Bridging the Gap Between MLOps and RLOps: An Industry 4.0 Case Study on Architectural Design Decisions in Practice

Stephen John Warnett[1, 2], Uwe Zdun[1]
[1] *Research Group Software Architecture, Faculty of Computer Science, University of Vienna, Vienna, Austria*
[2] *UniVie Doctoral School Computer Science DoCS, Faculty of Computer Science, University of Vienna, Vienna, Austria*
firstname.lastname@univie.ac.at

***Abstract*—In the domain of Industry 4.0 Cyber-Physical Production Systems (CPPSs), Reinforcement Learning (RL) has gained momentum as an effective strategy for training intelligent agents in digital twins. Whilst the practice of Machine Learning Operations (MLOps) has become established as a holistic approach to automating workflows in supervised and unsupervised Machine Learning (ML), the extent to which MLOps practices are applicable to RL, particularly due to major differences between ML and RL concerning model deployment and model training, are not currently well-understood. The literature on RLOps as a paradigm is scarce. We tackle this open question by conducting an exploratory, qualitative, deductive-inductive industry case study on a CPPS, performing content analysis of CPPS artefacts, such as architectural schematics and source code, and understanding their relation to 22 known Architectural Design Decisions and 86 associated decision options through classification into four distinct emergent categories. Our findings help bridge the gap between MLOps and RLOps architectures, contributing novel insights into understanding the application of MLOps to RL and providing practical insights and inspiration for further research.**

***Index Terms*—MLOps, RLOps, machine learning, reinforcement learning, architectural design decisions, industry 4.0, cyber-physical production systems**

I. Introduction

Industry 4.0 [1] has revolutionised manufacturing by integrating disparate digital technologies, resulting in the development of Cyber-Physical Production Systems (CPPSs) [2], which mix physical and computational elements to improve efficiency, flexibility and customisation [3]. Within this setting, Reinforcement Learning (RL) [4] has emerged as an effective technique for training intelligent agents in digital twins, allowing for real-time decision-making and process optimisation [5], [6]. MLOps (Machine Learning Operations) has become a standard method for automating workflows in supervised and unsupervised Machine Learning (ML), including data processing, model training, deployment and maintenance [7]. However, the application of MLOps concepts to RL – a burgeoning practice known as RLOps [8] – is largely unexplored. The potential applicability of MLOps to RL is unclear due to fundamental differences between ML and RL, for instance, in model deployment and training. Whilst ML models are usually trained on static datasets, RL models learn by interacting with dynamic environments, necessitating continual adaptation and real-time decision-making, facing their own set of challenges [9].

The scarcity of scientific literature on RLOps creates a gap in our understanding of how to integrate RL into the MLOps paradigm successfully. Addressing this gap is critical for improving CPPS capabilities and realising the full promise of Industry 4.0 [10]. We intend to bridge the knowledge gap between MLOps and RLOps by performing an exploratory, qualitative industry case study on a CPPS.

Our objectives are to evaluate the applicability of ML-specific Architectural Design Decisions (ADDs) to RL and RLOps, investigate the relationship between MLOps techniques and RL in a CPPS and offer a reference for practitioners as well as a framework for future study. The results should add to the expanding body of knowledge regarding Industry 4.0, MLOps and RLOps, aiding practitioners in improving the efficiency and adaptability of CPPSs and providing fresh perspectives on integrating RL into MLOps.

We carried out an exploratory, qualitative industry case study on a CPPS. We used content analysis [11] to determine how 86 decision options derived from 22 architectural design decisions related to 15 CPPS artefacts such as source code and architectural schematics, which we describe in more detail in Section IV-D. We aimed to answer the following research questions:

- **RQ1** To what extent are known MLOps and RL ADDs applicable to RL in a CPPS context?
- **RQ2** Which ADD decision options typically used in MLOps and RL apply directly to a real-world CPPS that utilises RL?
- **RQ3** Is it necessary to adapt ADD decision options typically used in MLOps for use with RL in a CPPS context?

We make three main contributions in this paper. Firstly, we conduct a qualitative industry case study on ML and RL practices in a real-world CPPS, modelling 15 system artefacts to establish the use of relevant ADDs, decision options, practices and patterns. Secondly, we interpret our findings, deriving four categories of ADD decision options in an RL context. Finally, we identify ML practices that, at first sight, may not appear applicable to RL but can be easily adapted. In

summary, we reduced the knowledge gap between MLOps and RLOps, providing insights and a reference for practitioners.

The rest of the paper is structured as follows: in Section II, we describe the concepts of ADDs, ML, RL, MLOps, Industry 4.0 and CPPSs. Section III compares our work with related studies. In Section IV, we describe the case study and our approach. Section V details our results, which we discuss in Section VI. We consider threats to validity in Section VII, and Section VIII concludes and suggests ideas for future work.

II. Background

A. Architectural Design Decisions

ADDs are a crucial aspect of software development that represent architectural decisions made during the design process [12], [13]. ADDs and their associated decision options encapsulate the rationale behind the choice of architectural elements, their relationships, and the trade-offs (in terms of *forces* or *decision drivers*) considered to meet functional and quality requirements [14].

Table I, which is discussed in more detail in Sections V and VI, lists the ADDs and decision options of relevance to this study. A central ADD is whether to adopt MLOps as a holistic approach. Other ADDs are mostly concerned with data processing (e.g. *Automatic Data Processing*, *Data Ingestion*, *Data Versioning*), triggers (e.g. *Pipeline/Orchestrator Triggers*), Model Building (e.g. *Model Building Pipeline Tasks*, *Training Strategy*) and model deployment (e.g. *Delivery Pipeline Tasks*, *Model Version Deployment*).

Automation is represented as a high-level ADD (i.e. whether *AutoML* is used) and as a cross-cutting concern that can be considered in the context of decision options in practices related to data processing, pipeline and orchestrator triggering, pipeline tasks, model training, model building, model integration and delivery. The ADDs associated with RL are related exclusively to model architecture, training and distribution strategies. Specifically. the *Model Architecture* and *Model Training* variations specifying single agent [15] vs multi-agent [16], [17] training, the use of *Checkpoints*, whether *Transfer Learning* is utilised, the *Distribution Strategy* adopted and the use of *RL Hyperparameter Tuning*.

B. ML, RL, MLOps and RLOps

ML and RL differ in their methodologies and applications. Whereas ML employs supervised or unsupervised learning techniques that learn from data to make predictions or classifications, RL focuses on learning optimal actions through interactions with an environment and feedback in the form of rewards or penalties for prior actions.

Given this distinction, one might assume the necessity of different operational frameworks for automating ML and RL. MLOps, already very well established, involves practices and tools for data processing and management, model training, deployment, management and maintenance in production. RLOps, an RL-equivalent operational framework for RL that has only been sparsely documented in the literature, must address the unique challenges associated with the RL workflow and production environments, particularly the differences in data processing, model architecture, training, distribution and the complexities of real-time decision-making and dynamic environments.

C. Industry 4.0 and Cyber-Physical Production Systems

Industry 4.0 [1] represents a paradigm shift in manufacturing due to its integration of digital technologies such as the Internet of Things (IoT), Artificial Intelligence (AI) and Cloud Computing to create smart factories. Smart factories are interconnected systems enabling real-time data exchange and autonomous decision-making, enhancing efficiency, flexibility and customisation [10].

CPPSs, which merge physical (e.g. tools, robots and sensors) and computational (e.g. communication networks, data analytics and AI) elements, are central to Industry 4.0. CPPSs monitor, control and optimise manufacturing operations [5], [6]. Seamless interaction between the physical and digital realms using digital twins can improve production quality, reduce costs and simplify maintenance [18], [19].

III. Related Work

RLOps is sparsely mentioned in the scientific literature, appearing more commonly in practitioner sources online, such as code repositories, libraries, commercial platforms and frameworks supporting RLOps. This disparity indicates that RLOps is a relatively new field that is becoming established in practice but demonstrates a knowledge gap from a scientific perspective. Indeed, we could not find any Industry 4.0 case studies focusing on RLOps.

Li et al. [8] study RL as applied to open radio access networks (O-RAN). They highlight the differences between ML and RL, providing a taxonomy of the ML/RL model lifecycle challenges and suggesting best practices for RLOps. They also design and implement a data analytics platform for O-RAN. As with our study, they recognise the differences between ML and RL and consider how MLOps can be applied to RL. Unlike our study, they focus on the model lifecycle rather than a wide range of ADDs, consider O-RAN rather than Industry 4.0 and do not conduct a case study.

Del Real Torres et al. [20] conduct a review of deep RL approaches for smart manufacturing in Industry 4.0 and 5.0 [21], noting the relevance of edge devices and cyber-physical systems. They consider a range of deep RL algorithms and their applicability in manufacturing processes, providing common guidelines for developing and improving factories. Similarly to our study, they consider RL in an Industry 4.0 context. Unlike our study, they conduct a literature review rather than a case study and focus on algorithms rather than MLOps and architectural aspects.

Kegyes et al. [22] conduct a systematic literature review of RL applications and methods used in Industry 4.0. As in our study, their work serves as an overview and reference. However, they focus on the theoretical underpinnings of RL and implementation details rather than architectural decisions

and MLOps. Unlike our study, they survey the literature rather than conduct a case study.

Zhou et al. [23] recognise the need for MLOps as an application of DevOps [24], [25] principles to unify ML system development and operation and to address the unique challenges faced when developing and deploying ML applications. They build an ML platform using DevOps and then measure its performance, providing a reference ML pipeline platform. Despite using a system as a case study, their example is not a real-world case study of an Industry 4.0 application and does not consider RL.

Our study examines the relevance of MLOps ADDs in an Industry 4.0 setting as applied to RL. It aims to bridge the gap between theory and practice in understanding how associated practices apply in that context. To our knowledge, ours is the first study to do so.

IV. Case Study

This section provides a detailed description of the case study design and research process, which is depicted in Figure 1. Our research methodology adheres to the case study guidelines outlined by Runeson and Höst [26], and Yin [27].

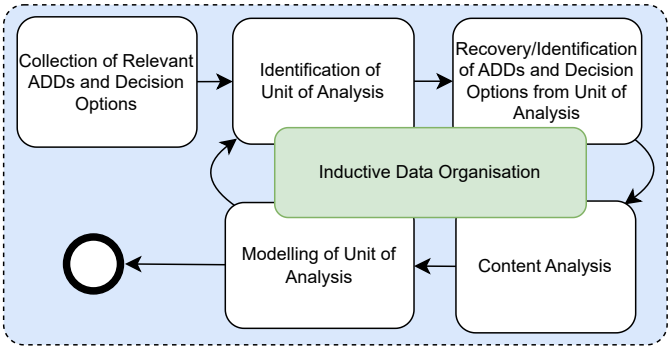

Fig. 1: An overview of the research process we followed in this study.

The study was conducted within the framework of a previously established academia-industry project collaboration between the authors and a large multinational technology company that provides production automation solutions. The project was chosen due to its relevance to the academic and industrial partners involved in the collaboration. The focus of the project is an RL-based CPPS solution being developed by a team of experts at the company for automating manufacturing processes in factories.

A. Research Process

The first step was the **Collection of Relevant ADDs and Decision Options**, which we sourced from our prior work [28], [29], [30], where we had derived them from practitioner grey literature using Straussian Grounded Theory [31]. The following four steps (collectively the **Inductive Data Organisation**) were performed iteratively until the entire *study object* (see Section IV-C) had been covered. The **Identification of Unit of Analysis** step involved finding the next potential *unit of analysis* (see Section IV-D). **Recovery/Identification of ADDs and Decision Options from Unit of Analysis** was a check to see if the current unit of analysis evidenced the use of a decision option. If so, **Content Analysis** of the unit of analysis, as described in Section IV-E was performed. Finally, if the unit of analysis evidenced the use of a decision option, **Modelling of Unit of Analysis** (see Section IV-F) involved writing component, pipeline and process models in Python and generating their UML visualisations.

B. Approach

Our study employs an empirical and qualitative design, examining contemporary phenomena within their context [27]. Unlike action research [26], which aims to effect change, our approach is observational, aiming for a high degree of realism at the expense of control. We focus on studying and analysing project artefacts and are interested in theory building and discovering how practitioners have applied ADDs. We aim to avoid influencing practitioners or introducing bias, e.g. via interviews, and we always strive to maintain a neutral frame of reference.

Despite the study's observational nature, practical uses of our findings are discussed in Sections VI and VIII. This exploratory study aims to investigate the implementation of the study object (see Section IV-C), gain new insights and understanding, address our research questions and generate ideas for future research.

Additionally, the study serves as a descriptive study, providing a detailed portrayal of a specific situation and its associated phenomena. The study design incorporates the flexibility to effectively manage "the complex and dynamic characteristics of real-world phenomena, like software engineering" [26].

C. Study Object

The focus of our research (the *study object*) is an innovative, proprietary software system developed by domain and RL experts within a large multinational technology company. This advanced solution integrates RL with a CPPS, leveraging MLOps and RLOps practices for an Industry 4.0 manufacturing environment. The system's task is to automate various aspects of industrial product manufacturing, including physical modification and assembly of components.

D. Units of Analysis and Data Collection

The *units of analysis* are those data sources within the study object that are related to ML, RL, MLOps and RLOps. A range of disparate data sources enabled the triangulation of the phenomena we encountered. Specifically, we practised *third-degree data collection*, which involves the independent analysis of work artefacts that are already available. We made use of project source code, CI/CD pipeline definitions and configuration files, high-level schematics, informal architectural diagrams, UML diagrams, presentations by team members and associated slide decks, and observation of team behaviour practices (e.g. source repository commits and their comments)

to gain an understanding of the system and to model relevant parts.

Since we did not conduct formal interviews, and this is not an ethnographic study where, for instance, cultural aspects within an organisation are studied, there are no human subjects in that sense. However, informal discussions and meetings with project partners throughout the normal course of our project collaboration helped with clarification and disambiguation when understanding the system and practices.

E. Content Analysis

Using the ML and RL ADDs from Table I as an organisational framework, we systematically analysed the data sources to identify and gather evidence of the use of any of the ADDs, along with associated decision options in the form of architectural patterns and practices. This aspect of the study was deductive in that we started with knowledge and understanding of the ADDs from the literature relevant to the domain and research questions that we deemed a reasonable starting point. Still, we had no predefined hypotheses or assumptions about what we expected to find.

We conducted a content analysis of the data source based on DeFranco and Laplante [11], which is a method well-suited for systematically examining software engineering data. This flexible inductive data organisation approach involves selecting suitable sources and employing coding techniques to categorise data. The method can facilitate the analysis of various software engineering artefacts and yield rich insights grounded in project-specific data, in our case study regarding the application of ADDs within a software system.

We identified patterns and themes and coded contextual information and findings to support sensemaking [27]. We developed broader categories for the discovered architectural properties of the system. We consulted additional literature to aid and validate our understanding of the encountered phenomena as necessary. Our methodology was an iterative, continuous process of ADD recovery and identification, interpretation, categorisation and modelling.

F. Modelling

Following the content analysis, we modelled relevant parts of the system using the same technique as in prior work [32], [33]. We used Codeable Models[1], a Python tool for specifying metamodels and model instances. These formal architectural models consist of nodes and connectors that represent components, pipeline and process steps and their relationships, and can be rendered in UML using PlantUML[2]. Figure 2 depicts a component view of part of the modelled RL system, and Figure 3 a process view.

Modelling the system during content analysis enhances the clarity and effectiveness of our case study. Simultaneously, the modelling process provides a technical reference, facilitates transparency and traceability of our process, and helps us relate our findings to the source material. A further advantage of modelling during our analysis is that it can be used for validation. By formalising our understanding of the system and use of ADDs, we provide an additional way of checking that we correctly understand the system in combination with the clarification and disambiguation provided by industry experts during our discussions.

Due to confidentiality agreements, the sources, that is, the study object itself and its units of analysis, cannot be shared. However, by generating UML visualisations of modelled aspects of the CPPS, we can communicate the salient aspects of the system architecture without violating confidentiality. Note that we did not model the entire system, but only those units of analysis that provided evidence for the use of ADDs and decision options. Our models were simplified to the minimal level of detail required to understand the units of analysis. Likewise, specific technologies were mostly omitted. Some aspects, such as component names, were renamed to protect intellectual property. For replicability and transparency, our models, model visualisations, and an overview of our chain of evidence from the ADDs to our findings are available in our replication package [34]. We continuously validated our results through triangulation across the range of source *units of analysis* and models, cross-checking among the author team. Finally, we provided our results to our project partners for evaluation as an additional means of validation.

G. Ethical Considerations

Confidentiality and the handling of sensitive data were maintained at all times. There was no direct human participation in the case study beyond the usual working relationship with the project partner. The project partner provided informed consent and had the opportunity to review, correct and give feedback on this paper and its artefacts. No inducements were offered, and there are no known conflicts of interest.

V. Results

In this section, we report our results, describing our findings for each ADD. Table I shows the findings of our comprehensive analysis of ADDs for the study object and provides significant insights into the application of ML and RL ADDs in this context.

A. MLOps

A major ADD is whether *MLOps* should be applied as a holistic approach throughout a project. MLOps touches on many other ADDs covered in this study, such as strategies for automating and streamlining data processing, rapid model training, deployment and performance monitoring. The *units of analysis* clearly shows that the decision to adopt an MLOps approach was made. However, the findings below indicate that for many related ADDs, MLOps as a practice needs to be adapted to accommodate the distinct demands of RL, leading to a modified form of MLOps for RL termed RLOps.

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

[2]https://plantuml.com

TABLE I: Summary of the Applicability and Use of Architectural Design Decisions in the System.

ADD/Decision Option	ADD/Decision Option
MLOps ✓ M [28]	**Data Processing** ✓ M [29]
No MLOps ~	Batched ✓
Apply MLOps ✓ E	Real-time, stream-based ~
AutoML ~ M [29]	**Automated Integration/Delivery** ✓ M [28]
No AutoML ✓	None ~
Use AutoML ~	Build and deployment scripts ~
Automatic Data Processing ✓ M [29]	CI/CD pipeline ✓
No data processing automation ~	Machine learning orchestrator ✓
Data pipeline ✗	**Delivery Pipeline Tasks** ✓ M [28]
ETL pipeline ✗	Packaging ~
Data processing component ✓	Testing ~
Data Processing Pipeline Tasks ✓ M [29]	Building ~
Data extraction ✗	Deployment ✓
Data transformation ✗	Containerisation ✓
Data preparation ✗	**Data Versioning** ✓ M [28]
Data validation ✗	None ~
Data selection ✓	Data repository ✓
Feature engineering ✓ E	Code repository ✓
Data processing hyperparameter tuning ✗	**Model Version Deployment** ✓ M [28]
Feature Storage ✓ M [29]	Single model in production ~
Data store ~	N versions in production ✓
Feature store ✓	Model version rollback ✓
Data Ingestion ✓ M [29]	**Model Architecture** ✓ R [30]
Streamed data ingestion ~	Monolithic model using single-agent RL ~
Data ingestion by request ✗	Specialised models + multi-agent coordination via shared rewards ✓
Data ingestion in batches ✓	Specialised models + multi-agent market-based coordination ~
Manual data ingestion ✗	Specialised models + coordinator specialist + hierarchichal models ~
Pipeline/Orchestrator Triggers ✓ M [28], [29]	Specialised models + coordinator specialist ~
On-demand ~	**Model Training** ✓ R [30]
Commit ~	Single-agent RL with parallel training of a single agent ~
Scheduled ✓	Market-based multi-agent RL ~
New training data ✓ E	Centralised training and execution multi-agent RL ~
Model performance degradation ✓ E	Centralised training and decentralised execution multi-agent RL ✓
Data distribution changes ✗	Distributed multi-agent RL ~
Model Building ✓ M [29]	Hierarchical RL ~
In development tool ✓ E	**Checkpoints** ✓ R [30]
Model building pipeline ✓ E	No checkpoints ~
Model builder component ✓	Use checkpoints ✓
Model Building Pipeline Tasks ✓ M [28], [29]	**Transfer Learning** ✓ R [30]
Model training ✓	No transfer learning ~
Data splitting ✗	Use transfer learning ✓
Data checkpoints ✓ E	**Distribution Strategy** ✓ R [30]
Model validation ✓	No distribution strategy ~
Model selection ✓	Custom distribution framework for each RL algorithm ~
Train multiple model versions ✓	Versatile distribution frameworks ✓
Model packaging ~	Distributed control-based distribution style ~
Model hyperparameter tuning ✓ E	Logically centralised control distribution style ✓
Development tool facade ~	Hierarchically parallel task distribution style ~
Development tool export ~	**RL Hyperparameter Tuning** ✓ R [30]
Training Strategy ✓ M [29]	No RL hyperparameter tuning ~
Batched ~	Use RL hyperparameter tuning ✓
Incremental ✓	
Batch-based/incremental hybrid ~	
Model Deployment ✓ M [28]	
Manual deployment ~	
Pre-prepared pipelines ~	
CI/CD pipeline automation ✓	

✓, ~ and ✗ represent an ADD or a decision option that was *applied*, *not applied* and *not applicable* to the study object respectively.
E represents an ML decision option that has an *RL equivalent*.
M represents an ADD that is associated with *ML*.
R represents an ADD that is *RL-specific*.

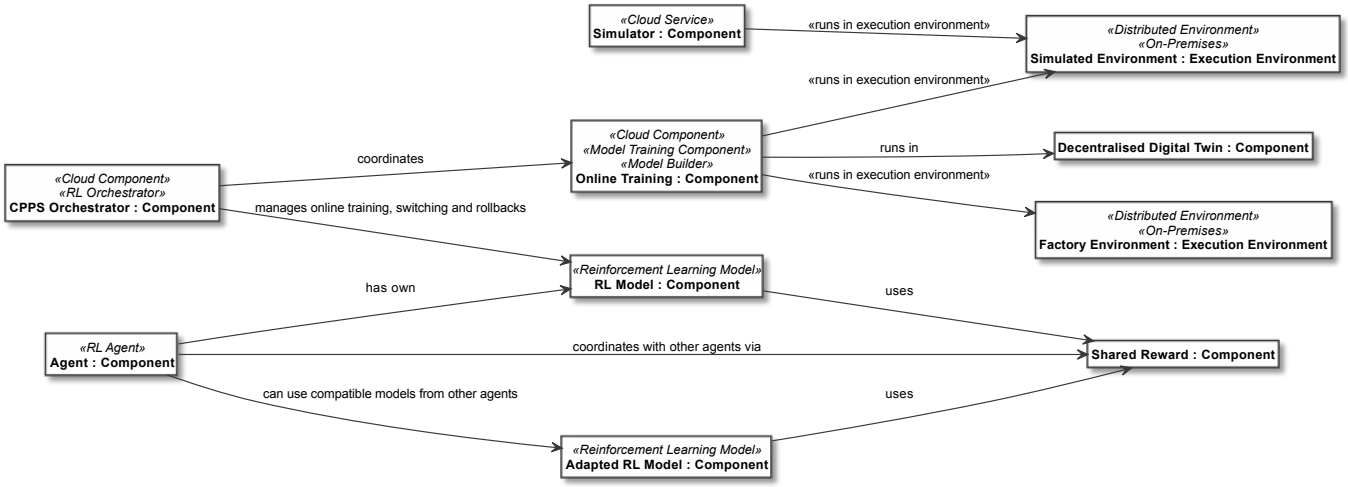

Fig. 2: An example component model UML diagram for a unit of analysis of the study object.

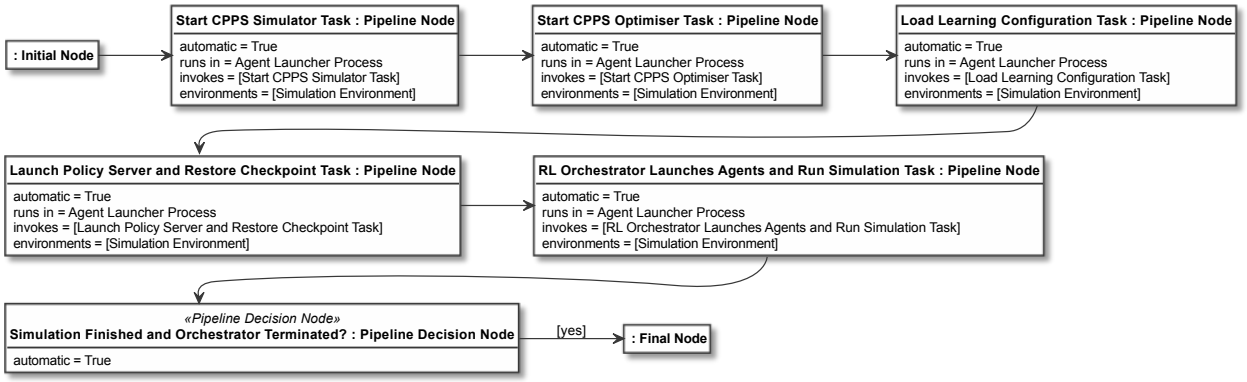

Fig. 3: An example process model UML diagram for a unit of analysis of the study object.

B. AutoML

AutoML automates several ML practices that would otherwise be done manually, such as data preparation, feature engineering, model training, hyperparameter tuning, model validation and model selection. It uses search algorithms to find optimal solutions for the various phases of the ML pipeline. In the study object, AutoML was not applied.

C. Automatic Data Processing

Deciding how to process the data used for model building automatically is central. Neither **data pipelines** nor **ETL pipelines** are suited to RL since RL models are trained based on environmental data and rewards rather than training datasets. In the study object, a **data processing component** in the form of an environment interpreter, which provides environment data to agents, is used, supporting *Automatic Data Processing*. An alternative implementation considered for a future version of the system is the *replay buffer* [4]. Replay buffers are a widely used practice to retain and replay old data so that it is not lost. This alternative would process production data and provide it to the training process.

D. Data Processing Pipeline Tasks

Deciding on *Data Processing Pipeline Tasks* is a follow-on ADD from *Automatic Data Processing* if, for instance, a **data pipeline** or **data processing component** was used. In RL, tasks related to ML-specific training data, such as **extraction**, **transformation**, **preparation**, **validation**, and **data processing hyperparameter tuning** are not applicable. During manual experimentation, **data selection** of environment features is used. This data is used for offline RL [35] and an RL-adapted form of **feature engineering**, which focuses on creating an effective state representation for imitation learning [4]. Offline RL is when an agent learns from past data without interacting with the environment, and imitation learning develops a policy based on datasets from expert demonstrations, potentially surpassing their performance.

E. Feature Storage

The *Data Processing Pipeline Tasks* decision option of **feature engineering**, as applied during manual experiments, prompts the architect to consider *Feature Storage* options. These include a simple **data store** or a specialised **feature**

store, the latter of which was applied in the study object. A feature in the context of RL is a specific attribute or property of the environment that the agent can use to make decisions. The **feature store** option facilitates persistence and access to feature data and, in the study object, the data flows back from the environment interpreter to the **feature store** via a trigger (see the *Pipeline/Orchestrator Triggers* ADD in Section V-G) to be used for offline RL and imitation learning.

F. Data Ingestion

Another key ADD is *Data Ingestion* to enable *Data Processing* and consequently *Model Training*. In this RL-based CPPS case, **data ingestion by request** and **manual data ingestion** are irrelevant since it only makes sense to ingest data when there has been a change in model performance or the environment (such as the factory layout) – see the *Pipeline/Orchestrator Triggers* ADD in Section V-G. Thus, it only remains to be decided whether data is **streamed** or **batched**. Both are feasible, but the **batched** option has been applied. Data resulting from exploring the action space is provided via a trigger, saved in a **feature store** for offline RL and imitation learning, and is then fed into an automated pipeline in batches for model training.

G. Pipeline/Orchestrator Triggers

ML pipelines, orchestrators, and model-building components can be started via *Pipeline/Orchestrator Triggers*. Triggers can be **on-demand** – not used in this project – or based on external events, such as a **commit** to the code base, which is not used either. In RL, triggering on **changes to the data distribution** does not make sense since the system does not deal with labelled datasets with a given distribution, as in supervised learning. In our case study, a modified version of the **new training data** trigger was applied based on action space (e.g. after agent actions or when robot characteristics are otherwise updated) or environment changes (e.g. when the factory layout changes). **Model performance degradation** also triggers pipelines in the study object and is achieved via agent performance monitoring (necessitating an RL-specific adaptation), triggering a training pipeline or component, or when a better model becomes available, triggering a deployment pipeline or model serving component. A model serving component is also triggered according to a **schedule**.

H. Model Building

A core ADD is *Model Building* in an ML project. Options include using **development tools** like **computational notebooks**. Alternatives are to use a **model building pipeline** or a **model builder component**. The study object uses all three – a **development tool** is used for running manual experiments, and a **model building pipeline** and manual experimentation use a **model builder component** to perform model training.

I. Model Building Pipeline Tasks

If a **model building pipeline** is chosen for the *Model Building* ADD, as in our case study, then the *Model Building Pipeline Tasks* ADD is for deciding which tasks are performed in that pipeline. The model building pipeline and other components perform **model validation**, **model selection** from an RL model registry, **training of multiple model versions** and an RL version of **hyperparameter tuning** (see the *RL Hyperparameter Tuning* ADD in Section V-V). **Model building components** perform **model training** and use **data checkpoints** (although modified for use in offline RL). **Data splitting** (e.g. into training and test data) does not apply to this project since it does not use datasets for supervised/unsupervised learning. Model **packaging** is not applied here because models are loaded directly from an RL model repository and into an agent via an orchestrator component. Finally, neither a **development tool facade** nor **export** are used since development tools are not integrated into the automated flow.

J. Training Strategy

Model training, discussed with the *Model Building Pipeline Tasks* ADD in Section V-I, entails a follow-on decision on which *Training Strategy* to adopt. The training strategy may be **batched**, **incremental** or a **hybrid** of the two. In our case study, models are built **incrementally** following automatic triggers described for the *Pipeline/Orchestrator Triggers* ADD in Section V-G.

K. Model Deployment

Once a model is trained, it needs to be deployed for use. Our case study does not use **manual deployment** nor semi-automated **pre-prepared pipelines** – though it is possible to use them, neither of these practices is ideal for an Industry 4.0 CPPS. Instead, **CI/CD pipeline automation** is used to automate a model serving component.

L. Data Processing

Data Processing is paramount in ML. It involves taking data that was already ingested and providing it to, e.g. a model builder component for use in model training. Processing can occur in **batches** when required for training, as in the study object where data is provided in batches with triggers and read in batches for offline and imitation learning, or in **real-time as a stream**, which is not implemented in our case study.

M. Automated Integration/Delivery

Automated Integration/Delivery involves deciding on the level of automation for integration and delivery. The most straightforward choice, **no integration or delivery automation**, is not applied, nor is manually performing deployments **using build and deployment scripts**. Instead, the system makes use of a **CI/CD pipeline** (for building and deployment) and an **ML orchestrator**. The orchestrator loads models from a model repository into agents. It executes the agent and evaluates its actions, observations and rewards in a simulator.

N. Delivery Pipeline Tasks

If **CI/CD pipelines** are used, as they are in the study object, then *Delivery Pipeline Tasks* are a consideration. Typical tasks include **packaging** (e.g. of models), **testing** delivery tasks (e.g. for A/B tests or canary tests), **building** (e.g. of executables),

the **deployment** of artefacts into a target environment and **containerisation** (e.g. of models or components). In our case study, the evaluation of the model is part of a training pipeline but not a delivery pipeline. Model building is part of the training pipeline and not the delivery pipeline. Our case study uses **deployment** tasks as specified in a deployment configuration to deploy a simulator and a **model builder component**. It also performs **containerisation** of Docker images and stores them in a container registry.

O. Data Versioning

Part of data management involves *Data Versioning*. ML or RL-relevant data can be stored and versioned in a dedicated **data repository**. Likewise, ML or RL code can be stored in a **code repository** like GitLab. In the study object, both practices were applied with source code stored in a **code repository**, environment data stored in the **feature store** described in Section V-E, model metadata stored in an ML metadata store and models stored in a model registry.

P. Model Version Deployment

Fundamental to more detailed ADDs (e.g. *Model Architecture* for RL in Section V-Q) is deciding on *Model Version Deployment*. We note that a **single model in production** was not applied since the system uses multi-agent RL. On the other hand, the system was designed to support **n versions in production** – to support staged releases and upgrade subsets of cyber-physical production units – and also **model version rollback** with the use of a model registry.

Q. Model Architecture

In RL, the correct choice of *Model Architecture* is more complex than in ML. One can either make use of a **single monolithic model** for centralised decision-making or multiple **specialised models** with various forms of coordination. Coordination may be achieved with **shared rewards**, be **market-based** or can be achieved with a **coordinator specialist**. If **specialised models** and a **coordinator specialist** are selected, then **hierarchacal models** may also be used. Our analysis shows that the study object uses **specialised models with multi-agent coordination via shared rewards**.

R. Model Training

For the *Model Training* ADD, decision options include patterns and practices incorporating a range of combinable strategies including **single-agent RL**, **multi-agent RL**, **hierarchical RL**, **market-based RL**, **parallel training** and **centralised/decentralised training and execution**. In our study, the system makes use of **centralised training and decentralised execution multi-agent RL** – the models are centrally trained in a model-building component (see the *Model Building* ADD discussed in Section V-H) and multiple decentralised digital twin agents are executed.

S. Checkpoints

Checkpoints are a binary choice and applied in the study object to facilitate resumable training. Resumable training requires saving training parameters, state and data at given intervals during training, allowing for the resumption of training later. This practice can aid training robustness by allowing the training process to recover with minimal waste of resources if interrupted and allows for model versioning during training.

T. Transfer Learning

Transfer Learning is a binary ADD representing a technique for using knowledge gained from solving one problem to solve a related but different task. In the study object, it is used by fetching pre-trained models from the model registry and retraining them to reduce overall training time.

U. Distribution Strategy

A suitable RL *Distribution Strategy* can aid training efficiency and scalability and reduce training times. Depending on specific needs, various combinations of factors characterise the decision options, including frameworks, methods of control and levels of parallelisation. The study object uses **versatile distribution frameworks** with **logically centralised control distribution**.

V. RL Hyperparameter Tuning

In RL, as in ML, the practitioner can apply *RL Hyperparameter Tuning*, which is considered distinct from the **data processing hyperparameter tuning** decision option described in Section V-D since despite the common overall goal of optimising model performance, there are some key differences, such as the optimisation of a reward function or policy over time across multiple episodes rather than a performance metric, with distinct training runs producing a completely new model. Our case study applies hyperparameter tuning during base training, i.e. in the agent's initial training phase.

VI. Discussion

In this section, we assess, interpret and discuss the results of our rigorous qualitative case study regarding the research questions defined in Section I.

RQ1 *To what extent are known MLOps and RL ADDs applicable to RL in a CPPS context?* After analysing the system, modelling 15 units of analysis and considering 22 ADDs with 86 decision options from the literature, we discovered evidence for 21 ADDs and 37 design decisions used in a concrete, RL-based Industry 4.0 CPPS. Only one ML ADD was not applied (but could have been). The most important observation is that all ML ADDs were applicable, and all bar one were indeed applied, indicating a high level of cross-compatibility between ML and RL practices. All six RL-specific ADDs were applied in the study object, closely corresponding to our understanding of the application of RL-specific ADDs.

In practice, the only ADD out of the 16 top-level MLOps ADDs that was not applied was *AutoML*, although it is not unsuited to RL and still could have been used in specific

circumstances [36], [37]. Practitioners should evaluate whether AutoML could offer benefits in their specific use cases, particularly for tasks that do not require highly specialised models. In our case study, it was not applied, but that does not preclude its applicability in other systems.

More generally, understanding the reasons for choosing certain options and not selecting alternative options is considered outside the scope of this study. We only considered project artefacts and known and evidenced ADDS, but this does not rule out other decision options or as-yet-unknown ones to be implemented over the lifetime of the system.

RQ2 *Which ADD decision options typically used in MLOps and RL apply directly to a real-world CPPS that utilises RL?* We considered a total of 86 decision options, 63 of which were ML-related and 23 RL-specific. Out of 63 ML decision options, 22 were directly applied, 22 ML decision options were not applied (but could have been), and 11 ML decision options were not applicable (however, see RQ3 below for a discussion on the remaining eight ML decision options, which were applicable with modification). These results suggest that many existing MLOps practices can be transferred to RL-based industrial systems without modification. Out of 23 RL-specific decision options, seven were applied, 16 were not applied (but could have been), and zero were not applicable. These results indicate that the study object conformed closely to our prior knowledge of RL decision options and exhibited a high degree of compatibility and relevance regarding known practices.

Noteworthy is the varying emphasis placed on certain decision options, evident in Table I, depending on whether the system is geared more towards ML or RL. For instance, ML decision options that are not suited to RL tend to be data-centric, such as with *Automatic Data Processing* practices (e.g. use of a **data** or **ETL pipeline**), *Data Processing Pipeline Tasks* (e.g. **data extraction**, **transformation**, **preparation** and **validation**). RL-specific decision options tend to focus on *Model Architecture* (**agents** and their specific roles), *Model Training* (**centralised/decentralised training** and **execution**) and model *Distribution Strategy* (**centralised** or **distributed**). These differences are evidenced in the units of analysis and are thus relevant when planning resource allocation (e.g. developers' time and effort).

RQ3 *Is it necessary to adapt ADD decision options typically used in MLOps for use with RL in a CPPS context?* Out of 63 ML decision options, eight ML decision options were applied in the form of an RL-equivalent, that is, a version of the original option modified to suit RL-specific needs. An interesting observation is that some of these decision options include practices that at first glance may not appear to have an RL-analogue, such as **feature engineering** and *Pipeline/Orchestrator Triggers* on **new training data**.

Given that all top-level ADDs from the literature were applicable in some way, the main differences are noticeable on the level of the decision options. Categories emerge at this level of granularity regarding their RL compatibility. We derived four categories to aid understanding of the use of ML and RL ADDs in CPPSs: **inapplicable ML decision options**, **unused ML decision options**, **ML decision options with RL equivalents** and **directly applied ML decision options**. The rationale for our classification can be independently verified via Table I and our replication package [34].

1) **Inapplicable ML Decision Options** are marked with ✗ in Table I, for example **data pipeline**, **data distribution changes** and **data splitting**. In total, we identified 11 such decision options. These decision options can be disregarded early in the architectural design process since they are not compatible with RL and do not provide for an RL-adapted equivalent. Recognising inapplicable options helps practitioners focus their efforts on relevant solutions, but it is crucial to reassess these options as technology and requirements evolve regularly.

2) **Unused ML Decision Options**, denoted by ~ in Table I, include **use AutoML**, **data store** and **model packaging**. 22 such decision options were found. These decision options are compatible with RL without RL-specific modification but were unused in the study object. They represent potential opportunities for future improvements or extensions of the system. Practitioners should maintain awareness of these options and regularly reconsider their applicability as systems mature.

3) **ML Decision Options with RL Equivalents** are marked with E in Table I, for instance, **new training data**, **model building pipeline** and **data checkpoints**. Eight of these decision options were identified. These decision options may appear at first glance to be incompatible with RL and, therefore, prematurely disregarded. However, practitioners should note that they can be modified to become more suitable. The fact that all eight ML decision options were modified and used for RL highlights the need for flexibility. Practitioners should be prepared to adapt standard MLOps practices to fit the RL paradigm. This category is particularly valuable for practitioners transitioning from traditional ML to RL in industrial settings. It provides a bridge between familiar ML practices and RL-specific requirements.

4) **Directly Applied ML Decision Options** are indicated by ✓ in Table I, and include **data processing component**, **model performance degradation** and **use checkpoints**. We noted 30 such decision options. Practitioners should be aware that the MLOps-related decision options in this category are directly compatible with RLOps. They should be encouraged to use them where appropriate, adapting their MLOps practices to accommodate RL-specific requirements. The application of all 6 RL-specific ADDs indicates the importance of considering RL's unique characteristics in industrial settings. These decision options also offer the most straightforward path to implementation. Practitioners should consider prioritising them to achieve rapid benefits and establish a foundation for more complex integration later in the system's lifecycle.

Our results demonstrate that RL practitioners wishing to implement RLOps have a rich set of architectural decisions at their disposal. This breadth of choice allows for tailored solutions that can address the unique challenges of RL in Industry 4.0 CPPS environments. By carefully considering the applicability of various ADDs and decision options and addressing RL's unique challenges in industrial settings, prac-

titioners can leverage these technologies to enhance efficiency, adaptability and decision-making in their operations. Practitioners are recommended to maintain flexibility and continually reassessment options, and a commitment to ongoing adaptation is recommended in this rapidly evolving field.

VII. Threats to Validity

In this section, we consider threats to validity according to Wohlin et al. [38] and Yin [27].

A. Internal Validity

Our selection of the study object and its context (for instance, its domain), as well as its level of MLOps/RLOps maturity, could have invalidated or skewed our results. However, we consider the study object to be representative of the broader industry, and this assertion is supported by the broad range of MLOps/RLOps practices evidenced in the case study.

If a software project is still in development at the time of a study, changes over time within the study object, e.g. in team composition, system architecture, or technologies, could influence findings. We do not consider this threat significant since the project was in an advanced stage of development, and any relevant architectural decisions had already been made. In any case, this factor applies to any project since all development and production systems evolve until decommissioned. Our case study represents an accurate temporal snapshot, and it is worth noting that none of the documented practices or ADDs contradict or are incompatible with one another.

B. External Validity

The generalisability of our findings is a potential threat to external validity if the study object is not representative of the broader industry or use of MLOps/RLOps in practice – the findings may not be applicable to other contexts. Despite the novel nature of the system, we were provided with many data sources for triangulation and the study object represents the state of the art. We consider the study object a typical case for the study and in other contexts, again due to the broad range of evidenced practices discovered. We expect our results to be relevant to other cases with common characteristics and expect our findings also to be relevant to those cases. Still, it would be interesting to conduct other case studies in similar and differing contexts for comparison (see Section VIII).

C. Construct Validity

We set out to understand how MLOps ADDs are applied in practice and how they relate to RLOps. Our definitions and understanding of MLOps/RLOps ADDs might not be in concordance with their use within the study object, leading to inconsistent interpretations and less convincing results. To mitigate this threat, we clearly, consistently and comprehensively defined all technical terms as we understood them, used standardised terminology and ensured the author team and project partners had a shared understanding. We also formally modelled the relevant parts of the system and generated UML visualisations of our models for comparison, documentation and validation. Our findings indicate that our understanding of the ADDs matched their use in the study object, thus underscoring their relevance and the validity of the study.

D. Reliability

A further risk is that the methods used in our prior work described in Section IV-A to identify practices, ADDs and decision options may not have accurately captured the nuances of relevant practices in the real world. The same applies to the translation of our understanding of concepts from our prior work, such as ADDs, into concrete practical use in the case study. To mitigate this threat, we triangulated all sources of project information as a confirmatory measure and, when necessary, asked project partners for clarification. In doing so, we successfully mapped the ADDs to phenomena encountered within the study object, which confirms the overall validity of the ADDs and the reliability of the study.

VIII. Conclusion and Future Work

We explored the application of ML ADDs in a real-world RL CPPS. Our research questions examine the applicability of known ML and RL ADDs, the decision options directly applied to RL, and the adaptation requirements for ML decision options to RL in a CPPS context. This study, the first of its kind, provides actionable insights into the use of MLOps and RLOps practices in Industry 4.0. It provides guidance and a reference for practitioners, linking theoretical concepts and real-world applications in the field of RL-driven CPPSs and informing architectural design in Industry 4.0.

We categorised ADD decision options into four emergent groups: **Inapplicable ML Decision Options**, **Unused ML Decision Options**, **ML Decision Options with RL Equivalents** and **Directly Applied ML Decision Options**. This classification provides insights into applying ADDs from the literature to real-world RLOps systems. The findings show that many MLOps practices are directly applicable or adaptable to RL, contributing to a better understanding of RLOps architecture design and aiding RL practitioners in implementing effective practices. These insights may be of particular interest and encouragement to project teams when implementing similar Industry 4.0 CPPSs to the study object, particularly to those who are unaware that many MLOps practices are applicable to RL during architectural decision-making.

This study on ML and RL ADDs in the Industry 4.0 CPPS domain also lays the groundwork for future studies. Ideas include exploring RLOps ADDs across various industries, longitudinal studies on RLOps ADD evolution and performing quantitative impact analyses. Such studies could reveal sector-specific practices and new ADDs, track the maturation of RLOps practices over time, and provide empirical evidence of their effectiveness. Collectively, such studies could enhance understanding of RLOps practices in different contexts and their impact on organisational performance.

IX. Acknowledgement

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

References

[1] H. Singh and B. Singh, "Industry 4.0 technologies integration with lean production tools: a review," *The TQM Journal*, 02 2024.

[2] L. Monostori, "Cyber-physical production systems: Roots, expectations and R&D challenges," *Procedia CIRP*, vol. 17, pp. 9–13, 2014, Variety Management in Manufacturing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827114003497

[3] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals*, vol. 65, no. 2, pp. 621–641, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007850616301974

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

[5] M. Bernas, A. Mykytyshyn, V. Kartashov, V. Levytskyi, and D. Martjanov, "The role of cyber-physical systems and internet of things in development smart cities for industry 4.0," in *Congreso Internacional de Tecnologías e Innovación*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:261698719

[6] A. Khdoudi, T. Masrour, I. El Hassani, and C. El Mazgualdi, "A deep-reinforcement-learning-based digital twin for manufacturing process optimization," *Systems*, vol. 12, no. 2, 2024. [Online]. Available: https://www.mdpi.com/2079-8954/12/2/38

[7] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," *ArXiv*, vol. abs/2205.02302, 2022.

[8] P. Li, J. Thomas, X. Wang, A. Khalil, A. Ahmad, R. Inacio, S. Kapoor, A. Parekh, A. Doufexi, A. Shojaeifard *et al.*, "RLOps: Development life-cycle of reinforcement learning aided open RAN," *IEEE Access*, vol. 10, pp. 113 808–113 826, 2022.

[9] C. Paduraru, D. J. Mankowitz, G. Dulac-Arnold, J. Li, N. Levine, S. Gowal, and T. Hester, "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis," *Machine Learning*, vol. 110, pp. 2419 – 2468, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:234868359

[10] K. Höse, A. Amaral, U. Götze, and P. Peças, "Manufacturing flexibility through industry 4.0 technological concepts—impact and assessment," *Global Journal of Flexible Systems Management*, vol. 24, pp. 271–289, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258150285

[11] J. DeFranco and P. Laplante, "A content analysis process for qualitative software engineering research," *Innovations in Systems and Software Engineering*, vol. 13, pp. 1–13, 09 2017.

[12] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. IEEE, 2005.

[13] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, "Do architectural design decisions improve the understanding of software architecture? two controlled experiments," *Journal of Systems and Software*, vol. 94, pp. 311–327, 2014.

[14] S. Stevanetic, K. Plakidas, T. B. Ionescu, F. Li, D. Schall, and U. Zdun, "Tool support for the architectural design decisions in software ecosystems," *Proceedings of the 2015 European Conference on Software Architecture Workshops*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:1144013

[15] J. Hao, T. Yang, H. Tang, C. Bai, J. Liu, Z. Meng, P. Liu, and Z. Wang, "Exploration in deep reinforcement learning: From single-agent to multiagent domain," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[16] H. M. Schwartz, *Multi-agent machine learning: A reinforcement approach*. John Wiley & Sons, 2014.

[17] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[18] N. Liyanawaduge, E. Kumarasinghe, S. S. Iyer, A. K. Kulatunga, and G. Lakmal, "Digital twin & virtual reality enabled conveyor system to promote learning factory concept," *2023 IEEE 17th International Conference on Industrial and Information Systems (ICIIS)*, pp. 85–90, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:262076517

[19] V. Havard, B. Jeanne, M. Lacomblez, and D. Baudry, "Digital twin and virtual reality: a co-simulation environment for design and assessment of industrial workstations," *Production & Manufacturing Research*, vol. 7, pp. 472 – 489, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202789864

[20] A. del Real Torres, D. S. Andreiana, Á. Ojeda Roldán, A. Hernández Bustos, and L. E. Acevedo Galicia, "A review of deep reinforcement learning approaches for smart manufacturing in industry 4.0 and 5.0 framework," *Applied Sciences*, vol. 12, no. 23, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/23/12377

[21] E. Commission, D.-G. for Research, Innovation, M. Breque, L. De Nul, and A. Petridis, *Industry 5.0 – Towards a sustainable, human-centric and resilient European industry*. Publications Office of the European Union, 2021.

[22] T. Kegyes, Z. Süle, and J. Abonyi, "The applicability of reinforcement learning methods in the development of industry 4.0 applications," *Complex.*, vol. 2021, pp. 7 179 374:1–7 179 374:31, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:246751425

[23] Y. Zhou, Y. Yu, and B. Ding, "Towards MLOps: A case study of ML pipeline platform," in *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, Oct 2020, pp. 494–500.

[24] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, 1st ed. Addison-Wesley Professional, 2015.

[25] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[26] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:207144526

[27] R. K. Yin, *Case study research and applications : design and methods*, 6th ed. Los Angeles London New Delhi Singapore Washington, DC Melbourne: SAGE, 2018.

[28] S. J. Warnett and U. Zdun, "Architectural design decisions for machine learning deployment," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, 2022, pp. 90–100.

[29] ——, "Architectural design decisions for the machine learning workflow," *Computer*, vol. 55, no. 3, pp. 40–51, 2022.

[30] E. Ntentos, S. J. Warnett, and U. Zdun, "Supporting architectural decision making on training strategies in reinforcement learning architectures," in *2024 IEEE 21st International Conference on Software Architecture (ICSA)*, 2024, pp. 90–100.

[31] J. Corbin and A. L. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociology*, vol. 13, pp. 3–20, 1990.

[32] S. J. Warnett and U. Zdun, "On the understandability of MLOps system architectures," *IEEE Transactions on Software Engineering*, vol. 50, no. 5, pp. 1015–1039, 2024.

[33] S. J. Warnett, E. Ntentos, and U. Zdun, "A model-driven, metrics-based approach to assessing support for quality aspects in MLOps system architectures," *Journal of Systems and Software*, vol. 220, p. 112257, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121224003017

[34] S. J. Warnett and U. Zdun, "Bridging the gap between MLOps and RLOps: An Industry 4.0 case study on architectural design decisions in practice: Replication package," Jan. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.14722767

[35] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *ArXiv*, vol. abs/2005.01643, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218486979

[36] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, F. Hutter, and M. Lindauer, "Automated reinforcement learning (AutoRL): A survey and open problems," *J. Artif. Int. Res.*, vol. 74, Sep. 2022. [Online]. Available: https://doi.org/10.1613/jair.1.13596

[37] X. Wang, B. Li, Y. Zhang, B. Kailkhura, and K. Nahrstedt, "Robusta: Robust AutoML for feature selection via reinforcement learning," *ArXiv*, vol. abs/2101.05950, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231627788

[38] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.