models and hierarchical models may offer better *scalability* as they can be expanded or adapted to support better *modularity*. The monolithic model may offer better *interpretability* as it provides a unified structure, making it easier to understand and interpret the decision-making process. Furthermore, it may support better *training efficiency* as it requires less training time and computational resources compared to more complex architectures.

B. ADD: Reinforcement Learning Model Training

There are several patterns and practices related to RL training strategies (Fig. 5) [S4-S21]. *Single-Agent Reinforcement Learning* is a viable strategy for employing a monolithic process control model. In this scenario, a singular model undertakes the formidable task of learning to manage all facets of production control. The technique of *Parallel Training of a Single Agent* is employed to expedite learning without introducing multiple agents interacting. This involves running multiple instances of the same task in parallel, a process known for its efficiency gains in reinforcement learning [22]. On the other hand, the utilization of *Multi-Agent Reinforcement Learning (MARL)* aligns with model architectures involving specialized models with distributed coordination.

The training methods within MARL encompass distinctive strategies, such as *Centralized Training with Centralized Execution*, where agents are trained with access to a centralized controller providing complete environmental state information. This centralization extends to the execution phase, ensuring consistent access to global information. Alternatively, *Centralized Training with Decentralized Execution* allows agents to train with centralized information but execute decisions independently. Additionally, the framework of *distributed MARL* involves multiple agents being trained independently on different computational nodes or devices, collaborating or competing in a shared environment.

Another practice of RL training strategies is *Market-Based Learning*, where agents engage in a market-like environment, exchanging resources or making decisions based on market dynamics. This practice can be combined with *Single-Agent Reinforcement Learning* and *Multi-Agent Reinforcement Learning*. This often involves mechanisms like auctions, negotiations, or trading. Another pattern, *Hierarchical Reinforcement Learning*, introduces a hierarchical decision-making process, where high-level controllers set goals, and lower-level controllers execute actions based on these goals [21].

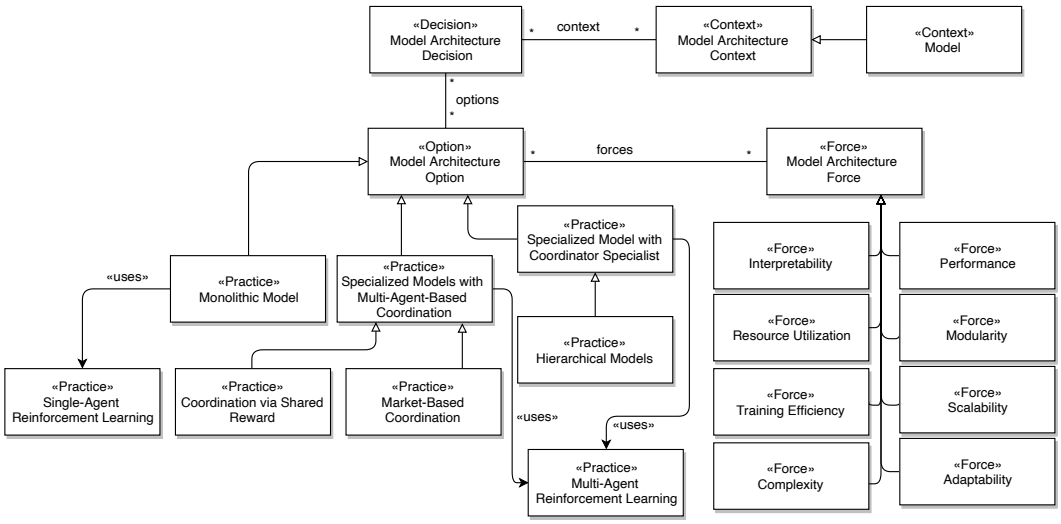

Fig. 4: Model Architecture Decision

Single-Agent Reinforcement Learning, particularly with parallel training, benefits *training efficiency* through parallelization techniques but may face challenges in highly complex systems. As a structured approach, *Hierarchical Reinforcement Learning* enhances *scalability* and mitigates *complexity* by allowing agents to focus on localized adaptations within a hierarchical framework. *Parallel Training of a Single Agent* enhances *training efficiency* and *performance* by leveraging parallelization techniques.

C. ADD: Reinforcement Learning Checkpoints

Checkpoints (Fig. 6) [S18, S19, S26] are a pivotal practice in RL model development. They serve a multifaceted purpose, primarily enabling resumable training by saving the model's parameters, optimizer state, and related data at specific intervals. This ensures that in the event of an interruption or system failure, training can be seamlessly resumed from the last checkpoint, promoting robustness in the training process. Furthermore, *Checkpoints* play a crucial role in version control, capturing the evolution of the model at different stages of development. In collaborative settings, this allows for easy tracking of changes and facilitates comparison between different versions. Additionally, *Checkpoints* are instrumental in the implementation of early stopping strategies, where the model's progress is periodically evaluated on a validation set. If the model fails to improve over a set number of epochs, training can be halted, and the model can be reverted to the checkpoint representing its optimal performance. This practice aids in preventing overfitting and ensures the development of a well-generalized RL model. The decision is binary with two main options: *No Use of Checkpoints* and *Use of Checkpoints*.

While this pattern enhances the *robustness* and *efficiency* of workflows, it introduces considerations such as storage and *computation overheads*, and *complexity* in implementation. Despite these drawbacks, the benefits of preserving the model state far outweigh the challenges, ensuring a more *reliable* and *flexible* training process.

D. ADD: Transfer Learning in Reinforcement Learning

Transfer Learning [5] (Fig. 7) [S24, S25, S26] involves leveraging knowledge gained from solving one problem and applying it to a different but related problem. In the realm of RL, transfer learning becomes a powerful tool when dealing with new tasks or environments. The process typically involves taking a pre-trained model, often trained on a large and diverse dataset or a different but related task, and adapting it to the target RL problem.

Transfer Learning is particularly beneficial when the source task shares some underlying features or patterns with the target task. It allows the RL agent to leverage relevant knowledge from the source task, providing a head start in learning the intricacies of the new environment. This approach is especially valuable in situations where collecting large amounts of task-specific data is expensive or time-consuming. This decision includes two main options: *No Use of Transfer Learning* and *Use of Transfer Learning*.

The utilization of transfer learning presents several advantages that contribute to enhanced *efficiency* and *adaptability*. One key benefit is *data efficiency*, where transfer learning leverages knowledge from a pre-trained model trained on a source task with abundant data. This knowledge encompasses learned feature representations, diminishing the reliance on extensive amounts of task-specific data. *Resource efficiency* is another compelling advantage, as transfer learning allows for the reuse of pre-trained model weights instead of training a model from scratch. This significantly reduces *computational resources*, leading to faster model convergence and decreased hardware demands. The application of transfer learning also results in *improved generalization*, as the model extracts high-level features and representations from the source domain, enhancing its capacity to generalize to new tasks, even when

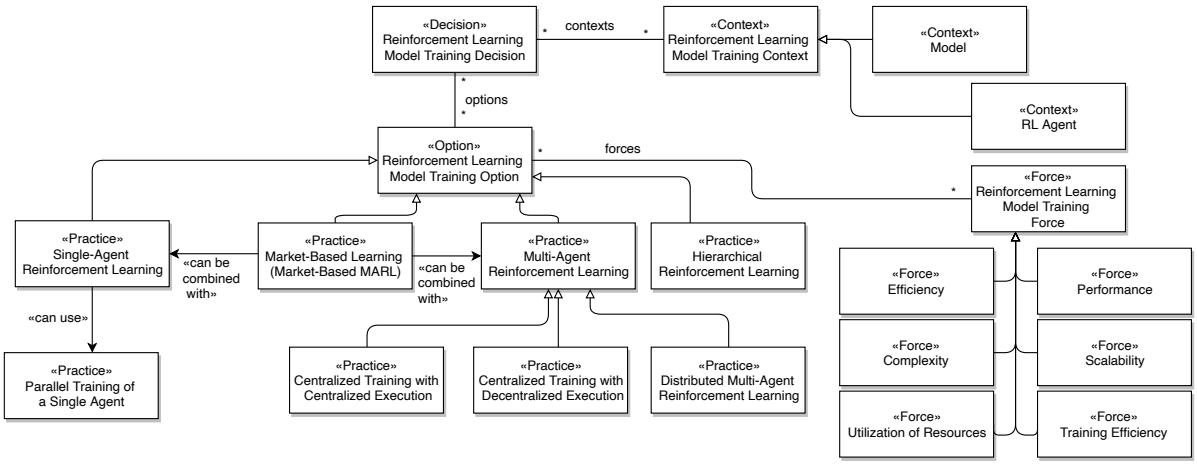

Fig. 5: Reinforcement Learning Model Training Decision

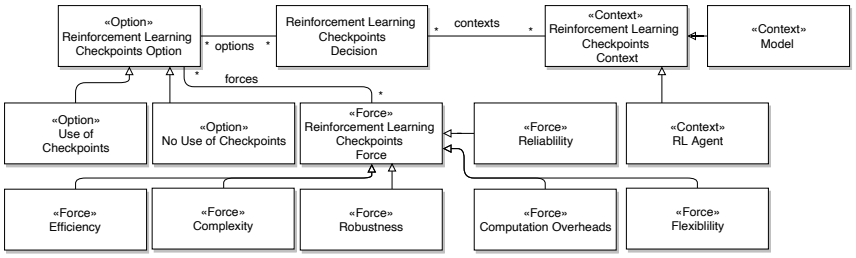

Fig. 6: Reinforcement Learning Checkpoints Decision

confronted with limited target data. Additionally, transfer learning facilitates domain adaptation by enabling the model to adapt to shifts in data distribution. The model identifies key features and knowledge from the source domain, effectively applying them to the target domain. These advantages make transfer learning valuable, especially in scenarios with limited data and computational resources, enhancing the model's *adaptability* and *generalization* capabilities.

E. ADD: Reinforcement Learning Distribution Strategy

In reinforcement learning, a resilient *Distribution Strategy* (Fig. 8) [S12-S15, S20, S30, S33] is vital for efficiently managing computationally demanding tasks [23]. Distributed Reinforcement Learning incorporates strategies like Parameter Server architectures and Distributed Experience Replay. Actor-Critic architectures, exemplified by A3C and Ape-X, facilitate effective learning through a combination of independent exploration and a central value function. Parameter sharing architectures, as seen in D4PG, and policy gradient methods like PPO and TRPO, can be parallelized for faster training. The selection among these approaches hinges on factors such as task complexity and available computational resources [23]. This decision includes two options: *No Use of Distribution Strategy* and *Use of Distribution Strategy*.

From a software architecture perspective, developers often contend with a variety of frameworks, such as parameter servers, MPI-like collective communication primitives, and task queues, to implement RL algorithms effectively (c.f. [24]). As algorithms become more complex, there's a tendency to construct custom distributed systems, where processes operate autonomously and coordinate without centralized control. This necessity arises from the irregular computational nature of RL algorithms, challenging conventional distribution frameworks and prompting developers to integrate various solutions for efficient implementation. Consequently, a choice emerges between *Custom Distribution Frameworks for Each RL Algorithm* and *Versatile Distribution Frameworks*, like Ray/RLlib and TensorFlow, which offer abstraction across a wide range of RL algorithms. Most RL algorithms today use a fully distributed style [24]. *Distributed Control-Based Distribution* involves implementing RL algorithms with independent, replicated processes that coordinate through various means. This is suitable for algorithms with minimal interaction and independent components. *Logically Centralized Control Distribution* employs a central controller delegating tasks to parallel processes, simplifying implementation. It suits scenarios where coordination is critical, and a single control point can manage parallel execution effectively. Hierarchical Parallel Task Distribution extends the logically centralized model, allowing nested delegation for complex algorithms. It enhances flexibility and scalability, making it suitable for tasks with varying levels of parallelism and complex dependencies.

Relating to the choice between *Custom Distribution Frameworks for Each RL Algorithm* and *Versatile Distribution Frameworks*, *Distributed Control-Based Distribution* aligns

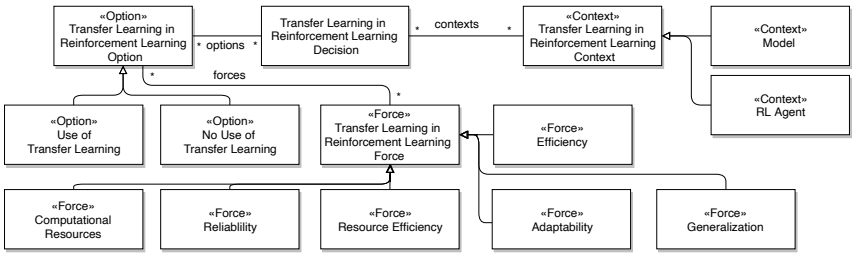

Fig. 7: Transfer Learning in Reinforcement Learning Decision

with custom frameworks, offering precision for specific algorithm needs [24]. *Logically Centralized Control Distribution* and *Hierarchical Parallel Task Distribution* are usually employed by versatile frameworks, providing a unified solution for diverse RL algorithms [24].

Distribution Strategy further enhances *resource utilization* by dividing the training dataset among different machines, enabling simultaneous processing and aggregation of updates. To coordinate this distributed effort, sophisticated distributed optimization algorithms ensure *synchronized parameter updates*, preventing conflicts and promoting convergence. The amalgamation of these strategies results in accelerated *training times* and *scalability*, making it particularly beneficial for handling large datasets and complex reinforcement learning models. This distributed approach represents a powerful paradigm shift in the training of RL models, addressing the computational challenges inherent in the development of sophisticated and high-performance reinforcement learning systems.

F. ADD: Reinforcement Learning Hyperparameter Tuning

RL Hyperparameter Tuning [9] (Fig. 9) [S19-S23] is a critical aspect of optimizing the performance and efficiency of RL models. Hyperparameters are external settings that control the behavior of the learning algorithm, and finding the right combination of these hyperparameters is crucial for achieving optimal results. This decision includes two options: *No Use of Hyperparameter Tuning* and *Use of Hyperparameter Tuning*.

Optimizing hyperparameters in reinforcement learning models yields multifaceted benefits. The quest for optimal hyperparameters translates directly into improved model *performance*, amplifying the learning capabilities of RL models and also improving *training efficiency*. Additionally, the systematic exploration of hyperparameter space contributes to efficient *resource utilization*. By avoiding computational expenditures tied to suboptimal configurations, these methods ensure that computing *resources are allocated* judiciously, optimizing both time and energy. Furthermore, the time savings afforded by automated hyperparameter tuning is a crucial advantage. By automating the tuning process, researchers and practitioners can redirect their time and effort toward other critical facets of model development, fostering a more streamlined and productive workflow.

VI. Evaluation

We systematically developed an ADD model by closely aligning with the selected sources as outlined in Table I. The nomenclature for the ADD model elements was derived from the terminology employed in these sources, and we established generic type names based on these element names. When introducing a new type name, we conducted a thorough comparison with existing names to ascertain its necessity. As depicted in Figure 10, the point of theoretical saturation in Grounded Theory [8] was reached after assimilating twenty-six sources. Across the initial thirteen sources, frequent adjustments to designated type names were necessary. However, in the subsequent thirteen sources, such modifications occurred less frequently. Notably, no further alterations were required for the remaining sources.

In addressing the RQs, we employ the count of new model element types and relation types introduced per source as an indicator of theoretical saturation. This is reasonable, as theoretical saturation is reached when the researcher can no longer uncover fresh concepts or relationships in the data. In our models, the introduction of new model element and relation signifies the emergence of novel concepts and relationships.

VII. Discussion

This section discusses our findings for the research questions from Section I.

RQ1: After analyzing *33* practitioner knowledge sources, we discovered evidence for *29* patterns and practices currently used by practitioners for supporting training in RL architectures, which we modeled as ADD decision options. These patterns and practices are associated with ADDs and were found to be independent of each other. An exception is the *Monolithic Model* and *Specialized Models with Multi-Agent-Based Coordination* patterns, which can use the *Single-Agent Reinforcement Learning* practice. Another commonality is that the *Specialized Models with Multi-Agent-Based Coordination* and *Specialized Model with Coordinator Specialist* patterns are used for training the *Multi-Agent Reinforcement Learning*. The adoption of *Specialized Models with Multi-Agent-Based Coordination* or *Hierarchical Models* is motivated by the pursuit of adaptability and efficient resource utilization, albeit with increased complexity. These strategies reflect the nuanced decision-making landscape where practitioners balance architectural choices with the specific demands of their applications.

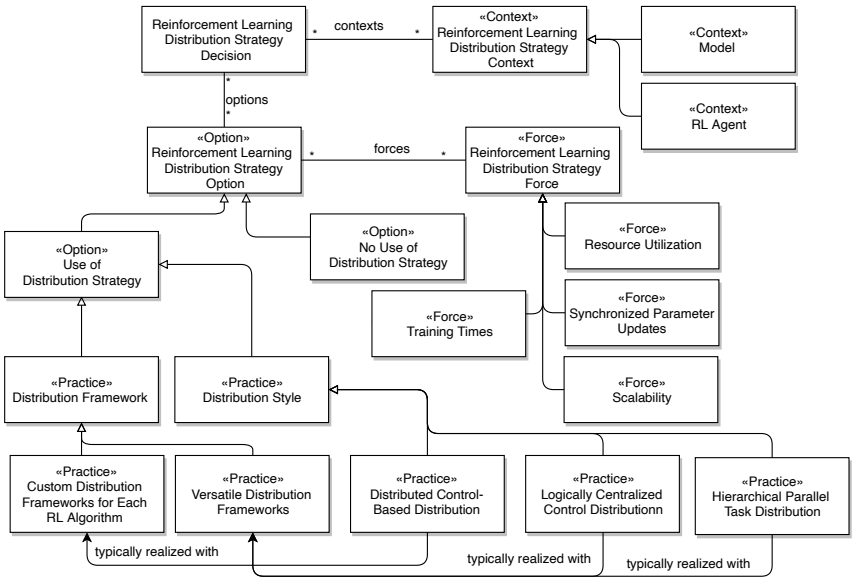

Fig. 8: Reinforcement Learning Distribution Strategy Decision

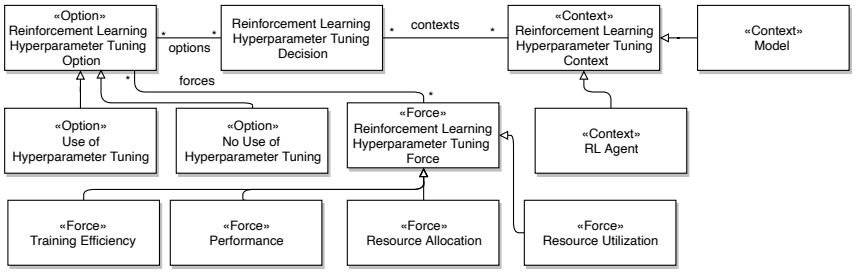

Fig. 9: Reinforcement Learning Hyperparameter Tuning Decision

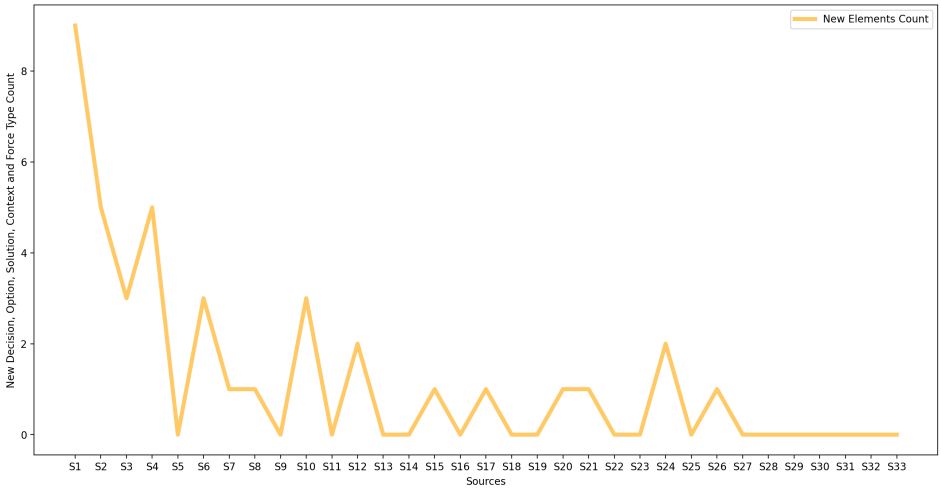

Fig. 10: Number of Elements of Newly-Added Sources

Moreover, *Single-Agent Reinforcement Learning* is applied to *Monolithic Model*, often with *Parallel Training of a Single Agent* for efficiency gains. *Multi-Agent Reinforcement Learning* involves distinctive strategies like *Centralized Training with Centralized Execution* and *Centralized Training with Decentralized Execution*, as well as *Distributed MARL* where agents train independently on different nodes. *Market-Based Learning* introduces market dynamics, while *Hierarchical Reinforcement Learning* features a hierarchical decision-making process. Furthermore, the implementation of *Checkpoints* is

crucial for resumable training, version control, and early stopping strategies. The practice of *Transfer Learning* leverages knowledge from one task to another. A *Distribution Strategy* optimizes RL model training using *Distributed Control-Based Distribution*, *Logically Centralized Control Distribution* and *Hierarchical Parallel Task Distribution*. *Hyperparameter Tuning* is essential for optimizing model performance and training efficiency by systematically exploring hyperparameter space.

RQ2: Given the central *Training Efficiency and Optimization Strategies* category, we identified *6* top-level ADDs for supporting training in RL architectures. Our research revealed subtle relations between ADDs and decision options. For instance, *Specialized Models with Multi-Agent-Based Coordination* leverage *Multi-Agent Reinforcement Learning* and allow for decentralized decision-making through collaboration. *Single-Agent Reinforcement Learning* is employed in a monolithic setting, where a single model manages all facets of production control and utilizes *Parallel Training of a Single Agent* to expedite learning without introducing multiple interacting agents.

RQ3: Through our research, we identified *19* influencing factors (forces) relevant to architecting training in RL architectures from the practitioners' perspective. While these forces tended to be specific to individual ADDs and decision options, we also recognized some commonalities.

The choice between a *Monolithic Model*, *Specialized Models with Multi-Agent-Based Coordination* and *Specialized Models with a Coordinator Specialist* depends on factors such as *complexity*, *flexibility*, *performance*, *adaptability*, *resource utilization*, *scalability*, and *interpretability*. Hierarchical models provide a structured framework, contributing to decision-making efficiency. *Single-Agent Reinforcement Learning* or *Multi-Agent Reinforcement Learning* introduces further considerations related to *training efficiency*, *performance*, *scalability*, and *modularity*. The incorporation of *Checkpoints* enhances *robustness* and *efficiency*, though challenges like storage and computation overheads must be considered. *Transfer Learning* and *Distribution Strategy* further influences *efficiency*, *adaptability*, and *resource utilization*. Additionally, *Reinforcement Learning Hyperparameter Tuning* plays a critical role in optimizing model *performance*, *training efficiency*, and *resource utilization*.

Considering the crucial role of the mentioned forces in various ADDs and their associated decision options, practitioners are advised to evaluate their significance early in the architectural planning phase. This assessment can guide informed decision-making throughout the system's development.

VIII. Threats to Validity

We discuss the threats to validity based on the threat types by Wohlin et al. [25].

To enhance internal validity, we included independent practitioner reports alongside our study, minimizing potential bias from participant awareness during interviews. While this mitigates bias, there's a risk of missing crucial information in reports that could have surfaced in interviews. To address this, we extensively examined various sources, exceeding theoretical saturation requirements. Considering diverse sources minimizes the chance of collectively overlooking vital information.

To mitigate researcher bias, different team members independently cross-verified all models. However, a potential threat to internal validity remains, as biases may persist within the research team, impacting modeling. Despite potential variations in modeling approaches by other researchers, our study's overarching goal to establish a comprehensive model for all observed phenomena reduces the significance of this threat.

We acknowledge the potential bias introduced by the experience and search-based procedure for identifying knowledge sources. However, our research method, which relies on additional sources meeting inclusion and exclusion criteria rather than a specific distribution, largely mitigates this threat. This approach aligns with common selection practices for interview participants in qualitative research studies in software engineering. Nevertheless, there's a potential risk of unintentional exclusion of specific sources, which we address by forming an author team with extensive field experience and conducting thorough, broad searches. Despite the diverse range of included sources, our results likely have generalizability to various architectures involving training strategies. However, a threat to external validity persists, indicating that our findings are specifically applicable to similar types of RL architectures.

We recognize the potential bias resulting from our study's restricted data scope and the absence of pertinent architectural solutions. While our inclusion of numerous sources suggests potential generalizability to various architectures requiring training strategies, the threat to external validity persists. Our findings may only be applicable to similar training strategies in RL architectures, limiting their generalization to novel or unconventional RL architectures without model modification.

IX. Conclusion

We conducted a literature study using Grounded Theory to develop a model for training strategies in RL architectures, encompassing ADDs, decision options, relations, and decision drivers. Our research centered on supporting training strategies in RL architectures and addressed three key research questions. In RQ1, an analysis of 33 practitioner knowledge sources revealed 29 options (patterns and practices) employed in RL training. These were modeled as ADD decision options, demonstrating a high degree of independence. RQ2 delved into the top-level ADDs for training strategies in RL architectures, identifying six key ADDs and uncovering subtle relationships among them, providing crucial insights for RL architectural planning. In RQ3, we identified 19 influencing factors (forces) impacting training in RL architecture design, recognizing their variations across individual ADDs and their options.

X. Acknowledgements

This work was supported by the FFG (Austrian Research Promotion Agency) project MODIS (no. FO999895431).

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] A. S. Bhatia, M. K. Saggi, A. Sundas, and J. Ashta, "Reinforcement learning," *Machine Learning and Big Data: Concepts, Algorithms, Tools and Applications*, pp. 281–303, 2020.

[3] D. Lee, N. He, P. Kamalaruban, and V. Cevher, "Optimization for reinforcement learning: From a single agent to cooperative agents," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 123–135, 2020.

[4] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, p. 4948, 2021.

[5] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[6] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 3053–3062.

[7] U. Zdun, M. Stocker, O. Zimmermann, C. Pautasso, and D. Lübke, "Supporting Architectural Decision Making on Quality Aspects of Microservice APIs," in *16th International Conference on Service-Oriented Computing*. Springer, 2018.

[8] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. de Gruyter, 1967.

[9] T. Eimer, M. Lindauer, and R. Raileanu, "Hyperparameters in reinforcement learning and how to tune them," *arXiv preprint arXiv:2306.01324*, 2023.

[10] H. Washizaki, F. Khomh, Y.-G. Guéhéneuc, H. Takeuchi, N. Natori, T. Doi, and S. Okuda, "Software-engineering design patterns for machine learning applications," *Computer*, vol. 55, no. 3, pp. 30–39, 2022.

[11] R. Sharma and K. Davuluri, "Design patterns for machine learning applications," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, 2019, pp. 818–821.

[12] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," *J. Syst. Softw.*, vol. 82, no. 8, pp. 1249–1267, 2009.

[13] I. Lytra, S. Sobernig, and U. Zdun, "Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study," in *Proc. of WICSA/ECSA*, 2012.

[14] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. Big Web Services: Making the right architectural decision," in *Proc. of the 17th World Wide Web Conference*, 2008, pp. 805–814.

[15] I. Gorton, J. Klein, and A. Nurgaliev, "Architecture knowledge for evaluating scalable databases," in *Proc. of the 12th Working IEEE/IFIP Conference on Software Architecture*, 2015, pp. 95–104.

[16] S. J. Warnett and U. Zdun, "Architectural design decisions for machine learning deployment," in *19th IEEE International Conference on Software Architecture (ICSA 2022)*, 2022. [Online]. Available: http://eprints.cs.univie.ac.at/7270/

[17] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions," *J. Syst. Softw.*, vol. 85, no. 4, pp. 795 – 820, 2012.

[18] J. Coplien, *Software Patterns: Management Briefings*. SIGS, New York, 1996.

[19] C. Hentrich, U. Zdun, V. Hlupic, and F. Dotsika, "An Approach for Pattern Mining Through Grounded Theory Techniques and Its Applications to Process-driven SOA Patterns," in *Proc. of the 18th European Conference on Pattern Languages of Program*, 2015, pp. 9:1–9:16.

[20] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: a critical review and guidelines," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 120–131. [Online]. Available: https://doi.org/10.1145/2884781.2884833

[21] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[22] J. Orr and A. Dutta, "Multi-agent deep reinforcement learning for multi-robot applications: A survey," *Sensors*, vol. 23, 2023.

[23] M. R. Samsami and H. Alimadad, "Distributed deep reinforcement learning: An overview," *arXiv preprint arXiv:2011.11012*, 2020.

[24] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 3053–3062. [Online]. Available: https://proceedings.mlr.press/v80/liang18b.html

[25] C. Wohlin, P. Runeson, M. Hoest, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering*. Springer, 2012.