

THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

**Towards systematic trade-off  
management for MLOps: quality model,  
architectural tactics, design patterns**

VLADISLAV INDYKOV

*Department of Computer Science and Engineering*  
UNIVERSITY OF GOTHENBURG  
Gothenburg, Sweden, 2025

# **Towards systematic trade-off management for MLOps: quality model, architectural tactics, design patterns**

VLADISLAV INDYKOV

© Vladislav Indykov, 2025  
except where otherwise stated.  
All rights reserved.

ISSN 1652-876X

Department of Computer Science and Engineering  
Division of Interaction Design and Software Engineering  
University of Gothenburg  
SE-412 96 Göteborg,  
Sweden  
Phone: +46(0)31 772 1000

Printed by Chalmers Digitaltryck,  
Gothenburg, Sweden 2025.

# Towards systematic trade-off management for MLOps: quality model, architectural tactics, design patterns

VLADISLAV INDYKOV

*Department of Computer Science and Engineering  
University of Gothenburg*

## Abstract

Machine-learning-enabled systems are booming within modern software-intensive domains, driving innovation in areas such as healthcare, finance, and autonomous systems. To produce high-quality and competitive solutions, the development and operation of ML-enabled systems (MLOps) require careful consideration of quality trade-offs across the entire production cycle. These trade-offs often differ from well-studied ones relevant to traditional software due to the inherently probabilistic and data-dependent nature of machine learning. Such differences particularly challenge startups and SMEs, who must operate in a non-standardized domain and seek frameworks that capture best practices for producing ML-enabled software.

In this thesis, we propose a framework for embedding systematic trade-off management into the MLOps lifecycle. Firstly, based on the results of a systematic literature review combined with an evaluation in industrial settings, we built a common quality model unique to ML-enabled systems. Second, through a systematic literature review and multiple-case study with four companies in the AI domain, we derived architectural and non-architectural tactics employed to achieve identified quality attributes. Third, we extracted existing design patterns from the component models of ML-enabled software identified through a multi-vocal literature review, and, using semi-structured expert interviews, evaluated their impact on quality attributes. In combination, these three studies lead to the insight that applying tactics or patterns to improve one quality attribute usually has side effects on other attributes, resulting in quality trade-offs.

Our findings reveal that trade-off management is crucial for ML software production, as it significantly influences decision-making at all dimensions of the MLOps lifecycle (data, model, operations, and development). Based on these insights, as a fourth and final contribution, we propose a vision of an extension to the overall MLOps paradigm by embedding explicit steps for trade-off management at all phases of the workflow.

## Keywords

Machine Learning Engineering, Trade-off Management, Software Architecture, MLOps



# Acknowledgment

I would like to express my deepest gratitude to my supervisors, Dr. Daniel Strüber, Dr. Rebekka Wohlrab, and Dr. Philipp Leitner. First of all, I would like to thank them for selecting me for the position of *doktorand*. This decision changed my life, as I have been able to spend wonderful years (without exaggeration, the best years of my life) in the country of my dreams. Not knowing what lies ahead, I firmly know that thanks to these wonderful people, at least for the duration of my studies, I have found a place that feels like Home for the first time in my life and this cannot be exhaustively described with any words of gratitude. Of course, I am sincerely grateful for their constant help and support throughout my Licentiate journey. Their extensive expertise in the research area, combined with their guidance and encouragement, has been invaluable in helping me acquire knowledge, develop skills, and master diverse research methodologies. Additionally, I warmly appreciate their moral support in the hardest times of my journey, caused by non-research-connected circumstances. With their encouraging words and supportive deeds, I was able to find motivation to continue the research even in the darkest times. Daniel, Rebekka, and Philipp instilled in me a love for science and research, and that love grows stronger every year. I am so glad that fate brought me together with these beautiful people, and I want our connection to extend far beyond my doctoral studies!

Furthermore, I would like to express my sincere gratitude to my examiner, Prof. Christian Berger, for his valuable feedback and the expertise he invested in this research. His constructive comments and thoughtful suggestions, without doubt, strengthened the quality of this thesis. I am warmly grateful for his support during the hardest times of my journey. His empathetic and compassionate attitude towards me as a PhD student allowed me to stay motivated and enthusiastic even in very stressful, non-research-connected circumstances I had to operate in.

I would also like to express my warm gratitude to the discussion leader, Prof. Manfred A. Jeusfeld, for his genuine interest in this research and his invaluable involvement in the Licentiate seminar.

I would also like to extend my special appreciation to the members of the doctoral school for their constructive feedback on operational and formal matters and their valuable administrative assistance. Additionally, I would like to thank all my colleagues in the IDSE division for the pleasant and friendly

environment they collectively created. It is a great pleasure of mine to be a part of IDSE.

I would like to thank all the Swedish industry representatives who took part in this research. I deeply appreciate the time, effort, and valuable insights they contributed, which have fundamentally enriched this work. I would like to thank Vetenskapsrådet for funding this research, and I sincerely hope that I am managing to meet the expectations with my thesis.

I would like to express my sincere appreciation to all the students I had the pleasure of working with as a supervisor, examiner, or teacher. I wish each of them great success in their careers, further studies, and creative pursuits.

I am also deeply grateful to the reviewers and editors of the journals and conferences to which I have submitted my studies. Their careful consideration of the findings and invaluable feedback have contributed significantly to my growth as a scientist and researcher.

I would like to wholeheartedly thank my wife, Vlada Sokolovskaia, for her unwavering support and for always being by my side throughout my studies. She has truly been my guiding light. I am equally grateful to my parents and relatives for their constant encouragement and their unwavering faith in me during the most challenging moments of my life. Their support has been a profound source of strength throughout my journey.

Finally, I would like to thank beautiful Sweden for welcoming me with warmth and care. This country and its people gave me the most precious thing a person could ever dream of. It is the feeling of being at home, right where I belong.

Last but not least, I want to thank everyone who is reading this thesis. Your attention to and interest in our findings is the biggest reward we could wish for.

# List of Publications

## Appended publications

This thesis is based on the following publications:

- [**Paper I**] **V. Indykov**, *Component-based Approach to Software Engineering of Machine Learning-enabled Systems*  
*Proceedings of the 3rd IEEE/ACM International Conference on AI Engineering (CAIN), 2024, pp. 250-252.*
- [**Paper II**] **V. Indykov**, D. Strüber, R. Wohlrab, *Architectural Tactics to Achieve Quality Attributes of Machine-Learning-Enabled Systems: a Systematic Literature Review*  
*Journal of Systems & Software (JSS), 2025, vol. 223, no. 112373, pp. 1-20.*
- [**Paper III**] **V. Indykov**, R. Wohlrab, D. Strüber, *Quality Trade-Offs in ML-Enabled Systems: a Multiple-Case Study*  
*Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing (SAC), 2025, pp. 1730 - 1737.*
- [**Paper IV**] E. Eriksson, J. Olausson, **V. Indykov**, D. Strüber, R. Wohlrab, *Extracting Design Patterns from Mined Component Models of ML-Enabled Systems*  
*Proceedings of the 51th Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA), 2025, pp. 113-129.*
- [**Paper V**] **V. Indykov**, D. Strüber, R. Wohlrab, *MLTradeOps: Embedding Trade-Off Management into the MLOps Workflow*  
*Proceedings of the 51th Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA), 2025, pp. 97-112.*

## Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

- [a] V. Campanello, S. Shahbaz, **V. Indykov**, D. Strüber, *On the Use of GPT-4 in the Reverse Engineering of Class Diagrams*  
*Journal of Object Technology (JOT)*, 2025, vol. 24, no. 2, pp. 1-14.

# Personal Contribution

I was the main driver and contributor of papers *I*, *II*, *III*, and *V*. Also, I have significant contributions to paper *IV*. My contributions in these papers are classified according to the Contributor Roles Taxonomy (CRediT) presented in Table 1.

Table 1: Personal contributions in the papers included in this Thesis (CRediT)

Role	Paper I	Paper II	Paper III	Paper IV	Paper V
Conceptualization	X	X	X	X	X
Data curation	X	X	X		X
Formal analysis	X	X	X	X	X
Funding acquisition					
Investigation	X	X	X		X
Methodology		X	X	X	X
Project administration					
Resources					
Software					
Supervision				X	
Validation		X	X	X	X
Visualization	X	X	X	X	X
Writing – original draft	X	X	X	X	X
Writing – Review Editing	X	X	X	X	X

In *Paper I*, I formulated the primary goal and objectives of the ongoing PhD research; investigated related work in the field, and stated the main research problem and main research hypothesis. I planned three main contributions to be delivered by the ongoing PhD research and aligned all the content of this paper with the primary expectations of my PhD project (SEMLA: Software Engineering for Machine Learning - Integrated Approach, funded by Vetenskapsrådet). I created the first draft of the paper and revised it iteratively according to the valuable feedback of my supervisors. I presented this paper at the CAIN doctoral symposium in 2024.

In *Paper II*, I conducted a systematic literature review on the existing quality attributes characterizing ML-enabled systems, architectural tactics employed to achieve them, and quality trade-offs arising after the implementation of such tactics. I designed data collection, data synthesis, and data validation strategies, considering detailed feedback from my scientific supervisors. I conducted several interviews to validate our findings in the industrial settings,

investigated threats to the validity of this study, and formulated implications for researchers and practitioners. I created the first draft of the paper and revised it iteratively according to the valuable feedback of my supervisors.

In *Paper III*, I conducted multiple interviews with ML developing companies to identify architectural and non-architectural tactics they regularly employ and quality trade-offs they frequently encounter. I made connections with the interview candidates at the GAIA conference in 2024 and designed case selection, data collection, and data extraction strategies, considering detailed feedback from my scientific supervisors. I investigated threats to the validity of this study and formulated implications for researchers and practitioners. I created the first draft of the paper and revised it iteratively according to the valuable feedback of my supervisors. I presented this paper at the Symposium on Applied Computing (SAC) in 2025.

The concept of *Paper IV* was first suggested by me and Daniel Strüber as an idea for the MSc thesis. I, together with Daniel Strüber and MSc students E. Eriksson and J. Olausson, designed the methodology for this study. I, together with Daniel Strüber, provided supervision to this MSc thesis project and conducted regular validation of the findings obtained. When the thesis was successfully defended, I took responsibility, with the agreement of the MSc students, to transform this thesis into a scientific paper. In this process, I had to conduct a formal analysis of the findings and create new visualizations to represent them graphically. Based on the content of the MSc thesis, I wrote the first draft of the paper and revised it iteratively according to the valuable feedback of my supervisors. I supported the presentation of this paper at the Conference on Software Engineering and Advanced Applications (SEAA) in 2025.

In *Paper V*, I formulated the vision of the MLTradeOps framework, embedding trade-off management into the MLOps workflow. I analyzed the evolution of DevOps, described the vision in detail, and built a roadmap for the creation of this framework. I wrote the first draft of the paper and updated it iteratively according to the valuable feedback of my supervisors. I presented this paper at the Conference on Software Engineering and Advanced Applications (SEAA) in 2025.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgement</b>	<b>iii</b>
<b>List of Publications</b>	<b>v</b>
<b>Personal Contribution</b>	<b>vii</b>
<b>I Summary</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Background and Related Work . . . . .	5
1.1.1 Machine Learning Engineering . . . . .	5
1.1.2 Software Quality for Machine Learning . . . . .	7
1.1.3 Trade-Off Management . . . . .	8
1.1.4 Machine Learning Operations (MLOps) . . . . .	8
1.2 Methodology . . . . .	9
1.2.1 Systematic Literature Review . . . . .	9
1.2.2 Multi-Vocal Literature Review . . . . .	11
1.2.3 Semi-Structured Interviews . . . . .	11
1.2.4 Multiple-Case Study . . . . .	12
1.2.5 Vision . . . . .	12
1.3 Results and Evaluation . . . . .	12
1.3.1 Software Quality of ML-enabled Systems . . . . .	13
1.3.2 Software Architecture of ML-enabled Systems . . . . .	15
1.3.3 Vision for the MLOps framework . . . . .	20
1.4 Answers to RQs . . . . .	23
<b>2 Discussion</b>	<b>25</b>
2.1 Implications for researchers . . . . .	25
2.2 Implications for practitioners . . . . .	26
2.3 Threats to Validity . . . . .	26
2.3.1 External Validity . . . . .	26
2.3.2 Internal Validity . . . . .	27
2.3.3 Construct Validity . . . . .	27

2.4 Conclusion and Future Work . . . . .	28
<b>Bibliography</b>	<b>29</b>
<b>II Appended Papers</b>	<b>57</b>
<b>Paper I - Component-based Approach to Software Engineering of Machine Learning-enabled Systems</b>	
1 Introduction	2 (I)
2 Contributions	4 (I)
3 Conclusion	6 (I)
<b>Paper II - Architectural Tactics to Achieve Quality Attributes of Machine-Learning-Enabled Systems: a Systematic Literature Review</b>	
1 Introduction	2 (II)
2 Background	4 (II)
3 Methodology	6 (II)
4 Results	14 (II)
5 Discussion	33 (II)
6 Conclusion	38 (II)
<b>Paper III - Quality Trade-Offs in ML-Enabled Systems: a Multiple- Case Study</b>	
1 Introduction	2 (III)
2 Background	4 (III)
3 Related Work	5 (III)
4 Methodology	7 (III)
5 Results	10 (III)
6 Discussion	18 (III)
7 Conclusion	20 (III)

---

**Paper IV - Extracting Design Patterns from Mined Component Models of ML-Enabled Systems**

<b>1</b>	<b>Introduction</b>	<b>2 (IV)</b>
<b>2</b>	<b>Background</b>	<b>4 (IV)</b>
<b>3</b>	<b>Related Work</b>	<b>8 (IV)</b>
<b>4</b>	<b>Methodology</b>	<b>11 (IV)</b>
<b>5</b>	<b>Design Patterns</b>	<b>19 (IV)</b>
<b>6</b>	<b>Quality Attribute Impacts</b>	<b>31 (IV)</b>
<b>7</b>	<b>Discussion</b>	<b>40 (IV)</b>
<b>8</b>	<b>Conclusion</b>	<b>44 (IV)</b>

**Paper V - MLTradeOps: Embedding Trade-Off Management into the MLOps Workflow**

<b>1</b>	<b>Introduction</b>	<b>2 (V)</b>
<b>2</b>	<b>Background</b>	<b>5 (V)</b>
<b>3</b>	<b>Vision</b>	<b>8 (V)</b>
<b>4</b>	<b>Road map</b>	<b>12 (V)</b>
<b>5</b>	<b>Discussion</b>	<b>17 (V)</b>
<b>6</b>	<b>Conclusions</b>	<b>19 (V)</b>



Part I

Summary



# Chapter 1

## Introduction

Machine-learning-enabled (ML-enabled) systems are currently in high demand across a wide range of domains. The development and operation of such systems are widespread now since machine learning (ML) technologies allow organizations to reach results that are difficult to achieve through traditional rule-based solutions. For example, in healthcare, ML models can analyze medical images to detect tumors [1] with accuracy comparable to human experts; in finance, ML models detect fraudulent transactions in real-time by identifying dynamic patterns and correlations that rule-based models may overlook [2]; retail and manufacturing use ML abilities to deliver personalized recommendations and predictive maintenance. Moreover, the adoption of machine learning significantly stimulates breakthrough innovations [3]. For example, the invention of self-driving cars has been made possible largely due to advances in ML by processing the complex environments and behaviors.

Initially, the use of ML was already a solid competitive advantage for companies in the market. Early adoption of machine learning allowed them to extract actionable insights from massive datasets before competitors. However, nowadays, with the wide use of ML, the competition in the market is dynamically intensifying. As a result, companies are forced to enhance the quality of their ML-related products to remain competitive. This competitive pressure is especially pronounced in software production, where the performance and trustworthiness of ML-enabled systems directly affect business outcomes.

The pursuit of quality is usually aligned with trade-offs. Achieving certain quality attributes within software development and operation usually implies sacrificing others. For example, improving the accuracy of machine learning models usually requires significant computational resources, negatively affecting the overall resource efficiency of the system, which can be prohibitively expensive in cases that require frequent re-training and model improvement to respond to new trends [4]. These trade-offs often differ from well-studied ones relevant to traditional software due to the inherently probabilistic and data-dependent nature of machine learning, other acceptable levels of uncertainty for the final results, and specific procedures of model training and testing, involving unique quality attributes specific to ML-enabled systems, unique tactics and design

decisions that might cause these trade-offs, and unique techniques and strategies for their balancing.

The specific nature of ML creates challenges for all the companies involved in the production of ML-enabled software due to the lack of systematic exploration of common development practices and widely adopted standards in comparison with the production of traditional systems. However, this challenge has the most critical impact on start-ups and small-to-medium enterprises (SMEs). While large companies often have the advantage of internally established workflows and mature pipelines that leverage robust development practices, start-ups and SMEs typically face resource constraints and limited access to recent insights in the field. As a result, these companies struggle to build an efficient workflow, ensuring a consistently high quality of their products.

This issue is partly addressed by the emergence of machine learning operations (MLOps). MLOps is a paradigm, including best practices, sets of concepts, and principles of development culture in end-to-end conceptualization, implementation, monitoring, deployment, and scalability of machine learning products [5]. With the help of MLOps, start-ups and SMEs received access to the recent experience in the development of ML-enabled systems to optimize their workflow; however, to obtain competitiveness in the market, such companies are still forced to be oriented on quality and, therefore, consider quality trade-offs. While the existing MLOps paradigm includes trade-off management in separate phases, it does not make their consideration fundamental for decision-making and does not provide actionable insights to achieve certain quality attributes. This limitation leads to the issue that, in certain cases, the final product may not be optimal or even unsatisfactory from the perspective of certain quality characteristics. For example, if the MLOps team never balanced their decisions with ethical considerations, the final product may fail to meet the expectations of stakeholders and society [6]. Because of this, in worst-case scenarios, start-ups and SMEs cannot withstand competition and close before they even fully enter the market [7].

In this thesis, we propose a framework for embedding systematic trade-off management into the MLOps lifecycle. To this end, first, based on the results of a systematic literature review, we built a common quality model unique to ML-enabled systems. We investigated the most frequently mentioned quality attributes of such systems (including subsets such as deep learning systems) in scientific literature and created a two-level quality model using taxonomic analysis. Further, we evaluated the resulting quality model using semi-structured interviews with several industry representatives. Second, through a systematic literature review and multiple-case study with four companies in the AI domain, we derived architectural and non-architectural tactics employed to achieve the attributes from the quality model and examined the quality trade-offs they impose. Third, through a multi-vocal literature review, we collected the component models of existing ML-enabled software systems and, using content analysis, we derived widespread design patterns from them. Using semi-structured expert interviews, we evaluated their impact on the attributes from the quality model. In combination, these three studies led us to the insight that applying tactics or patterns to improve one quality attribute usually has

significant side effects on other attributes, resulting in quality trade-offs at all dimensions of the MLOps cycle (data, model, operations, and development). Based on this insight, as a fourth and final contribution, we developed a vision of an extension to the overall MLOps paradigm by embedding explicit steps for trade-off management to balance phase-specific, product-level, and meta-level trade-offs.

To guide our research, we address the following research questions in this thesis:

**RQ1:** *What quality attributes are the most common for ML-enabled systems?*

By answering this research question, we aimed to identify the most frequently reported quality attributes for ML-enabled systems and build a quality model common to all types of ML-enabled software.

**RQ2:** *How do widespread architectural tactics affect quality attributes of ML-enabled systems?*

By answering this research question, we aimed to explore the architectural tactics applied to achieve certain quality attributes and analyze their side-effects (positive, negative, ambivalent) on all the quality attributes identified previously, providing a broad mapping of quality trade-offs.

**RQ3:** *How do widespread design patterns affect the quality attributes of ML-enabled systems?*

By answering this research question, we aimed to identify existing design patterns in ML-enabled systems and analyze their impact (positive, negative, ambivalent) on all the quality attributes identified previously, thereby extending the mapping of quality trade-offs.

**RQ4:** *How can systematic trade-off management be embedded into the existing MLOps paradigm?*

By answering this research question, we aimed to propose a vision on how the identified trade-off management issues could be solved within the existing MLOps paradigm.

## 1.1 Background and Related Work

In this section, we introduce background and related work in machine learning engineering, software quality for machine learning, MLOps, and trade-off management.

### 1.1.1 Machine Learning Engineering

Machine learning engineering is an emerging field of research that lies at the intersection of software engineering and data science. A great interest in this area from the scientific community and industries has been observed in recent years due to the dynamic spread of ML applications in various fields. According to Burkov et al. [8], “Machine learning engineering (MLE) is the use of scientific principles, tools, and techniques of machine learning and traditional software engineering to design and build complex computing systems”. In

modern MLE, a current trend is to integrate ML functionality into complex systems as architectural components [9]. This trend aligns the existing MLE with fundamental principles of component-based software engineering (CBSE). According to Heineman et al. [10], CBSE is “the disciplined process of building software systems from prefabricated software components”, whereas a software component is “a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard”. Following this trend, many industries, especially less mature and expertized, face several obstacles, such as the issues of versioning and reproducibility due to the changing behavior of machine learning models after retraining, issues of integration and interoperability due to complex data structures required by ML components (while traditional CBSE usually relies on well-defined interfaces), unique testing and validation strategies due to probabilistic nature of ML models and many others. As a result, these issues usually lead to complicated challenges, affecting the overall quality of delivered products and the efficiency of the development process.

To investigate the challenges affecting the quality of delivered products, it is essential to examine them from the angle of software architecture, since the integration of ML components is directly related to the design decisions made at the architectural level and the system qualities these decisions affect. According to Bass et al. [11]: “The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them”.

Nowadays, we are witnessing the dynamic growth of scientific activity at the intersection of MLE, CBSE, and software architecture. For example, Lewis et al. [12] explored software architecture challenges for ML-enabled systems, Washizaki et al. [13] investigated the architecture of ML-enabled software via a systematic literature review, and Serban et al. [14] studied the adaptation of software architectures to machine learning and the issues associated with it. The researchers identify the wide spectrum of challenges implicitly connected to certain system quality attributes, and the majority of these challenges remain unsolved and relevant to this day.

While the architectural challenges are relatively well-studied, the ways of solving them require extra investigation. One of the ways to address such challenges may be the implementation of design patterns. According to Gamma et al. [15], design patterns are “general solutions to recurring problems in software”. The SWEBoK [16] defines design patterns as “concepts describing both higher-level architecture and organization of code as well as lower-level design and implementation details”. Design patterns have been investigated in software engineering for more than four decades [17]. Within AI specifically, design patterns for responsible AI (i.e., the ability of AI to behave ethically and responsibly) [18] and multi-agent systems (i.e., a subfield of AI involving the interaction of intelligent agents) [19] have also been studied for latest decade. There have been studies on design patterns for the Internet of Things [20] and microservices [21]. However, by the time of this research, only a few secondary studies on design patterns for ML-enabled systems were produced.

Washizaki et al. conducted a multivocal literature review to identify patterns for ML architecture and design, which they claimed was the first of its kind [22]. In total, they found 33 patterns and anti-patterns, with only two of them described in detail. Later, Washizaki et al. continued the work on these patterns and narrowed them down to 15, removing those that were vaguely defined or had questionable usefulness [23]. The identified patterns are either high-level architectural patterns (e.g., “Distinguish Business Logic from ML Models”) or more design-oriented ones (e.g., “Data Lake”). In 2023, Heiland et al. conducted an MLR with a similar purpose - identifying design patterns for ML systems [24]. They identified 70 patterns, 36 of which they classified as “traditional patterns”, i.e., patterns that can be found in non-AI contexts but have been adapted to an AI context, while the rest remained exclusively ML-specific. All patterns were categorized into different groups, for example, deployment, implementation, or security and safety. The category with the majority of patterns was architecture, with 25 of them, which, according to the authors, adds to the current understanding of the importance of architecture and its reuse.

The design patterns from related studies made a significant contribution to the investigation of solutions to the open challenges of MLE. At the same time, they were not aligned with the principles of CBSE. Therefore, we decided to employ another approach and analyze the software architecture of existing ML-enabled systems and use their component models as the main source for deriving design patterns. According to Crnković [25], “a component model defines standards for properties that individual components must satisfy and methods for composing such components”. The results we obtained were different from the ones in related studies and were further compared to them, as presented in *Paper IV*.

To make the use of design patterns more informed and justified, we decided to evaluate their impact on all main quality attributes relevant to ML-enabled systems. This task appeared to be quite complicated due to the lack of structured and well-established quality models, considering the specifics of ML, at the time of conducting this research. It motivated us to devote a dedicated investigation of software quality for ML-enabled systems.

### 1.1.2 Software Quality for Machine Learning

The study of software quality for ML-enabled systems is an in-demand topic among researchers and practitioners [14], [26]. The space for interpreting the quality of these systems has only been partially explored, and a conclusive view is yet to form, which is proved by the emergence of different quality models based on industrial experience [27]–[29]. While such studies significantly contributed to the exploration of software quality for ML-enabled systems, they worked with non-functional requirements relevant to a certain system and most often received them from domain experts. The generalizability of such models can be debatable due to context dependence. Their systematization and the identification of the most common quality attributes provide a way to build a more generalized picture based on real examples. Such a strategy

supports a collection of the most recent materials and makes current research more independent from external inputs.

According to Bass et al. [11], a quality attribute is “a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders”. Required quality attributes can be achieved by the application of dedicated techniques commonly known as architectural tactics (ATs) [11]. While ATs are a well-established concept in software architecture [11], they are rarely emphasized in the context of ML-enabled systems. Instead, there are plenty of review papers on architectural design decisions in this context [30]–[32]. These studies explore a collection of decisions without a clear reference to system qualities or with a focus on one or two selected quality attributes and their metrics in isolation from the overall quality picture of the system. As a result, the potential negative impacts of decisions on quality attributes, other than the ones they were initially aimed at, remain unnoticed. Therefore, a vast number of trade-offs are ignored, which in the end can lead to significant expenses on mitigating negative effects further or even to the failure of the project. To address this issue, the implementation of systematic trade-off management is essential.

### 1.1.3 Trade-Off Management

According to Berander et al. [33], quality trade-off is a “situation, where improving one software quality attribute or achieving one goal comes at the expense of another attribute or goal”.

Existing research actively explores quality trade-offs arising from the implementation of selected decisions in ML-enabled systems. For instance, between energy efficiency and system accuracy [4], performance and interpretability [34], performance and privacy [35]. While such studies provide deep investigations of specific cases, they leave aside the identification of the most widespread trade-offs or all the trade-offs the decision may cause aside and focus on one particular trade-off. Therefore, the landscape of quality trade-offs remains largely unmapped, requiring significant systematic investigation.

The identification of trade-offs and the selection of techniques and strategies for their balancing is called trade-off management. In order to balance trade-offs on different levels of abstraction and during different stages of software production, such a process must be integrated into all operational phases. As this thesis will explain, in the context of ML, such a process should be integrated into machine learning operations (MLOps) due to its broad use in companies of different maturity levels and across a wide range of domains.

### 1.1.4 Machine Learning Operations (MLOps)

The broadest definition of MLOps is given by Kreuzberger et al. [5], who described it as “a paradigm, including aspects like best practices, sets of concepts, as well as a development culture when it comes to the end-to-end conceptualization, implementation, monitoring, deployment, and scalability of machine learning products”. This definition covers not only the production of

ML pipelines but also the production of ML-enabled systems in general. We note that the selection of an MLOps definition is important for understanding its scope and the challenges it is intended to address, since there are a lot of interpretations of this paradigm proposed by different researchers. In other words, according to the selected definition, MLOps explores not only the behavior and characteristics of the model and data flows but also monitors the decisions and their consequences within the overall system architecture and production workflow. Wide adoption of MLOps, especially among SMEs and start-ups, motivated us to build our vision based on the fundamental concepts of this paradigm.

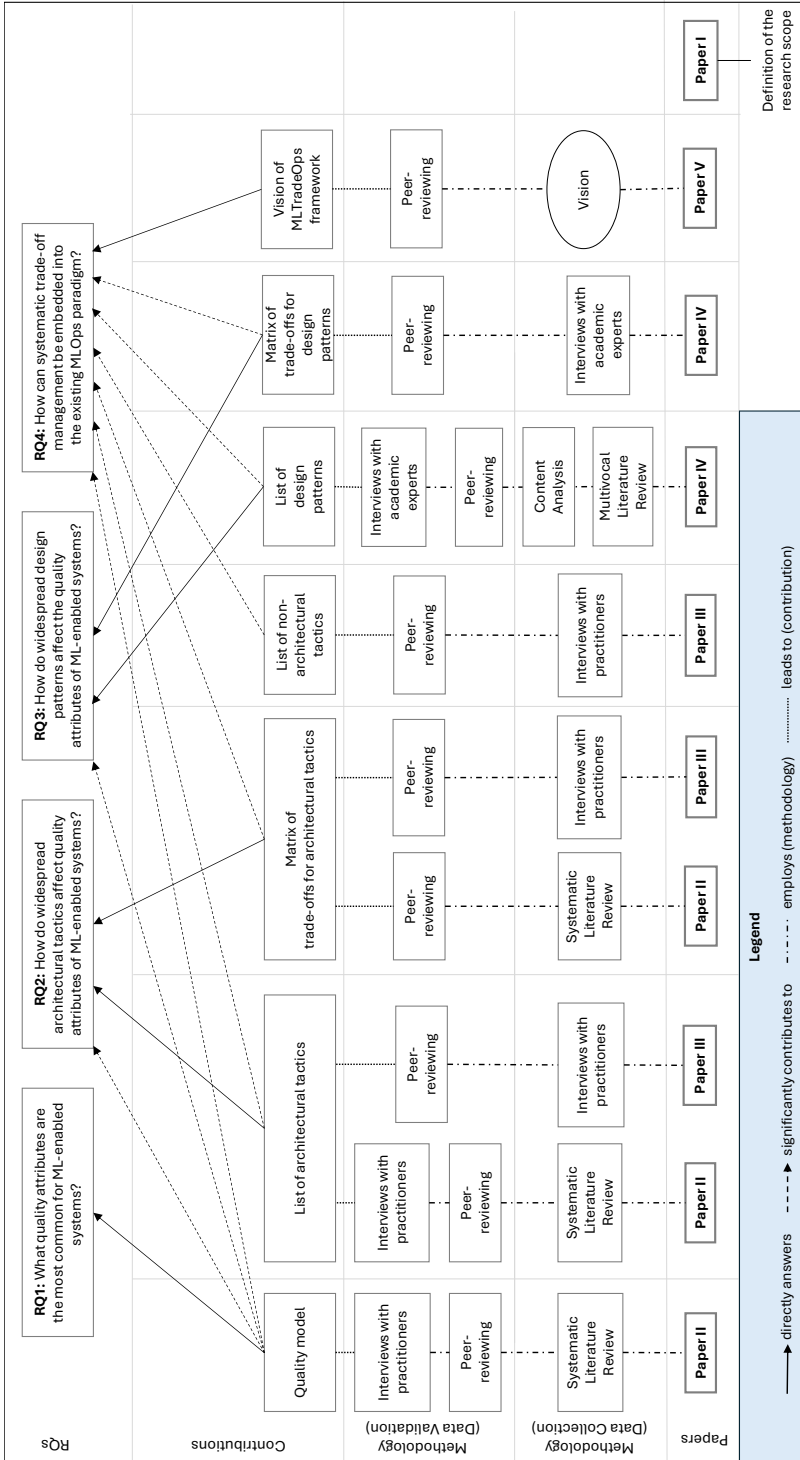
## 1.2 Methodology

Figure 1.1 depicts all the studies included in this thesis, the research questions they addressed, and the methodologies they involved. In our research, we employed a variety of empirical methods to gain diverse perspectives on the investigated phenomena, thereby ensuring methodological triangulation and strengthening the validity of the findings. Taken together, the combination of methodologies employed constitutes a *mixed-method study* [36], which represents a distinct methodological design aimed at a multi-angled exploration of the phenomena.

### 1.2.1 Systematic Literature Review

We employed the methodology of systematic literature review to identify the most widespread quality attributes of ML-enabled systems, commonly used architectural tactics to achieve identified quality attributes, and the impacts each tactic has on all identified attributes. According to Kitchenham et al. [37], a systematic literature review is a means of identifying, evaluating, and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest in a systematic, transparent, and replicable manner. The methodology of systematic literature review (SLR) allowed us to work with a large amount of scientific information, find common approaches to different systems, and effectively extract information from different scientific sources. In our research, we followed the guidelines by Kitchenham et al. [37] as we found them the most applicable to the software engineering field and the most detailed and structured ones. According to these guidelines, we built data collection, data extraction, and data synthesis strategies. We created and regularly updated the review protocol (the main formal document confirming the consistency of review according to Kitchenham et al. [37]). All the details regarding the application of SLR methodology to current research can be found in *Paper II*. The artifacts obtained during the systematic literature review (such as a list of studied literature and data extraction sheets) are publicly available and can be found in the supplementary artifact to *Paper II*.

Figure 1.1: Overview of contributions and applied methodologies



### 1.2.2 Multi-Vocal Literature Review

We employed the methodology of multivocal literature review to collect component models of existing ML-enabled systems to further derive design patterns from them. The Multivocal Literature Review (MLRs) is a specific type of systematic literature review where the grey literature is included along with white literature. It gives the benefits of introducing the experience of both researchers and practitioners [38]. In a rapidly growing area of research, the inclusion of grey literature in a review can be highly beneficial [39]. It also has the advantage of bringing research and practice closer together, which is especially important in areas with high practitioner interest or a lack of corroboration between research and practice. Garousi et al. [38] have created a set of guidelines for conducting MLRs in software engineering, which was used as an extension to Kitchenham et al.'s guidelines for conducting systematic literature reviews.

The most widely accepted definition of grey literature was the Luxembourg definition proposed by Cooper et al. [40]. Further, an updated Prague definition [41] was established that adjusts the definition in regards to publishing on the web. In the guidelines for conducting MLRs in software engineering, Garousi et al. [38] also highlight the existence of multiple definitions of grey literature and imply to use of an adapted version of a model by Adams et al. [42]. This model is built on tiers of outlet control and credibility. However, Kitchenham et al. point out that these tiered models are “largely defined by example, which is a weak method of definition” [37]. They instead suggest using the Prague definition because it stipulates that grey literature is collected and preserved, making the systematic review reproducible.

However, this definition might lead to potentially valuable sources being neglected, which would not be in line with our goal of determining patterns used in existing component models from the literature, which is best solved by looking at both research and practice. Henceforth in this study, grey literature refers to the following definition, given by Garousi et al. in a more recent publication [41], whereas “grey literature is any material about software engineering that is neither formally peer-reviewed nor formally published”.

### 1.2.3 Semi-Structured Interviews

To verify the previously created quality model, previously collected architectural tactics and trade-offs, as well as to evaluate the derived design patterns and their connections to quality attributes, semi-structured interviews with practitioners and academic experts, respectively, were planned and conducted. According to Hove et al. [43], a semi-structured interview is the type of interview in which the interviewer follows a pre-defined set of themes or questions but is free to explore topics in more depth as they arise during the conversation. The semi-structured interviews give the right balance between freedom and limitations when it comes to the questions addressed. During the planning and conducting of the interviews, recommendations for semi-structured interviews in empirical software engineering research by Hove et al.[43] were followed.

### 1.2.4 Multiple-Case Study

To investigate quality trade-offs in ML-enabled systems in the real industry, as well as to extend the list of architectural tactics with widely used ones in practice, we performed a multiple-case study. According to Verner et al. [44], a multiple-case study is an empirical research method in which several individual cases are studied in depth, using a common framework, to enable cross-case comparisons and to identify patterns, similarities, and differences among the cases. This method allowed us to gain a deep understanding of how companies handle quality objectives, implement tactics, and navigate trade-offs in practice by focusing on several cases of interest. We selected our cases by focusing on a particular business segment of companies that provide ML-based solutions for other companies, in short, ML developing companies, as this would allow us to benefit from particular rich insights over different application domains. The selected companies illustrate the wide-ranging application of ML across various sectors, highlighting the importance of both development and maintenance for quality-driven solutions. Each company brings specific expertise and capabilities, enabling them to represent diverse needs and priorities.

### 1.2.5 Vision

The final contribution of this research is associated with our vision of the framework embedding trade-off management into the MLOps workflow. While the vision is not considered a formal empirical method, it remains a powerful tool for understanding and interpreting phenomena. According to Kantabutra [45], vision is a modelling approach defined by the author that guides their choices and actions, serving as a framework to direct decision-making and behavior. Unlike other methods used in this research, which led to conclusions based exclusively on qualitative and quantitative data, the vision is especially effective for the design conceptual insights based on general experience obtained during the whole research.

## 1.3 Results and Evaluation

While the results of this thesis cover different aspects of machine learning engineering (MLE), particularly software quality, software architecture, and software engineering operations, together they form the basis for a framework, that can be used by ML developing companies, especially by startups and SMEs. Our contributions provide insights for a basic understanding of existing practices in the design of ML-enabled systems and, most importantly, support practitioners with the mapping of potential quality trade-offs and highlight the importance of complex and broad evaluation of such trade-offs during the decision-making process. Together, these contributions culminated in the vision of *MLTradeOps*, which integrates systematic trade-off management into MLOps workflows. Below, we report our main results, structuring them by the aspect they cover.

### 1.3.1 Software Quality of ML-enabled Systems

To build the quality model, we examined 37 scientific sources to obtain a comprehensive list of quality attributes that characterize various ML-enabled systems, including their subtypes, such as deep learning systems. A full list of all quality attributes extracted from the literature and the number of their occurrences can be found in *Paper II*. The quality model was built based on the scientific papers that employed different methodologies, such as interviews, questionnaires, expert assessments, and others.

The quality model is presented in *Paper II* and depicts a two-level diagram built based on the results of taxonomic analysis, where the higher level represents more general quality attributes and the lower level represents more specific quality characteristics that detail the general ones.

Our quality model comprises the following 11 high-level common quality attributes:

- *Functional suitability* is the degree to which a system corresponds to functional requirements. It covers issues associated with functional suitability itself and system correctness.
- *Resource efficiency* is the degree to which a system fulfills a given functionality within an existing amount of hardware capacities. It covers issues associated with time efficiency, energy efficiency, and memory efficiency.
- *System accuracy* is the degree to which a system performs and analyzes the contextual environment and refers to the performance of the entire system in real-world conditions, including model inference and data Postprocessing. Particularly, system accuracy is different from model accuracy, which specifically measures the predictive performance of the trained machine learning model on a given dataset.
- *Usability* is the degree to which a system can be employed by end users to achieve specified goals.
- *Reliability* is the degree to which a system performs specified functions under specified conditions in a fixed domain. It covers issues associated with scalability, reliability itself, safety, robustness, and trustworthiness.
- *Security* is the degree to which a system protects information and data. It covers the issues associated with security itself and privacy.
- *Maintainability* is the degree to which a system can be modified and supported by developers and maintainers to achieve specified goals. It covers issues associated with maintainability itself, testability, and transparency.
- *Portability* is the degree of effectiveness with which a system can be transferred from one domain, software, or hardware basis to another. It covers issues associated with portability itself and adaptability.
- *Explainability* is the degree to which the behavior of a system (primarily, the behavior of ML models) and its output can be explained by humans. It covers issues associated with explainability itself and interpretability.

- *Fairness* is the degree to which a system can detect and prevent an algorithmic bias created by a model. It covers issues associated with fairness itself and with ethical considerations.
- *Data quality* is the degree of integrity and sufficiency of data for model training, testing, and validation, including the reliability of the related data sources. It covers issues associated with data quality and data quantity.

The developed quality model was presented to six experts at the Swedish Requirements Engineering Meeting (SiREN 2023) in Gothenburg, Sweden, organized by Chalmers University of Technology. Four experts out of six participants rated the model as fully complete, general, and relevant. One participant pointed out that the proposed model lacks the “compatibility” attribute (with reference to ISO 25010:2011 [46]). According to the methodology used, this attribute was rarely reported in the literature (only 2 times). This fact did not allow us to include it in the final model. The last expert noted that the selection of terms for quality characteristics is not as important as specific metrics and ways to achieve them; however, they fully supported the proposed model in terms of completeness, generalizability, and relevance.

Further, the model was presented to four ML engineers from Swedish AI software companies. They conducted a theoretical comparison of the quality attributes we found with the product-level qualities considered by them when designing real ML solutions. Three of the practitioners concluded that the resulting model fully exhausts the quality of all the systems developed in their company and is relevant nowadays. The last expert noted the importance of compliance of the developed solutions with all the quality attributes we identified, as well as with the business goals of stakeholders. According to our findings, meeting business requirements can be expressed in both functional and non-functional requirements. Each NFR can be ranked according to the identified quality attributes, while strict adherence to the functional requirements corresponds to the identified “functional suitability” attribute. After clarifying the context and terminology, the expert agreed that the proposed model was complete, general, and relevant.

ML engineers from four companies were further asked to select the most significant quality attributes in their workflow, not necessarily from the proposed quality model. While interviewees did not name any other quality attributes than the ones presented in the proposed quality model, the selection of those varied significantly due to the specifics of each company. These insights were further presented in *Paper III*, where these selected quality attributes were defined as quality priorities and grouped by the respective companies.

The resulting quality model, as well as insights regarding the importance of specific quality attributes, further formed the basis for the mapping of quality trade-offs.

### 1.3.2 Software Architecture of ML-enabled Systems

**Architectural tactics.** Through a systematic literature review (SLR) of 73 papers, we were able to identify 16 architectural tactics aimed at achieving attributes from the quality model as presented in *Paper II*. Through a multiple-case study with 4 ML developing companies, we were able to identify 24 architectural tactics aimed at achieving attributes from the quality model: where 10 architectural tactics fully overlapped with the ones previously identified by SLR and 14 were unique and novel entities to our study. The list of tactics obtained through a multiple case study is presented in *Paper III*. As a result, we collected a list of 30 architectural tactics grouped by the quality attributes they aimed at.

All of the quality attributes were addressed by at least one architectural tactic, except for functional suitability, since this attribute is general in meaning (all ML-enabled systems must follow functional requirements), but not general in the ways of its achievement (for each system there is a unique set of functional requirements), and also because we were unable to find common architectural tactics to satisfy this attribute with the selected methodology.

Below, we present brief descriptions of 16 architectural tactics collected through SLR.

1. *Distributed Learning* is an architectural approach to machine learning aimed at parallelizing computing powers among several computers.
2. *Automated Data Reduction* is an automated process aimed at minimizing the complexity and size of datasets while preserving their essential information (widespread in IoT)
3. *Federated Learning* is an architectural approach to machine learning aimed at training on local heterogeneous datasets.
4. *Human-in-the-Loop (HitL)* is an architectural approach where a human (expert) is integrated into the ML-enabled system as a separate component aimed at monitoring and improving the system's behavior.
5. *Automated Data Versioning* is an automated process aimed at the creation, tracking, and management of different versions or iterations of datasets used for model training, testing, and validation.
6. *Intrusion Detection* is a tactic for complex systems (primarily, IoT systems) aimed at the detection and classification of network intrusions and anomalies.
7. *Automated Data Encryption* is an automated process aimed at securing sensitive data used for training, inference, or model deployment to protect it from unauthorized access.
8. *Containerization* is an architectural approach aimed at packaging an entire system or model (incl. its dependencies and runtime environment), into a standardized unit called a container.

9. *Componentization* is an architectural approach aimed at breaking down a software system into modular components or building blocks that can be independently developed, tested, and deployed.
10. *Local Interpretable Models (LIME)* is an approach to machine learning aimed at explaining black boxes by approximating the behavior of a complex model around a specific prediction using simpler (more interpretable) models.
11. *Rule-based Models* are a type of models that rely on explicit rules (i.e. if-then) that are designed and specified by humans or domain knowledge to approximate complex model behavior.
12. *Automated Hyperparameter Tuning* (or Hyperparameter Optimization) is a method aimed at searching for the best hyperparameter values for the model based on certain criteria.
13. *Automated Algorithm Selection* (or Algorithm Configuration) is an automated process aimed at searching for the most appropriate method(s) for a certain task or in certain circumstances.
14. *Automated Bias Mitigation* is an automated process aimed at identifying and reducing bias in algorithms, models, and datasets by their evaluation through fairness metrics or “sensitive” feature monitoring.
15. *Automated Data Preprocessing* is an automated process aimed at preparing raw data for analysis and model training.
16. *Automated Data Profiling* is an automated process aimed at analyzing and summarizing the characteristics of a dataset to gain insights into its structure, quality, and distribution.

We were conducting constant peer-reviewing of the resulting list of ATs. During weekly meetings, a list of collected architectural tactics was discussed against identified quality attributes based on the expertise of each research participant and co-author of the respective study (*Paper II*). During the validation, issues related to the architectural nature of the identified artifacts and the advisability of classifying them as tactics were discussed. In other words, based on the studied literature, we checked if the artifacts affected the overall principle of architectural design (e.g., containerization) or could be integrated as constituent parts into the overall system architecture (e.g., automated bias mitigation module).

Further, the final list of ATs was shared with four ML engineers from Swedish AI software companies for further evaluation. The details of this evaluation can be found in *Paper II*.

Finally, through a systematic literature review of 96 papers, we built a complex mapping of quality trade-offs caused by each architectural tactic. To achieve this, we identified the impacts of each tactic on each attribute from the quality model, other than the ones they were initially aimed at. We reported the impact of a tactic on a quality attribute if at least one source indicated that

applying the tactic influenced metrics or other indicators for that attribute. When all sources agreed on the impact's direction, either predominantly positive or negative, we reported it as such. If sources reported both predominantly positive and negative impacts for the same tactic-quality attribute combination, depending on conditions of the environment or domain, we marked the impact as ambivalent. This work led to the creation of the complicated trade-off matrix and emphasized the complex and time-consuming nature of trade-off management, underscoring the need to update the existing MLOps paradigm with dedicated processes.

While all the architectural tactics identified through SLR were validated by practitioners, we aimed to extend this list with additional tactics currently used in industry through the multiple-case study, since the list of tactics according to the interviewed practitioners was not complete.

The tactics described by the interviewees in the multiple-case study covered 8 quality attributes, namely, resource efficiency, maintainability, reliability, data quality, fairness, explainability, system accuracy, and security.

Below, we present brief descriptions of 14 novel architectural tactics collected by the multiple-case study.

1. *Cloud-based Components* is an architectural approach that employs cloud infrastructure to host or manage components within an ML-enabled system.
2. *Model Pruning* is an automated process of reducing the size and complexity of the models.
3. *Data Caching* is an automated process of temporarily storing frequently accessed data.
4. *Microservices* is an architectural approach to system decomposition into independently deployable services connected via APIs.
5. *Experiment Tracking* is an automated process of monitoring and managing ML experiment results.
6. *Dependency Tracking* is an automated process of monitoring and managing data dependencies within an ML-enabled system.
7. *Model Verification Module* is an automated process for validating and testing ML models.
8. *Code Versioning* is an automated process for tracking changes in the source code.
9. *Pipeline Management* is an approach for automating the machine learning workflow by enabling data to be transformed and correlated into a model that can then be analyzed to achieve outputs.
10. *User Feedback Module* is an architectural approach that systematically integrates user feedback into the system.

11. *Data Source Evaluation* is an automated process for assessing the reliability and quality of the sources from which the datasets are taken.
12. *Feature Engineering* is an approach for creating, selecting, and transforming input features.
13. *SHAP Explanations* is an approach for explaining the output of ML models by calculating the contribution of each feature to a specific prediction.
14. *Data Postprocessing* is an automated process applied to model outputs to refine or format the data for downstream use.

The multiple-case study re-identified 10 architectural tactics previously found via the SLR. These architectural tactics were: *containerization*, *data versioning*, *human-in-the-loop (HitL)*, *rule-based models*, *rule-based models*, *automated bias mitigation*, *local interpretable models (LIME)*, *data preprocessing*, *hyperparameter tuning*, *data encryption*. The fact that these tactics re-emerged in the multiple-case study reflects their high practical relevance and cross-domain applicability in real-world systems.

When the list of tactics was collected, the interviewees were asked to describe the most common quality trade-offs they encountered in their practice. In contrast with findings obtained by SLR, in the multiple-case study, interviewees did not provide explicit connections between tactic implementations and the trade-offs they cause. Such mapping was quite complicated to build, given the available time frame for the interviews and the chosen methodology of semi-structured interviews. Instead, the interviewees focused on the product-level trade-offs that have significant importance for the delivered products or services. According to interviewees, such trade-offs were usually caused by certain decisions or a combination of decisions, not necessarily architectural, which are made at different stages of development. This insight contributed to our understanding of the nature of trade-offs and later affected our vision of how to balance such trade-offs in practice.

**Design Patterns.** Through a multi-vocal literature review of 80 primary sources, including repository documentation and patent reports, we collected 49 component models of existing ML-enabled solutions. Using content analysis, we further identified 14 design patterns that occur in at least two independent component models. Then, we investigated related literature to build the definitions of the collected patterns and to identify the main challenge they address. We present the definitions of collected design patterns below, while the details regarding the background, problem to be solved, and example of implementation for each design pattern can be found in *Paper IV*.

1. *Ensemble Learning* is an approach to machine learning in which multiple models run in parallel and combine into one prediction.
2. *Parallel Independent Models* is an approach to machine learning in which multiple independent models run on the same input data to find different characteristics using different specialized models to get different outputs.

3. *Sequentially Dependent Models* is an approach to machine learning in which multiple models run in sequence, each with a distinct responsibility of solving a smaller step in the process of solving a greater problem.
4. *Situation-Based Model Selection* is an approach to machine learning in which the system has multiple models to choose from and picks one of them to use based on certain criteria.
5. *Data Transformation* is an approach to data management in which data is modified through a series of procedures that produce incremental results to improve the data to the desired input type for a model.
6. *Feature Extraction* is an approach to data management in which the input data for the model first goes through a component that extracts a set of features instead of using the raw data as input for the model.
7. *Postprocessing* is an approach to data management in which the output data of the model is fed into a component that processes it into the desired format.
8. *Prediction Cache* is an approach to data management in which the input and/or prediction is saved in the storage so that it can later be queried.
9. *Multi-layer pattern* is an approach to system design in which a system is divided into different layers, which have clearly defined purposes, where each layer communicates only with its closest neighbors.
10. *Orchestrator* is an approach to system design in which there is a central component that handles and initiates communication between the machine learning-related components, including but not limited to data handling, training, evaluation, and prediction requests.
11. *Server-Side ML Model* is an approach to system design in which one component has the role of server and allows components or sub-systems to take the role of client, initiating a connection with the server to obtain a prediction from its model.
12. *Expert Validation* is an approach to system validation in which a human domain expert verifies a critical process to ensure it fulfills certain criteria.
13. *Runtime Evaluation and Improvement* is an approach to system validation in which there is a system that monitors and evaluates the deployed ML model, and makes improvements to it either by retraining, tweaking the model, or creating a new model with some changes.
14. *System Evaluation and Improvement* is an approach to system validation in which an ML model evaluates the system and makes iterative changes to the system.

We were conducting constant peer-reviewing of the resulting list of design patterns. During weekly meetings, a list of collected design patterns was

discussed against the correspondence with the selected definition of design pattern based on the expertise of each research participant and co-author of the respective study (*Paper IV*). During the validation, issues related to the recurring nature of the identified artifacts and the advisability of classifying them as patterns were discussed.

Further, the final list of design patterns was shared with ten experts from academia. All the experts verified the list of design patterns and provided valuable insights on their impact on all the attributes from the component model. The interview guide and data extraction sheets can be found in the supplementary artifact to *Paper IV*.

We note that four identified design patterns, namely, *postprocessing*, *feature extraction*, *prediction cache*, and *runtime evaluation and improvement*, largely correspond to architectural tactics previously identified via the multiple-case study, namely, *data postprocessing*, *feature engineering*, *data caching*, and *model verification module*, respectively. This happened largely due to the independence of the methodologies chosen when conducting the corresponding studies. This fact is not a contradiction, since according to the selected definitions, one artifact may be an architectural tactic (if it is explicitly aimed at improving a certain quality attribute) and a design pattern (if it is a solution to the recurring problem within different systems) at the same time. Moreover, this intersection confirms the relevance and wide use in practice of architectural tactics identified previously. Another remarkable observation is that **all** the connections of architectural tactics to the target quality attributes identified in the multiple-case study were never refuted by the identification of impacts of corresponding design patterns on the same quality attributes: e.g., AT of *feature engineering* was primarily aimed at enhancing system accuracy and fairness, while the design pattern of *feature extraction* was described as profitable for improving system accuracy and fairness as well; AT of *data caching* was aimed at improving resource efficiency, while the design pattern of *prediction cache* had positive impact on resource efficiency as well.

As expected, the validation by experts in terms of quality impacts led to the identification of several quality trade-offs for each design pattern evaluated. Together with previous trade-off mapping for architectural tactics, this once again highlighted the need for embedding dedicated trade-off management procedures into the existing MLOps paradigm due to their pervasive and complex nature. Moreover, the identification of trade-offs for patterns on each level of abstraction (data management, model management, system design, and operational validation) proved our hypothesis that trade-offs appear on every dimension of MLOps (data, model, development, operations) and affect the decision-making process at every phase. We took the idea that trade-offs should be balanced after each phase of the production cycle and proposed a new vision for trade-off management in MLOps.

### 1.3.3 Vision for the MLOps framework

Our previous contributions led to the insight that the existing MLOps paradigm is not fully adapted to the needs of multi-objective development, pointing to

the need for special investigation of architectural as well as operational methods for the trade-off management. When we conducted the multiple-case study, we also identified certain non-architectural (operational) tactics applied to achieve quality attributes. It made our perspective broader. We received the evidence that not only design decisions may affect the quality of the delivered products, but also organizational decisions (e.g., knowledge transferring among team members). Such decisions, on the one hand, could potentially lead to new trade-offs, but on the other hand, can become a way of balancing other ones. This fact motivated us to integrate trade-off management into every dimension of the MLOps cycle.

In *Paper V*, we analyzed the evolution of MLOps. We identified that this evolution is aligned with the specialization of MLOps on certain quality attributes, such as security (SecMLOps [47]) or reliability (SafeMLOps [48]). However, due to their focus on one exclusive quality characteristic, such frameworks have limited applicability for production aimed at achieving multiple quality objectives at once (e.g., high reliability with the least resources consumed). Explicitly managing the trade-offs between different, potentially competing, quality objectives can help organizations by enhancing the flexibility and predictability of the MLOps workflow.

Based on this analysis, together with conclusions from previous subsections, we presented a vision around the novel notion of MLTradeOps, focused on explicitly managing trade-offs during the MLOps workflow. It brings together the expertise of existing and emerging MLOps branches focused on specific quality attributes, and also ongoing monitoring and addressing of other general quality characteristics of in-production software systems. We envisioned a framework entitled “MLTradeOps” that makes trade-off management a core part of the decision-making process and contains a high-level cycle to make conscious trade-offs for the ML-enabled system, which are then reflected in lower-level decisions during the MLOps lifecycle. Figure 1.2 illustrates the overall concept of “MLTradeOps”.

In Table 1.1, we present the main steps from the roadmap proposed for the development of this framework and explain what previous results contribute to which step.

While the roadmap and the vision itself are quite ambitious, certain contributions were already made by this licentiate thesis. For instance, we explored the product-level quality attributes and potential product-level quality trade-offs; we conducted an initial exploration of balancing strategies and formulated the rationale for the creation of the envisioned framework. All the results together formed a solid basis for the further development of the proposed framework.

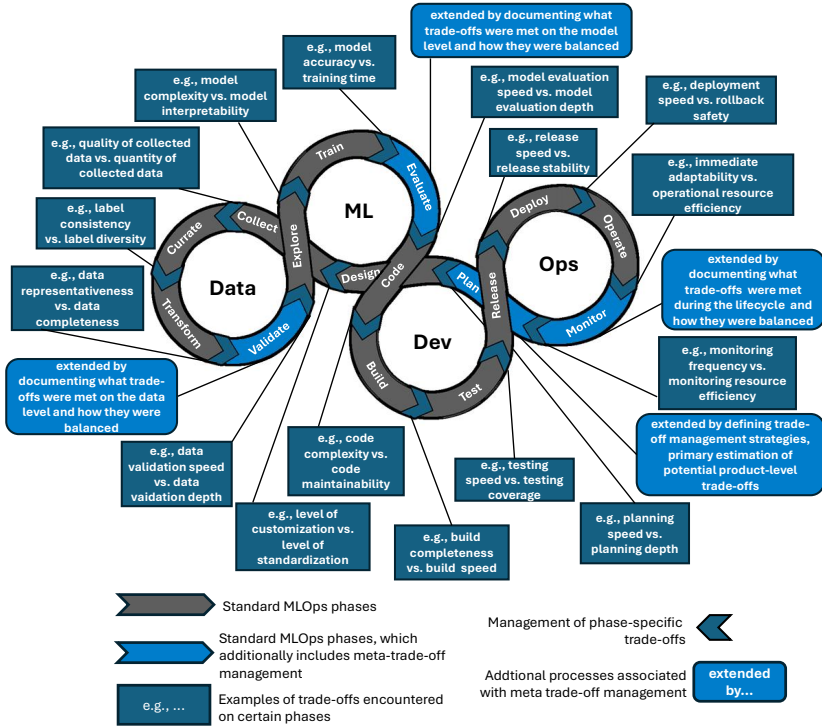


Figure 1.2: Overview of MLTradeOps

Table 1.1: Contributions to the development steps of the MLTradeOps framework

Step of development	Contributions
<b>Step 1:</b> Planning the embedding of the framework into the existing MLOps paradigm.	[Explicitly] <i>Paper V</i> contains the primary analysis and the roadmap. [Implicitly] <i>Papers I, II, III, and IV</i> contain conclusions that highlight the need for the framework.
<b>Step 2:</b> Collecting the most common product-level quality attributes of ML-enabled solutions	<i>Paper II</i> contains a quality model of ML-enabled systems.
<b>Step 3:</b> Collecting the most common trade-offs encountered on the product level.	<i>Papers II, III, and IV</i> contain mappings of trade-offs from different sources.
<b>Step 4:</b> Collecting the most common meta-level trade-offs	<i>Paper V</i> listed some; however, the full investigation is yet to come.
<b>Step 5:</b> Collecting the most common meta trade-off balancing strategies.	<i>Paper V</i> listed some, and <i>Paper III</i> provided some experience on them in the form of non-architectural tactics; however, the full investigation is yet to come.
<b>Step 6:</b> Collecting the most common quality attributes specific to all standard MLOps phases	<i>The full investigation is yet to come.</i>
<b>Step 7:</b> Collecting the most common trade-offs emerging between phase-specific attributes.	<i>The full investigation is yet to come.</i>
<b>Step 8:</b> Collecting the most common techniques to balance phase-specific trade-offs.	<i>The full investigation is yet to come.</i>
<b>Step 9:</b> Assembling and formalizing knowledge into a single coherent framework.	<i>The full investigation is yet to come.</i>

## 1.4 Answers to RQs

In this Section, we present the answers to all RQs of this research.

**RQ1:** *What quality attributes are the most common for ML-enabled systems?*

According to the results of the systematic literature review supplemented by expert validation and taxonomy analysis, we derived **11 main quality attributes** common to ML-enabled systems: *functional suitability, resource efficiency, usability, reliability, security, maintainability, portability, system accuracy, explainability, fairness, and data quality*. The first 7 ones are relevant for ML-enabled systems as well as for traditional software, while the last 4 are exclusively specific to ML-enabled systems. The multiple-case study confirmed the relevance of these quality attributes and demonstrated that the most significant ones for the industry are *reliability, functional suitability, and resource efficiency*.

**RQ2:** *How do widespread architectural tactics affect quality attributes of ML-enabled systems?*

By employing different methodologies, we were able to identify **30 architectural tactics** aimed at achieving all the quality attributes delivered by the answer to RQ1, except for *functional suitability*, since this attribute is general in meaning (all ML-enabled systems must follow functional requirements), but not general in the ways of its achievement (for each system there is a unique set of functional requirements).

According to the results of the systematic literature review supplemented by expert validation, we identified 16 architectural tactics: *Distributed Learning, Automated Data Reduction, Federated Learning, Human-in-the-Loop (HitL), Automated Data Versioning, Intrusion Detection, Automated Data Encryption, Containerization, Componentization, Local Interpretable Models (LIME), Rule-based Models, Automated Hyperparameter Tuning, Automated Algorithm Selection, Automated Bias Mitigation, Automated Data Preprocessing, Automated Data Profiling*. The mapping of the predominantly positive, predominantly negative, and ambivalent impacts of the identified architectural tactics on quality attributes is presented in a trade-off matrix in *Paper II*.

The multiple-case study further identified 24 architectural tactics, 10 of which overlapped with the previously identified ones by SLR, once again confirming their relevance, and 14 of which were novel and unique entities, that were namely: *Cloud-based Components, Model Pruning, Data Caching, Microservices, Experiment Tracking, Dependency Tracking, Model Verification Module, Code Versioning, Pipeline Management, User Feedback Module, Data Source Evaluation, Feature Engineering, SHAP Explanations, Data Postprocessing*. The full list of tactics grouped by quality attributes they address, together with descriptions of frequently encountered trade-offs in practice, can be found in *Paper III*.

**RQ3:** *How do widespread design patterns affect the quality attributes of ML-enabled systems?*

Based on the component models collected via multivocal literature review, through a content analysis supplemented by expert validation, we identified **14**

**design patterns:** *Ensemble Learning, Parallel Independent Models, Sequentially Dependent Models, Situation-Based Model Selection, Data Transformation, Feature Extraction, Postprocessing, Prediction Cache, Multi-layer Pattern, Orchestrator, Server-Side ML Model, Expert Validation, Runtime Evaluation and Improvement, System Evaluation and Improvement.* The mapping of the predominantly positive, predominantly negative, and ambivalent impacts of the identified design patterns on quality attributes is presented in a trade-off matrix in *Paper IV*.

**RQ4:** *How can systematic trade-off management be embedded into the existing MLOps paradigm?*

According to the envisioned framework, trade-off management must be embedded into the MLOps paradigm **after each phase of MLOps** (e.g., data collection, model training, etc.) **on each MLOps dimension** (data, model, development, operations) to balance phase-specific trade-offs (e.g., model complexity vs. model interpretability). Moreover, meta-level trade-off management must be conducted on specific phases (data validation, model evaluation, operational monitoring, and system planning) to make trade-off handling a first-class concern in MLOps, ensuring that phase-specific and product-level trade-offs are addressed in a way that reflects overall goals, priorities, and constraints.

# Chapter 2

## Discussion

Our study contributes to the evolution and systematization of architectural and operational approaches in the area of machine learning engineering (MLE) and, particularly, machine learning operations (MLOps). Our findings can be applicable to theory building as well as to certain industrial processes. Below, we present possible implications for researchers and practitioners. Further, we supplement this Discussion with the analysis of threats to validity.

### 2.1 Implications for researchers

Our research proposes a broad space for conducting associated studies to extend or deepen our findings. In the area of quality for ML-enabled systems, there is a potential to investigate product-level quality attributes relevant to certain types of ML-enabled systems (e.g., quality attributes of large language model (LLM)-based systems) and further compare them with common quality attributes proposed by this thesis. Such an investigation could provide useful insights into the generalizability of our findings and the specifics that certain types of systems may impose in terms of quality.

Another direction of associated research may be the investigation of architectural tactics associated with certain types of ML-enabled systems, which are not relevant for the majority of ML-enabled solutions, but have significant value for a certain type of such systems (e.g., “persona agents” for LLM-based systems [49]). By applying the methodologies proposed by this study, the processes of data collection and data synthesis may be guided in a way that enables researchers to obtain unique, system-specific insights that would otherwise remain unexplored.

Finally, associated studies related to the overall concept of *MLTradeOps* are of special value for the evolution of this framework. The investigation of different phase-specific trade-offs, balancing techniques, and strategies requires significant effort and multi-dimensional expertise, and the involvement of other researchers in this topic may significantly increase the productivity of the research process and the results achieved.

## 2.2 Implications for practitioners

For practitioners, our findings provide several actionable insights that can inform the design, evaluation, and continuous improvement of ML-enabled systems.

First of all, it is a systematic consideration of quality attributes. The proposed quality model can serve as a practical checklist for practitioners, especially in startups and SMEs, to ensure that system design decisions explicitly address both traditional software qualities (e.g., maintainability, reliability) and ML-specific ones (e.g., fairness, explainability, data quality). This helps practitioners structure requirements engineering processes beyond functionality and increases the awareness of less tangible but critical aspects, such as reliability and usability.

Second, our study provides decision support. The catalog of 30 architectural tactics, 8 non-architectural tactics, and 14 design patterns proposes a structured list of solutions that can be applied to achieve specific quality attributes. The mapping of trade-offs provides insights into the positive and negative side effects that such solutions may have on the rest of the quality attributes, making the decision-making process more informed.

Finally, our study propagates trade-off awareness in MLOps workflows. Our vision of MLTradeOps emphasizes the integration of trade-off management into every phase of the ML lifecycle. For practitioners, this means shifting from ad-hoc decision-making to a systematic evaluation of how design and operational choices affect competing quality attributes. Integrating trade-off analysis into existing CI/CD and monitoring pipelines can help organizations achieve more predictable and balanced outcomes in production, as well as significantly enhance the processes of multi-objective development.

## 2.3 Threats to Validity

In this subsection, we discuss threats to internal, external, and construct validity. This subsection covers the only main threats to validity associated with this thesis as a whole. Threats to the validity of specific studies are discussed in the corresponding papers.

### 2.3.1 External Validity

The main threats to external validity concern the generalizability of the thesis findings beyond the studied context.

First, the findings may not be relevant for all types of ML-enabled systems. The systems considered in the thesis, both in the literature and in the industrial studies, represent only ML-enabled systems in general. Different types of ML-enabled systems (e.g., embedded systems, large-scale cloud-based platforms, or safety-critical applications) may exhibit architectural concerns, constraints, and trade-offs that are not captured in the current set of findings. While we took action to mitigate this threat and included certain types of ML-enabled systems in our search strategies, such as deep learning systems and neural networks,

some other types may remain unconsidered. As a result, the generalizability of the proposed patterns, tactics, and trade-offs may be limited for such types of systems, and replication across a broader spectrum of systems is required to strengthen their applicability.

Another threat to external validity arises from the fact that the field of ML and AI evolves rapidly, with new tools and practices emerging continuously. Consequently, the literature-based findings presented in this thesis may become incomplete or outdated as the state of practice progresses. Architectural tactics or patterns not yet documented in the reviewed literature or observed in the studied companies may gain relevance in the near future. This dynamic nature of the field poses a risk to the long-term validity of the results, which calls for periodic updates through new literature reviews, empirical studies, and practitioner feedback to ensure that the insights remain relevant.

Finally, company-specific factors affect the generalizability of the industry-based studies. The selected companies may reflect the priorities and constraints of particular organizational contexts. Furthermore, the fact that all companies in the industrial part of the study were headquartered in Sweden may bias the findings toward regional practices and values. To mitigate this limitation, broader replications with companies across different countries, sizes, and domains are needed.

### 2.3.2 Internal Validity

. Internal validity threats concern whether the conclusions drawn in this thesis accurately reflect cause-and-effect relationships.

Across the studies, several sources of subjectivity and bias may have influenced the results. For instance, in the multivocal literature review, our interpretations of component models, design patterns, and quality attributes may have shaped the resulting list of design patterns and trade-offs. Similarly, in the multiple-case study, interview data reflect the subjective perspectives of selected participants, which may not fully capture all viewpoints within a company.

### 2.3.3 Construct Validity

Construct validity concerns the extent to which the measurements used in a study are appropriate and accurately represent what is being investigated

One threat is associated with commonly reported quality attributes in the literature. This approach implicitly assumes that frequent reporting correlates with generalizability and justifies its inclusion in a common quality model for ML-enabled software. However, less frequently reported attributes may still play a significant role in particular domains or use cases. One way to mitigate this threat would be to conduct a separate study dedicated to such attributes.

Another threat is associated with the evaluation of the impacts of design patterns and architectural tactics on quality attributes. While we relied on both the literature and expert interviews to identify these impacts, the assessment still carries potential biases. The literature reported the impacts of tactics on

quality attributes; however, such impacts in practice might be more dependent on the context than we suggested. While all interviewees had extensive academic or industrial experience in the domain and judgments were shared between the experts and the authors, it may still introduce subjectivity in measuring proposed impacts.

## 2.4 Conclusion and Future Work

This thesis proposes a new approach to the design, development, and operation of ML-enabled systems by introducing systematic trade-off management into MLOps workflows. This approach is based on empirically collected findings in the fields of software quality, software architecture, and software engineering operations.

The envisioned framework builds a solid foundation for the implementation of this approach and implies several directions of future work, which are detailed in the proposed roadmap. These directions of future work include investigating quality attributes specific to certain MLOps lifecycle phases and potential trade-offs between these attributes, exploring low-level techniques to balance these trade-offs, developing an empirically grounded theory of trade-off management, and identifying high-level strategies associated with it.

# Bibliography

- [1] Y. Jiang, M. Yang, S. Wang, X. Li and Y. Sun, ‘Emerging role of deep learning-based artificial intelligence in tumor pathology,’ *Cancer Commun.*, 2020 (cit. on p. 3).
- [2] J. K. Afriyie, K. Tawiah, W. A. Pels *et al.*, ‘A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions,’ *Decis. Anal. J.*, 2023 (cit. on p. 3).
- [3] M. R. Kumar, J Venkatesh and A. M. Z. Rahman, ‘Data mining and machine learning in retail business: Developing efficiencies for better customer retention,’ *AIHC*, 2021 (cit. on p. 3).
- [4] A. E. Brownlee, J. Adair, S. O. Haraldsson and J. Jabbo, ‘Exploring the accuracy-energy trade-off in machine learning,’ in *GI*, IEEE, 2021 (cit. on pp. 3, 8, 6 (III)).
- [5] D. Kreuzberger, N. Kühn and S. Hirschl, ‘Machine learning operations (MLOps): Overview, definition, and architecture,’ *IEEE Access*, 2023 (cit. on pp. 4, 8, 2 (V), 2 (V), 8 (V)).
- [6] M. G. Hanna, L. Pantanowitz, B. Jackson *et al.*, ‘Ethical and bias considerations in artificial intelligence/machine learning,’ *Mod. Pathol.*, 2025 (cit. on p. 4).
- [7] R. G. Cooper, ‘Why AI projects fail: Lessons from new product development,’ *Eng. Man. Rev.*, 2024 (cit. on p. 4).
- [8] A. Burkov, *Machine learning engineering*. True Positive Inc., 2020 (cit. on p. 5).
- [9] C. Kästner, *Machine learning in production: From models to systems*. MIT Press, 2022 (cit. on pp. 6, 2 (I)).
- [10] G. T. Heineman and W. T. Councill, *Component-based software engineering*. Springer, 2001 (cit. on p. 6).
- [11] L. Bass, P. Clements and R. Kazman, *Software architecture in practice*. Addison-Wesley, 2021 (cit. on pp. 6, 8).
- [12] G. A. Lewis, I. Ozkaya and X. Xu, ‘Software architecture challenges for ML systems,’ in *ICSME*, IEEE, 2021 (cit. on pp. 6, 20 (II)).
- [13] H. Washizaki, H. Uchida, F. Khomh and Y.-G. Guéhéneuc, ‘Machine learning architecture and design patterns,’ *Software*, 2020 (cit. on p. 6).

- [14] A. Serban and J. Visser, ‘Adapting software architectures to machine learning challenges,’ in *SANER*, 2022, pp. 152–163 (cit. on pp. 6, 7, 4 (II), 2 (III), 5 (III)).
- [15] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995 (cit. on p. 6).
- [16] J. I. Olszewska, *Guide to the Software Engineering Body of Knowledge (SWEBoK) v4.0*. CS, 2024, vol. 4.0 (cit. on pp. 6, 6 (I)).
- [17] B. B. Mayvan, A. Rasoolzadegan and Z. G. Yazdi, ‘The state of the art on design patterns: A systematic mapping of the literature,’ *JSS*, 2017 (cit. on p. 6).
- [18] Q. Lu, L. Zhu, X. Xu, J. Whittle, D. Zowghi and A. Jacquet, ‘Responsible AI pattern catalogue: A collection of best practices for AI governance and engineering,’ *Comput. Surv.*, 2024 (cit. on p. 6).
- [19] J. Juziuk, D. Weyns and T. Holvoet, ‘Design patterns for multi-agent systems: A systematic literature review,’ in *AOSE*, Springer, 2014 (cit. on p. 6).
- [20] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E. B. Fernandez and N. Yoshioka, ‘Landscape of architecture and design patterns for IoT systems,’ *IoT*, 2020 (cit. on p. 6).
- [21] D. Taibi, V. Lenarduzzi and C. Pahl, ‘Architectural patterns for microservices: A systematic mapping study,’ in *CLOSER*, SciTePress, 2018 (cit. on p. 6).
- [22] H. Washizaki, H. Uchida, F. Khomh and Y.-G. Guéhéneuc, ‘Studying software engineering patterns for designing machine learning systems,’ in *IWESEP*, IEEE, 2019 (cit. on pp. 7, 16 (II)).
- [23] H. Washizaki, F. Khomh, Y.-G. Guéhéneuc *et al.*, ‘Software-engineering design patterns for machine learning applications,’ *Computer*, 2022 (cit. on p. 7).
- [24] L. Heiland, M. Hauser and J. Bogner, ‘Design patterns for AI-based systems: A multivocal literature review and pattern repository,’ *arXiv preprint*, 2023 (cit. on pp. 7, 3 (I)).
- [25] I. Crnkovic, S. Sentilles, A. Vulgarakis and M. R. Chaudron, ‘A classification framework for software component models,’ *TSE*, 2010 (cit. on pp. 7, 6 (IV)).
- [26] P. Santhanam, ‘Quality management of machine learning systems,’ in *EDSMLS*, 2020, pp. 1–13 (cit. on pp. 7, 4 (II), 2 (III), 5 (III)).
- [27] J. Siebert, L. Joeckel, J. Heidrich *et al.*, ‘Construction of a quality model for machine learning systems,’ *SQ*, 2022 (cit. on pp. 7, 5 (II), 5 (III), 15 (V)).
- [28] H. Kuwajima, H. Yasuoka and T. Nakae, ‘Engineering problems in machine learning systems,’ *ML*, 2020 (cit. on pp. 7, 5 (II), 15 (II), 5 (III)).

- [29] B. Gezici and A. K. Tarhan, 'Systematic literature review on software quality for AI-based software,' *ESE*, 2022 (cit. on pp. 7, 5 (II)).
- [30] X. Franch, S. Martínez-Fernández, C. P. Ayala and C. Gómez, 'Architectural decisions in ai-based systems: An ontological view,' in *QUATIC*, 2022 (cit. on pp. 8, 5 (II), 2 (III), 5 (III)).
- [31] M. Bhat, K. Shumaiev, U. Hohenstein, A. Biesdorf and F. Matthes, 'The evolution of architectural decision making as a key focus area of software architecture research: A semi-systematic literature study,' in *ICSA*, 2020 (cit. on pp. 8, 5 (II), 2 (III), 5 (III)).
- [32] H. Muccini and K. Vaidhyanathan, 'Software architecture for ML-based systems: What exists and what lies ahead,' in *WAIN*, IEEE, 2021 (cit. on pp. 8, 5 (II), 2 (III), 5 (III)).
- [33] P. Berander, L.-O. Damm, J. Eriksson *et al.*, 'Software quality attributes and trade-offs,' *Blekinge Institute of Technology*, 2005 (cit. on pp. 8, 11 (V)).
- [34] G. Baryannis, S. Dani and G. Antoniou, 'Predicting supply chain risks using machine learning: The trade-off between performance and interpretability,' *Future Gener. Comput. Syst*, 2019 (cit. on pp. 8, 6 (III)).
- [35] M. van Haastrecht, M. Brinkhuis and M. Spruit, 'Federated learning analytics: Investigating the privacy-performance trade-off in machine learning for educational analytics,' in *AIED*, 2024 (cit. on pp. 8, 6 (III)).
- [36] J. W. Creswell, 'Mixed-method research: Introduction and application,' in *Handbook of educational policy*, Elsevier, 1999 (cit. on p. 9).
- [37] B. Kitchenham and S. Charters, 'Guidelines for performing systematic literature reviews in software engineering,' *EBSE Technical Report*, 2007 (cit. on pp. 9, 11, 6 (IV), 6 (IV), 11 (IV), 14 (IV)).
- [38] V. Garousi, M. Felderer and M. V. Mäntylä, 'Guidelines for including grey literature and conducting multivocal literature reviews in software engineering,' *Inf. Softw. Technol*, 2019 (cit. on p. 11).
- [39] V. Garousi, M. Felderer and M. V. Mäntylä, 'The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature,' in *EASE*, 2016 (cit. on p. 11).
- [40] H. Cooper, L. V. Hedges and J. C. Valentine, *The handbook of research synthesis and meta-analysis*. Russell Sage Foundation, 2019 (cit. on p. 11).
- [41] J. Schöpfel, 'Towards a prague definition of grey literature,' *TGJ*, 2011 (cit. on p. 11).
- [42] R. J. Adams, P. Smart and A. S. Huff, 'Shades of grey: Guidelines for working with the grey literature in systematic reviews for management and organizational studies,' *Int. J. Manag. Rev*, 2017 (cit. on p. 11).
- [43] S. E. Hove and B. Anda, 'Experiences from conducting semi-structured interviews in empirical software engineering research,' in *METRICS*, IEEE, 2005 (cit. on p. 11).

- [44] J. M. Verner, J. Sampson, V. Tasic, N. A. Bakar and B. A. Kitchenham, ‘Guidelines for industrially-based multiple case studies in software engineering,’ in *RCIS*, IEEE, 2009 (cit. on p. 12).
- [45] S. Kantabutra, ‘What do we know about vision,’ in *Leading Organizations: Perspectives for a New Era*, Russell Sage Foundation, 2010 (cit. on p. 12).
- [46] I. E. Commission and I. O. for Standardization, ‘Iso/iec 9126: Software engineering—product quality,’ ISO, Technical Report, 2001 (cit. on p. 14).
- [47] X. Zhang and J. Jaskolka, ‘Conceptualizing the secure machine learning operations (SecMLOps) paradigm,’ in *QRS*, IEEE, 2022 (cit. on p. 21).
- [48] A. Nouri *et al.*, ‘The DevSafeOps dilemma: A systematic literature review on rapidity in safe autonomous driving development and operation,’ *arXiv preprint*, 2025 (cit. on p. 21).
- [49] G. Sun, X. Zhan and J. Such, ‘Building better AI agents: A provocation on the utilisation of persona in LLM-based conversational agents,’ in *SIGCHI CUI*, ACM, 2024 (cit. on p. 25).
- [50] J. Diaz-de Arcaya, A. I. Torre-Bastida, G. Zárate, R. Miñón and A. Almeida, ‘A joint study of the challenges, opportunities, and roadmap of mlops and aiops: A systematic survey,’ *CS*, 2023 (cit. on p. 2 (I)).
- [51] C. Szyperski, D. Gruntz and S. Murer, *Component software: beyond object-oriented programming*. Pearson Edu, 2002 (cit. on p. 2 (I)).
- [52] S. Alla, S. K. Adari, S. Alla and S. K. Adari, ‘Beginning of MLOps,’ *What is MLOps?*, 2021 (cit. on pp. 2 (I), 36 (II)).
- [53] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch and H. H. Olsson, ‘Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions,’ *IST*, 2020 (cit. on pp. 2 (I), 16 (II)).
- [54] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson and I. Crnkovic, ‘A taxonomy of software engineering challenges for machine learning systems: An empirical investigation,’ in *XP*, Springer, 2019 (cit. on p. 2 (I)).
- [55] L. E. Lwakatare, I. Crnkovic, E. Rånge and J. Bosch, ‘From a data science-driven process to a continuous delivery process for machine learning systems,’ in *PROFES*, Springer, 2020 (cit. on p. 2 (I)).
- [56] S. Amershi, A. Begel, C. Bird *et al.*, ‘Software engineering for machine learning: A case study,’ in *ICSE-SEIP*, IEEE, 2019 (cit. on p. 3 (I)).
- [57] D. Sculley, G. Holt, D. Golovin *et al.*, ‘Hidden technical debt in machine learning systems,’ *NeurIPS*, 2015 (cit. on p. 3 (I)).
- [58] M. S. Rahman, E. Rivera, F. Khomh, Y.-G. Guéhéneuc and B. Lehnert, ‘Machine learning software engineering in practice: An industrial case study,’ *arXiv*, 2019 (cit. on p. 3 (I)).

- [59] G. Giray, ‘A software engineering perspective on engineering machine learning systems: State of the art and challenges,’ *JSS*, 2021 (cit. on p. 3 (I)).
- [60] H. Washizaki, H. Uchida, F. Khomh and Y.-G. Guéhéneuc, ‘Studying software engineering patterns for designing machine learning systems,’ in *IWESEP*, IEEE, 2019 (cit. on p. 3 (I)).
- [61] Z. Wan, X. Xia, D. Lo and G. C. Murphy, ‘How does machine learning change software development practices?’ *TSE*, 2019 (cit. on pp. 3 (I), 15 (II)).
- [62] R. Nazir, A. Bucaioni and P. Pelliccione, ‘Architecting ML-enabled systems: Challenges, best practices, and design decisions,’ *JSS*, 2024 (cit. on p. 3 (I)).
- [63] A. Serban, K. van der Blom, H. Hoos and J. Visser, ‘Software engineering practices for machine learning—adoption, effects, and team assessment,’ *JSS*, 2024 (cit. on p. 3 (I)).
- [64] S. J. Warnett and U. Zdun, ‘Architectural design decisions for machine learning deployment,’ in *ICSA*, IEEE, 2022 (cit. on pp. 3 (I), 21 (II), 26 (II)).
- [65] H.-M. Heyn, E. Knauss and P. Pelliccione, ‘A compositional approach to creating architecture frameworks with an application to distributed AI systems,’ *JSS*, 2023 (cit. on pp. 3 (I), 16 (II)).
- [66] S. Wagner, ‘Software product quality control,’ 2013 (cit. on p. 4 (I)).
- [67] V. Indykov, D. Strüber and R. Wohlrab, ‘Architectural tactics to achieve common quality attributes of machine-learning-enabled systems,’ in *JSS*, Elsevier, 2025 (cit. on pp. 4 (I), 4 (I), 6 (III)).
- [68] S. Angelov, P. Grefen and D. Greefhorst, ‘A classification of software reference architectures: Analyzing their success and effectiveness,’ in *ECSCA*, IEEE, 2009 (cit. on p. 5 (I)).
- [69] G. Hulten, *Building Intelligent Systems: A Guide to Machine Learning Engineering*. Apress, 2018 (cit. on p. 2 (II)).
- [70] R. Monson-Haefel, *97 things every software architect should know: collective wisdom from the experts*. O’Reilly Media, Inc., 2009 (cit. on p. 2 (II)).
- [71] I. O. for Standardization, ‘ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation — System and software quality models,’ ISO, Tech. Rep., 2011 (cit. on pp. 2 (II), 3 (II), 7 (II), 10 (II), 18 (II), 34 (II)).
- [72] I. O. for Standardization, ‘ISO/IEC 42010:2011 - Systems and software engineering — Architecture description,’ ISO, Tech. Rep., 2011 (cit. on p. 2 (II)).

- [73] I. O. for Standardization, ‘ISO/IEC 25059:2023 Software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality model for AI systems,’ ISO, Tech. Rep., 2023 (cit. on pp. 2 (II), 3 (II), 10 (II), 34 (II), 5 (III), 15 (V)).
- [74] X. Wang and M. Miao, ‘A framework for requirements specification of machine-learning systems,’ in *SEKE*, 2022 (cit. on p. 3 (II)).
- [75] M. U. Hassan, M. H. Rehmani, R. Kotagiri, J. Zhang and J. Chen, ‘Differential privacy for renewable energy resources based smart metering,’ *JPDC*, 2019 (cit. on p. 3 (II)).
- [76] L. Wanganoo and V. K. Shukla, ‘Real-time data monitoring in cold supply chain through NB-IoT,’ in *ICCCNT*, 2020 (cit. on p. 3 (II)).
- [77] L. Bass, P. Clements and R. Kazman, *Software architecture in practice*. Addison-Wesley, 2003 (cit. on pp. 4 (II), 4 (II), 4 (II), 4 (III), 4 (III)).
- [78] I. O. for Standardization, ‘ISO/IEC 9126:2001 Software engineering - Product quality,’ ISO, Tech. Rep., 2001 (cit. on p. 4 (II)).
- [79] L. Lundberg, J. Bosch, D. Häggander and P.-O. Bengtsson, ‘Quality attributes in software architecture design,’ in *IASTED*, 1999 (cit. on p. 4 (II)).
- [80] B. Kitchenham and S. Charters, ‘Guidelines for performing systematic literature reviews in software engineering,’ EBSE Technical Report, Tech. Rep., 2007 (cit. on pp. 6 (II), 12 (II)).
- [81] K. M. Habibullah, G. Gay and J. Horkoff, ‘Non-functional requirements for machine learning: Understanding current use and challenges among practitioners,’ *RE*, 2023 (cit. on pp. 8 (II), 14 (II), 15 (II)).
- [82] S. Vojří and T. Kliegr, ‘Editable machine learning models? a rule-based framework for user studies of explainability,’ *ADAC*, 2020 (cit. on p. 8 (II)).
- [83] J. W. Drisko and T. Maschi, *Content analysis*. Oxford Publ., 2016 (cit. on p. 9 (II)).
- [84] S. K. Reed, ‘A taxonomic analysis of abstraction,’ *Perspectives on Psychological Science*, 2016 (cit. on p. 9 (II)).
- [85] S. Peldszus, H. Knopp, Y. Sens and T. Berger, ‘Towards ML-integration and training patterns for AI-enabled systems,’ in *International Conference on Bridging the Gap between AI and Reality*, Springer, 2023 (cit. on pp. 11 (II), 36 (II), 19 (IV)).
- [86] A. Vogelsang and M. Borg, ‘Requirements engineering for machine learning: Perspectives from data scientists,’ in *REW*, 2019 (cit. on p. 14 (II)).
- [87] C. Kästner and E. Kang, ‘Teaching software engineering for AI-enabled systems,’ in *ICSE*, 2020 (cit. on p. 15 (II)).
- [88] M. A. Ağca, S. Faye and D. Khadraoui, ‘A survey on trusted distributed artificial intelligence,’ *ECSA*, 2022 (cit. on p. 15 (II)).

- [89] H. Liu, S. Eksmo, J. Risberg and R. Hebig, ‘Emerging and changing tasks in the development process for machine learning systems,’ in *ICSSP*, 2020 (cit. on p. 15 (II)).
- [90] P. Haindl, T. Hoch, J. Dominguez, J. Aperribai, N. K. Ure and M. Tunçel, ‘Quality characteristics of a software platform for human- AI teaming in smart manufacturing,’ in *QUATIC*, 2022 (cit. on p. 15 (II)).
- [91] F. Ishikawa and N. Yoshioka, ‘How do engineers perceive difficulties in engineering of machine-learning systems?-questionnaire survey,’ in *CESI*, 2019 (cit. on p. 15 (II)).
- [92] A. Serban, K. van der Blom, H. Hoos and J. Visser, ‘Adoption and effects of software engineering best practices in machine learning,’ in *IEEE*, 2020 (cit. on p. 15 (II)).
- [93] R. H. Yap, ‘Towards certifying trustworthy machine learning systems,’ in *TAILOR*, 2021 (cit. on p. 15 (II)).
- [94] I. Ozkaya, ‘What is really different in engineering AI-enabled systems?’ *IEEE Softw.*, 2020 (cit. on p. 15 (II)).
- [95] J. M. Zhang, M. Harman, L. Ma and Y. Liu, ‘Machine learning testing: Survey, landscapes and horizons,’ *TSE*, 2020 (cit. on p. 15 (II)).
- [96] H.-L. Truong, ‘Coordination-aware assurance for end-to-end machine learning systems: The R3E approach,’ in *AI Assurance*, 2023 (cit. on p. 15 (II)).
- [97] H. Kuwajima and F. Ishikawa, ‘Adapting SQuaRE for quality assessment of artificial intelligence systems,’ in *ISSREW*, 2019 (cit. on p. 15 (II)).
- [98] J. Horkoff, ‘Non-functional requirements for machine learning: Challenges and new directions,’ in *RE*, 2019 (cit. on p. 15 (II)).
- [99] Q. Chen, Y. Gong, Y. Lu and J. Tang, ‘Classifying and measuring the service quality of AI chatbot in frontline service,’ *Journal of Business Research*, 2022, ISSN: 0148-2963 (cit. on p. 15 (II)).
- [100] A. Poth, B. Meyer, P. Schlicht and A. Riel, ‘Quality assurance for machine learning—an approach to function and system safeguarding,’ in *QRS*, 2020 (cit. on p. 15 (II)).
- [101] H. Challa, N. Niu and R. Johnson, ‘Faulty requirements made valuable: On the role of data quality in deep learning,’ in *AIRE*, 2020 (cit. on p. 16 (II)).
- [102] A. Chakraborty, R. Bagavathi and U. Tomer, ‘A comprehensive decomposition towards the facets of quality in IoT,’ in *ICOSEC*, 2020 (cit. on p. 16 (II)).
- [103] A. Perera, A. Aleti, C. Tantithamthavorn *et al.*, ‘Search-based fairness testing for regression-based machine learning systems,’ *ESE*, 2022 (cit. on p. 16 (II)).
- [104] K. Nakamichi, K. Ohashi, I. Namba *et al.*, ‘Requirements-driven method to determine quality characteristics and measurements for machine learning software and its evaluation,’ in *RE*, 2020 (cit. on p. 16 (II)).

- [105] A. L. Smith and R. Clifford, ‘Quality characteristics of artificially intelligent systems,’ in *IWESQ APSEC*, 2020 (cit. on p. 16 (II)).
- [106] H. Yokoyama, ‘Machine learning system architectural pattern for improving operational stability,’ in *ICSA*, 2019 (cit. on p. 16 (II)).
- [107] H. Barzamini, M. Shahzad, H. Alhoori and M. Rahimi, ‘A multi-level semantic web for hard-to-specify domain concept, pedestrian, in ML-based software,’ *RE*, 2022 (cit. on p. 16 (II)).
- [108] S. Khan, S. Tsutsumi, T. Yairi and S. Nakasuka, ‘Robustness of AI-based prognostic and systems health management,’ *Annu. Rev. Control.*, 2021 (cit. on p. 16 (II)).
- [109] N. Balasubramaniam, M. Kauppinen, K. Hiekkanen and S. Kujala, ‘Transparency and explainability of AI systems: Ethical guidelines in practice,’ in *REFSQ*, 2022 (cit. on p. 16 (II)).
- [110] L. M. Cysneiros and J. C. S. do Prado Leite, ‘Non-functional requirements orienting the development of socially responsible software,’ in *CAiSE*, 2020 (cit. on p. 16 (II)).
- [111] K. Ahmad, M. Abdelrazek, C. Arora, M. Bano and J. Grundy, ‘Requirements practices and gaps when engineering human-centered artificial intelligence systems,’ *ASOC*, 2023 (cit. on p. 16 (II)).
- [112] M. Felderer and R. Ramler, ‘Quality assurance for AI-based systems: Overview and challenges (introduction to interactive session),’ in *SWQD*, 2021 (cit. on p. 16 (II)).
- [113] M. Felderer, B. Russo and F. Auer, ‘On testing data-intensive software systems,’ *C-CPS*, 2019 (cit. on p. 16 (II)).
- [114] D. G. Arseniev, D. E. Baskakov, J. Kasurinen, V. P. Shkodyrev and A. Mergasov, ‘Software engineering principles apply to artificial intelligence systems,’ in *CPS&C*, 2021 (cit. on p. 16 (II)).
- [115] M. M. Morovati, A. Nikanjam, F. Khomh and Z. M. Jiang, ‘Bugs in machine learning-based systems: A faultload benchmark,’ *ESE*, 2023 (cit. on p. 16 (II)).
- [116] C. Tao, J. Gao and T. Wang, ‘Testing and quality validation for AI software—perspectives, issues, and practices,’ *JSS*, 2019 (cit. on p. 16 (II)).
- [117] J. Dodge, Q. V. Liao, Y. Zhang, R. K. Bellamy and C. Dugan, ‘Explaining models: An empirical study of how explanations impact fairness judgment,’ in *IUI*, 2019 (cit. on p. 16 (II)).
- [118] G. Amaral, R. Guizzardi, G. Guizzardi and J. Mylopoulos, ‘Ontology-based modeling and analysis of trustworthiness requirements: Preliminary results,’ in *ER*, 2020 (cit. on p. 16 (II)).
- [119] S. Picard, C. Chapdelaine, C. Cappi *et al.*, ‘Ensuring dataset quality for machine learning certification,’ in *ISSREW*, 2020 (cit. on p. 16 (II)).
- [120] S. V. Garbuk, ‘Intellimetry as a way to ensure AI trustworthiness,’ in *IC-AIAI*, 2018 (cit. on p. 16 (II)).

- [121] F. Boenisch, V. Battis, N. Buchmann and M. Poikela, ““i never thought about securing my machine learning systems”: A study of security and privacy awareness of machine learning practitioners,’ in *MuC*, 2021 (cit. on p. 16 (II)).
- [122] S. Shi, Q. Wang, X. Chu *et al.*, ‘Communication-efficient distributed deep learning with merged gradient sparsification on gpus,’ in *INFOCOM*, IEEE, 2020 (cit. on p. 19 (II)).
- [123] J. Rao, X. Bu, K. Wang and C.-Z. Xu, ‘Self-adaptive provisioning of virtualized resources in cloud computing,’ in *SIGMETRICS*, 2011 (cit. on pp. 19 (II), 19 (II), 26 (II)).
- [124] B. Zhao, M. Song, S. Liu *et al.*, ‘Mosaicnet: A deep-learning-based multi-tile biomedical image stitching method,’ in *EMBC*, IEEE, 2023 (cit. on p. 19 (II)).
- [125] S. Noureen, M. Zubair, M. Ali and M. Q. Mehmood, ‘Deep learning based sequence modeling for optical response retrieval of photonic nanostructures,’ in *IBCAST*, IEEE, 2021 (cit. on p. 19 (II)).
- [126] G. Drainakis, P. Pantazopoulos, K. V. Katsaros, V. Sourlas, A. Amditis and D. I. Kaklamani, ‘From centralized to federated learning: Exploring performance and end-to-end resource consumption,’ *Computer Networks*, 2023 (cit. on pp. 20 (II), 26 (II)).
- [127] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis and W. Shi, ‘Federated learning of predictive models from federated electronic health records,’ *International journal of medical informatics*, 2018 (cit. on p. 20 (II)).
- [128] A. P. Singh and S. Chaudhari, ‘Embedded machine learning-based data reduction in application-specific constrained IoT networks,’ in *SIGAPP*, 2020 (cit. on pp. 20 (II), 20 (II), 26 (II)).
- [129] M. H. ur Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah and S. U. Khan, ‘Big data reduction methods: A survey,’ *Data Science and Engineering*, 2016 (cit. on pp. 20 (II), 20 (II), 20 (II)).
- [130] A. M. Hussein, A. K. Idrees and R. Couturier, ‘Distributed energy-efficient data reduction approach based on prediction and compression to reduce data transmission in IoT networks,’ *International Journal of Communication Systems*, 2022 (cit. on p. 20 (II)).
- [131] D. Petrelli, A.-S. Dadzie and V. Lanfranchi, ‘Mediating between AI and highly specialized users,’ *AI Magazine*, 2012 (cit. on pp. 20 (II), 20 (II), 26 (II)).
- [132] M. Winter, P. Jackson and S. Fallahkhair, ‘Gesture Me: A machine learning tool for designers to train gesture classifiers,’ in *CHIRA*, Springer, 2023 (cit. on p. 20 (II)).
- [133] M. Kröll and K. Burova-Keßler, ‘AI and learning in the context of digital transformation,’ in *AHFE*, Springer, 2021 (cit. on p. 20 (II)).

- [134] F. Heimerl, S. Koch, H. Bosch and T. Ertl, ‘Visual classifier training for text document retrieval,’ *TVCG*, 2012 (cit. on pp. 20 (II), 20 (II)).
- [135] F. Sperrle, M. El-Assady, G. Guo *et al.*, ‘A survey of human-centered evaluations in human-centered machine learning,’ in *Computer Graphics Forum*, Wiley Online Library, 2021 (cit. on p. 20 (II)).
- [136] O. Gómez-Carmona, D. Casado-Mansilla, D. López-de Ipiña and J. García-Zubia, ‘Human-in-the-loop machine learning: Reconceptualizing the role of the user in interactive approaches,’ *Internet of Things*, 2024 (cit. on p. 20 (II)).
- [137] T. Van Der Weide, D. Papadopoulos, O. Smirnov, M. Zielinski and T. Van Kasteren, ‘Versioning for end-to-end machine learning pipelines,’ in *DM-ML*, 2017 (cit. on pp. 21 (II), 21 (II)).
- [138] P. T. Rajendran, H. Espinoza, A. Delaborde and C. Mraidha, ‘Human-in-the-loop learning methods toward safe DL-based autonomous systems: A review,’ in *SAFECOMP*, Springer, 2021 (cit. on pp. 21 (II), 21 (II), 21 (II), 21 (II)).
- [139] P. Sanju, ‘Enhancing intrusion detection in IoT systems: A hybrid metaheuristics-deep learning approach with ensemble of recurrent neural networks,’ *JER*, 2023 (cit. on pp. 21 (II), 21 (II), 21 (II), 26 (II)).
- [140] Q. Liu, L. Chen, H. Jiang *et al.*, ‘A collaborative deep learning microservice for backdoor defenses in industrial IoT networks,’ *Ad Hoc Networks*, 2022 (cit. on p. 21 (II)).
- [141] F. Qu, J. Zhang, Z. Shao and S. Qi, ‘An intrusion detection model based on deep belief network,’ in *ICNCC*, 2017 (cit. on pp. 21 (II), 21 (II)).
- [142] S. Laqtib, K. E. Yassini and M. L. Hasnaoui, ‘A deep learning methods for intrusion detection systems based machine learning in MANET,’ in *SCA*, 2019 (cit. on pp. 21 (II), 21 (II)).
- [143] S. M. M. Rashid, M. Toufikuzzaman and M. S. Hossain, ‘A deep learning based semi-supervised network intrusion detection system robust to adversarial attacks,’ in *NSysS*, 2023 (cit. on pp. 21 (II), 21 (II)).
- [144] H. El Balbali and A. Abou El Kalam, ‘AI-driven big data quality improvement for efficient threat detection in agricultural IoT systems,’ in *AI2SD*, Springer, 2023 (cit. on pp. 21 (II), 30 (II)).
- [145] G. McGraw, ‘Security engineering for machine learning (keynote),’ in *SIGSOFT*, 2020 (cit. on pp. 22 (II), 26 (II)).
- [146] W. Bekri, R. Jmal and L. C. Fourati, ‘Secure and trustworthiness IoT systems: Investigations and literature review,’ *TS*, 2024 (cit. on pp. 22 (II), 22 (II)).
- [147] Y.-H. Wu, X.-H. Huang, J.-X. Liu and L. Chang, ‘A big data encryption method based on lorenz and feistel structures,’ in *ICCEAI*, IEEE, 2022 (cit. on p. 22 (II)).
- [148] M. Kantarcioglu and F. Shaon, ‘Securing big data in the age of AI,’ in *TPS-ISA*, IEEE, 2019 (cit. on p. 22 (II)).

- [149] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, ‘Privacy-preserving deep learning via additively homomorphic encryption,’ *TIFS*, 2017 (cit. on pp. 22 (II), 22 (II)).
- [150] M. Kim, O. Günlü and R. F. Schaefer, ‘Federated learning with local differential privacy: Trade-offs between privacy, utility, and communication,’ in *ICASSP*, IEEE, 2021 (cit. on p. 22 (II)).
- [151] J. Zhou, Z. Su, J. Ni, Y. Wang, Y. Pan and R. Xing, ‘Personalized privacy-preserving federated learning: Optimized trade-off between utility and privacy,’ in *GLOBECOM*, IEEE, 2022 (cit. on pp. 22 (II), 22 (II), 22 (II)).
- [152] H. Kaur, V. Rani, M. Kumar, M. Sachdeva, A. Mittal and K. Kumar, ‘Federated learning: A comprehensive review of recent advances and applications,’ *Multimedia Tools and Applications*, 2023 (cit. on pp. 22 (II), 22 (II)).
- [153] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li and Y. Gao, ‘A survey on federated learning,’ *KBS*, 2021 (cit. on pp. 22 (II), 22 (II)).
- [154] M. M. Rovnyagin, A. S. Hrapov, A. V. Guminskaia and A. P. Orlov, ‘ML-based heterogeneous container orchestration architecture,’ in *EIConRus*, IEEE, 2020 (cit. on pp. 22 (II), 22 (II), 26 (II)).
- [155] A. B. Kolltveit and J. Li, ‘Operationalizing machine learning models: A systematic literature review,’ in *SE4RAI*, 2022 (cit. on pp. 22 (II), 22 (II)).
- [156] M. Openja, F. Majidi, F. Khomh, B. Chembakottu and H. Li, ‘Studying the practices of deploying machine learning projects on docker,’ in *EASE*, 2022 (cit. on pp. 22 (II), 22 (II), 22 (II), 30 (II)).
- [157] J. González-Abad, Á. López García and V. Y. Kozlov, ‘A container-based workflow for distributed training of deep learning algorithms in HPC clusters,’ *Cl. Comp.*, 2023 (cit. on p. 22 (II)).
- [158] H. B. Braiek and F. Khomh, ‘On testing machine learning programs,’ *JSS*, 2020 (cit. on p. 23 (II)).
- [159] S. Singaravel, J. Suykens and P. Geyer, ‘Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction,’ *Adv. Eng. Inform.*, 2018 (cit. on pp. 23 (II), 23 (II)).
- [160] J. Wonsil, J. Sullivan, M. Seltzer and A. Pocock, ‘Integrated reproducibility with self-describing machine learning models,’ in *CRR*, 2023 (cit. on pp. 23 (II), 23 (II), 26 (II)).
- [161] N. Shadab and A. Salado, ‘Towards an interface description template for reusing AI-enabled systems,’ in *SMC*, IEEE, 2020 (cit. on p. 23 (II)).
- [162] P. Geyer and S. Singaravel, ‘Component-based machine learning for performance prediction in building design,’ *Applied energy*, 2018 (cit. on pp. 23 (II), 23 (II), 23 (II)).

- [163] N. Naydenov and S. Ruseva, ‘Combining container orchestration and machine learning in the cloud: A systematic mapping study,’ in *INFOTEH*, IEEE, 2022 (cit. on pp. 23 (II), 23 (II), 30 (II)).
- [164] S. Joshi, B. Hasan and R. Brindha, ‘Optimal declarative orchestration of full lifecycle of machine learning models for cloud native,’ in *ICAAIC*, IEEE, 2024 (cit. on p. 23 (II)).
- [165] V. U. Gongane, M. V. Munot and A. D. Anuse, ‘A survey of explainable AI techniques for detection of fake news and hate speech on social media platforms,’ *JCS*, 2024 (cit. on pp. 23 (II), 26 (II)).
- [166] D. Minh, H. X. Wang, Y. F. Li and T. N. Nguyen, ‘Explainable artificial intelligence: A comprehensive review,’ *Artificial Intelligence Review*, 2022 (cit. on p. 23 (II)).
- [167] J. Maan, ‘Deep learning-driven explainable AI using generative adversarial network (GAN),’ in *INDICON*, IEEE, 2022 (cit. on pp. 24 (II), 25 (II), 25 (II), 25 (II)).
- [168] K. Höhle, M. Kern, T. Hewson and R. Westermann, ‘A comparative study of convolutional neural network models for wind field downscaling,’ *Meteorological Applications*, 2020 (cit. on p. 24 (II)).
- [169] D. Gaspar, P. Silva and C. Silva, ‘Explainable AI for intrusion detection systems: Lime and shap applicability on multi-layer perceptron,’ *ICE/ITMC*, 2024 (cit. on p. 24 (II)).
- [170] H. Saadatfar, Z. Kiani-Zadegan and B. Ghahremani-Nezhad, ‘US-LIME: Increasing fidelity in LIME using uncertainty sampling on tabular data,’ *Neurocomputing*, 2024 (cit. on pp. 24 (II), 24 (II)).
- [171] N. Burkart and M. F. Huber, ‘A survey on the explainability of supervised machine learning,’ *JAIR*, 2021 (cit. on p. 24 (II)).
- [172] P. E. Love, W. Fang, J. Matthews, S. Porter, H. Luo and L. Ding, ‘Explainable artificial intelligence (XAI): Precepts, models, and opportunities for research in construction,’ *AEI*, 2023 (cit. on pp. 24 (II), 26 (II)).
- [173] R. Moraffah, M. Karami, R. Guo, A. Raglin and H. Liu, ‘Causal interpretability for machine learning-problems, methods and evaluation,’ *SIGKDD*, 2020 (cit. on p. 24 (II)).
- [174] C. P. Vieira and L. A. Digiampietri, ‘Machine learning post-hoc interpretability: A systematic mapping study,’ in *SBSI*, ACM, 2022 (cit. on p. 24 (II)).
- [175] E. Soares, P. P. Angelov, B. Costa, M. P. G. Castro, S. Nagesh Rao and D. Filev, ‘Explaining deep learning models through rule-based approximation and visualization,’ *TFS*, 2020 (cit. on p. 24 (II)).
- [176] D. Rajapaksha, C. Bergmeir and W. Buntine, ‘LoRMiKA: Local rule-based model interpretability with k-optimal associations,’ *Information Sciences*, 2020 (cit. on pp. 24 (II), 24 (II)).

- [177] Y. Rimal, N. Sharma and A. Alsadoon, 'The accuracy of machine learning models relies on hyperparameter tuning: Student result classification using random forest, randomized search, grid search, bayesian, genetic, and optuna algorithms,' *Multimedia Tools and Applications*, 2024 (cit. on p. 24 (II)).
- [178] M. Daviran, A. Maghsoudi, R. Ghezelbash and B. Pradhan, 'A new strategy for spatial predictive mapping of mineral prospectivity: Automated hyperparameter tuning of random forest approach,' *Computers & Geosciences*, 2021 (cit. on p. 24 (II)).
- [179] A. L. C. Ottoni, A. M. Souza and M. S. Novo, 'Automated hyperparameter tuning for crack lee2021landscapeimage classification with deep learning,' *Soft Computing*, 2023 (cit. on pp. 24 (II), 24 (II), 26 (II)).
- [180] P. Kerschke, H. H. Hoos, F. Neumann and H. Trautmann, 'Automated algorithm selection: Survey and perspectives,' *Evolutionary computation*, 2019 (cit. on pp. 24 (II), 25 (II), 26 (II)).
- [181] N. Pise and P. Kulkarni, 'Algorithm selection for classification problems,' in *SAI*, IEEE, 2016 (cit. on pp. 24 (II), 25 (II)).
- [182] H. H. Rashidi, N. Tran, S. Albahra and L. T. Dang, 'Machine learning in health care and laboratory medicine: General overview of supervised learning and Auto-ML,' *International Journal of Laboratory Hematology*, 2021 (cit. on p. 25 (II)).
- [183] F. Deeba and S. R. Patil, 'Utilization of machine learning algorithms for prediction of diseases,' in *i-PACT*, IEEE, 2021 (cit. on p. 25 (II)).
- [184] M. Alissa, K. Sim and E. Hart, 'A feature-free approach to automated algorithm selection,' in *GECCO*, Wiley, 2023 (cit. on pp. 25 (II), 25 (II), 25 (II)).
- [185] M. S. A. Lee and J. Singh, 'The landscape and gaps in open source fairness toolkits,' in *CHI*, 2021 (cit. on p. 25 (II)).
- [186] C. Ferrara, G. Sellitto, F. Ferrucci, F. Palomba and A. De Lucia, 'Fairness-aware machine learning engineering: How far are we?' *ESE*, 2024 (cit. on pp. 25 (II), 27 (II)).
- [187] A. Agarwal and H. Agarwal, 'A seven-layer model with checklists for standardising fairness assessment throughout the AI lifecycle,' *AI and Ethics*, 2023 (cit. on p. 25 (II)).
- [188] A. Castelnovo, R. Crupi, G. Greco, D. Regoli, I. G. Penco and A. C. Cosentini, 'A clarification of the nuances in the fairness metrics landscape,' *Scientific Reports*, 2022 (cit. on p. 25 (II)).
- [189] J. Zhang, Y. Shu and H. Yu, 'Fairness in design: A framework for facilitating ethical artificial intelligence designs,' *IJCS*, 2023 (cit. on p. 25 (II)).
- [190] S. Siddiqi, R. Kern and M. Boehm, 'Saga: A scalable framework for optimizing data cleaning pipelines for machine learning applications,' *MoD*, 2023 (cit. on pp. 25 (II), 27 (II)).

- [191] G. Canbek, S. Sagioglu and T. T. Temizel, ‘New techniques in profiling big datasets for machine learning with a concise review of android mobile malware datasets,’ in *IBIGDELFT*, 2018 (cit. on p. 25 (II)).
- [192] F. Mostafa, L. Tao and W. Yu, ‘An effective architecture of digital twin system to support human decision making and AI-driven autonomy,’ *CCPE*, 2021 (cit. on p. 25 (II)).
- [193] I. Logothetis, S. Barnett, L. Hoon *et al.*, ‘Pims: A pre-ML labelling tool,’ in *e-Science*, IEEE, 2022 (cit. on p. 25 (II)).
- [194] R. R. Pansara, B. Y. Kasula, A. B. Bhatia and P. Whig, ‘Enhancing sustainable development through machine learning-driven master data management,’ in *International Conference on Sustainable Development through Machine Learning, AI and IoT*, Springer, 2024 (cit. on p. 25 (II)).
- [195] R. Gawhade, L. R. Bohara, J. Mathew and P. Bari, ‘Computerized data-preprocessing to improve data quality,’ in *ICPC2T*, 2022 (cit. on pp. 25 (II), 27 (II)).
- [196] M Ramkumar, K Malathi and K Pavithra, ‘Optimizing machine learning model accuracy via OBNT algorithm: Advanced data preprocessing technique,’ in *ICESSES*, IEEE, 2023 (cit. on pp. 25 (II), 26 (II)).
- [197] L. Santos and L. Ferreira, ‘Atlantic—automated data preprocessing framework for supervised machine learning,’ *Software Impacts*, 2023 (cit. on pp. 25 (II), 25 (II)).
- [198] M. Bilal, G. Ali, M. W. Iqbal, M. Anwar, M. S. A. Malik and R. A. Kadir, ‘Auto-prep: Efficient and automated data preprocessing pipeline,’ *IEEE Access*, 2022 (cit. on p. 26 (II)).
- [199] O. Nassef, W. Sun, H. Purmehdi, M. Tatipamula and T. Mahmoodi, ‘A survey: Distributed machine learning for 5G and beyond,’ *Computer Networks*, 2022 (cit. on pp. 28 (II), 28 (II)).
- [200] H.-P. Cheng, P. Yu, H. Hu *et al.*, ‘Towards decentralized deep learning with differential privacy,’ in *ICCC*, Springer, 2019 (cit. on p. 28 (II)).
- [201] F. Zerka, V. Urovi, F. Bottari *et al.*, ‘Privacy preserving distributed learning classifiers—sequential learning with small sets of data,’ *Computers in Biology and Medicine*, 2021 (cit. on p. 28 (II)).
- [202] B. Guijarro-Berdiñas, S. Fernandez-Lorenzo, N. Sánchez-Maróño and O. Fontenla-Romero, ‘A privacy-preserving distributed and incremental learning method for intrusion detection,’ in *ICANN*, Springer, 2011 (cit. on p. 28 (II)).
- [203] N. Mandela, I. Alam, A Amudha, D Priyanka, D. K. Singh *et al.*, ‘Enabling scalable applications with intelligent distributed data processing,’ in *INCOFT*, IEEE, 2023 (cit. on p. 28 (II)).
- [204] A. Tuladhar, D. Rajashekar and N. D. Forkert, ‘Distributed learning in healthcare,’ *Trends of Artificial Intelligence and Big Data for E-Health*, 2023 (cit. on p. 28 (II)).

- [205] D. Fan, Y. Wu and X. Li, ‘On the fairness of swarm learning in skin lesion classification,’ in *MICCAI*, Springer, 2021 (cit. on p. 28 (II)).
- [206] T. Lane and C. E. Brodley, ‘Temporal sequence learning and data reduction for anomaly detection,’ *TISSEC*, 2019 (cit. on p. 28 (II)).
- [207] M. Tomei, A. Schwing, S. Narayanasamy and R. Kumar, ‘Sensor training data reduction for autonomous vehicles,’ in *MOBICOM*, 2019 (cit. on p. 29 (II)).
- [208] M. Z. A. Bhuiyan, T. Wang, A. Zaman and G. Wang, ‘Data reduction through decision-making based on event-sensitivity in IoT-enabled event monitoring,’ in *HPCC*, IEEE, 2019 (cit. on p. 29 (II)).
- [209] S. Shen, T. Zhu, D. Wu, W. Wang and W. Zhou, ‘From distributed machine learning to federated learning: In the view of data privacy and security,’ *CCPE*, 2022 (cit. on p. 29 (II)).
- [210] H. Jeong and T.-M. Chung, ‘Security and privacy issues and solutions in federated learning for digital healthcare,’ in *FDSE*, Springer, 2022 (cit. on p. 29 (II)).
- [211] G. K. Jagarlamudi, A. Yazdinejad, R. M. Parizi and S. Pouriyeh, ‘Exploring privacy measurement in federated learning,’ *The Journal of Supercomputing*, 2023 (cit. on p. 29 (II)).
- [212] S. Shin, M. Boyapati, K. Suo, K. Kang and J. Son, ‘An empirical analysis of image augmentation against model inversion attack in federated learning,’ *Cluster Computing*, 2023 (cit. on p. 29 (II)).
- [213] L. Lyu, H. Yu, X. Ma *et al.*, ‘Privacy and robustness in federated learning: Attacks and defenses,’ *IEEE transactions on neural networks and learning systems*, 2022 (cit. on pp. 29 (II), 29 (II)).
- [214] M. Kirienko, M. Sollini, G. Ninatti *et al.*, ‘Distributed learning: A reliable privacy-preserving strategy to change multicenter collaborations using AI,’ *EJNMMI*, 2021 (cit. on p. 29 (II)).
- [215] F. Sattler, K.-R. Müller, T. Wiegand and W. Samek, ‘On the byzantine robustness of clustered federated learning,’ in *ICASSP*, IEEE, 2020 (cit. on p. 29 (II)).
- [216] H. Lycklama, L. Burkhalter, A. Viand, N. Kuchler and A. Hithnawi, ‘RoFL: Robustness of secure federated learning,’ in *SP*, IEEE, 2023 (cit. on p. 29 (II)).
- [217] X. Gu, Z. Tianqing, J. Li, T. Zhang, W. Ren and K.-K. R. Choo, ‘Privacy, accuracy, and model fairness trade-offs in federated learning,’ *Computers & Security*, 2022 (cit. on p. 29 (II)).
- [218] T. N. Nguyen and R. Choo, ‘Human-in-the-loop XAI-enabled vulnerability detection, investigation, and mitigation,’ in *ASE*, IEEE, 2021 (cit. on p. 29 (II)).
- [219] M. L. Jones, E. Kaufman and E. Edenberg, ‘AI and the ethics of automating consent,’ *SP*, 2018 (cit. on p. 29 (II)).

- [220] S. Jena, S. Sundarrajan, A. Meena and B. Chandavarkar, ‘Human-in-the-loop control and security for intelligent cyber-physical systems (CPSs) and IoT,’ in *ICDSIAI*, Springer, 2022 (cit. on p. 29 (II)).
- [221] D. Xin, L. Ma, J. Liu, S. Macke, S. Song and A. Parameswaran, ‘Accelerating human-in-the-loop machine learning: Challenges and opportunities,’ in *DEEM*, 2018 (cit. on p. 29 (II)).
- [222] Y. Kang, Y.-W. Chiu, M.-Y. Lin, F.-Y. Su and S.-T. Huang, ‘Towards model-informed precision dosing with expert-in-the-loop machine learning,’ in *IRI*, IEEE, 2021 (cit. on p. 29 (II)).
- [223] D. M. Rodríguez, M. P. Cuéllar and D. P. Morales, ‘Concept logic trees: Enabling user interaction for transparent image classification and human-in-the-loop learning,’ *AI*, 2024 (cit. on p. 29 (II)).
- [224] N. Zhang, R. Bahsoon, N. Tziritas and G. Theodoropoulos, ‘Explainable human-in-the-loop dynamic data-driven digital twins,’ in *DDAS*, Springer, 2022 (cit. on p. 29 (II)).
- [225] J. Liu, ‘Human-in-the-loop ethical AI for care robots and confucian virtue ethics,’ in *ICSR*, Springer, 2022 (cit. on p. 29 (II)).
- [226] S. Kalanathan, A. Kichutkin, Z. Shang, A. Strausz, F. J. S. Bautiste and M. El-Assady, ‘Mindset: A bias-detection interface using a visual human-in-the-loop workflow,’ in *ECAI*, Springer, 2023 (cit. on p. 29 (II)).
- [227] B. Ghai and K. Mueller, ‘D-bias: A causality-based human-in-the-loop system for tackling algorithmic bias,’ *TVCG*, 2022 (cit. on p. 29 (II)).
- [228] M. Priestley, F. O’donnell and E. Simperl, ‘A survey of data quality requirements that matter in ML development pipelines,’ *JDIQ*, 2023 (cit. on p. 29 (II)).
- [229] J. Jakubik, M. Vössing, N. Kühn, J. Walk and G. Satzger, ‘Data-centric artificial intelligence,’ *BISE*, 2024 (cit. on p. 29 (II)).
- [230] M. H. N. Yousefi, V. Degeler and A. Lazovik, ‘Empowering machine learning development with service-oriented computing principles,’ in *SummerSOC*, Springer, 2023 (cit. on p. 29 (II)).
- [231] S. Tsimenidis, T. Lagkas and K. Rantos, ‘Deep learning in IoT intrusion detection,’ *Journal of network and systems management*, 2022 (cit. on p. 29 (II)).
- [232] R. Devendiran and A. V. Turukmane, ‘Dugat-LSTM: Deep learning based network intrusion detection system using chaotic optimization strategy,’ *Expert Systems with Applications*, 2024 (cit. on p. 29 (II)).
- [233] B. Lampe and W. Meng, ‘A survey of deep learning-based intrusion detection in automotive applications,’ *Expert Systems with Applications*, 2023 (cit. on p. 29 (II)).
- [234] M. Pawlicki, A. Pawlicka, M. Śrutek, R. Kozik and M. Choraś, ‘Interpreting intrusions-the role of explainability in AI-based intrusion detection systems,’ in *CORES*, Springer, 2023 (cit. on p. 29 (II)).

- [235] C. Shand, R. Fong and U. Butt, ‘How explainable artificial intelligence (XAI) models can be used within intrusion detection systems (IDS) to enhance an analyst’s trust and understanding,’ in *ICGS3*, Springer, 2023 (cit. on p. 30 (II)).
- [236] S. Aljawarneh, M. B. Yassein and W. A. Talafha, ‘A multithreaded programming approach for multimedia big data: Encryption system,’ *MTA*, 2018 (cit. on p. 30 (II)).
- [237] D. Weng, ‘Performance and energy evaluation of lightweight cryptography for small IoT devices,’ in *UEMCON*, IEEE, 2023 (cit. on p. 30 (II)).
- [238] R. Cantoro, N. I. Deligiannis, M. S. Reorda, M. Traiola and E. Valea, ‘Evaluating data encryption effects on the resilience of an artificial neural network,’ in *DFT*, IEEE, 2020 (cit. on p. 30 (II)).
- [239] C.-J. Wang, P.-P. Li, X.-Y. Zhou and N. Liu, ‘Privacy-preserving breast cancer prediction via inner-product functional encryption,’ in *ICCC*, IEEE, 2021 (cit. on p. 30 (II)).
- [240] G. Gupta and K. Lakhwani, ‘An enhanced approach to improve the encryption of big data using intelligent classification technique,’ *Multimedia Tools and Applications*, 2022 (cit. on p. 30 (II)).
- [241] M. M. Rovnyagin, K. V. Timofeev, A. A. Elenkin and V. A. Shipugin, ‘Cloud computing architecture for high-volume ML-based solutions,’ in *EIconRus*, IEEE, 2019 (cit. on pp. 30 (II), 30 (II)).
- [242] C. Figueroa, T. Knowles, V. Kukreja and C.-H. Lung, ‘IoT management with container orchestration,’ in *ICEIB*, IEEE, 2023 (cit. on p. 30 (II)).
- [243] N. Joraviya, B. N. Gohil and U. P. Rao, ‘DL-HIDS: Deep learning-based host intrusion detection system using system calls-to-image for containerized cloud environment,’ *The Journal of Supercomputing*, 2024 (cit. on p. 30 (II)).
- [244] S. Arisdakessian, O. A. Wahab, A. Mourad and H. Otrok, ‘Towards instant clustering approach for federated learning client selection,’ in *ICNC*, IEEE, 2023 (cit. on p. 30 (II)).
- [245] H. Siddiqui, F. Khendek and M. Toeroe, ‘Microservices based architectures for IoT systems-state-of-the-art review,’ *IoT*, 2023 (cit. on p. 30 (II)).
- [246] H. S. Sarjoughian, F. Fallah, S. Saeidi and E. J. Yellig, ‘Transforming discrete event models to machine learning models,’ in *WSC*, IEEE, 2023 (cit. on p. 30 (II)).
- [247] B. Heisele, P. Ho and T. Poggio, ‘Face recognition with support vector machines: Global versus component-based approach,’ in *ICCV*, IEEE, 2011 (cit. on p. 30 (II)).
- [248] N. B. Kumarakulasinghe, T. Blomberg, J. Liu, A. S. Leao and P. Papatrou, ‘Evaluating local interpretable model-agnostic explanations on clinical machine learning classification models,’ in *CBMS*, IEEE, 2020 (cit. on p. 30 (II)).

- [249] T. Mori and N. Uchihiro, ‘Balancing the trade-off between accuracy and interpretability in software defect prediction,’ *ESE*, 2019 (cit. on p. 30 (II)).
- [250] V. Bhargava, M. Couceiro and A. Napoli, ‘LimeOut: An ensemble approach to improve process fairness,’ in *ECML PKDD*, Springer, 2020 (cit. on p. 30 (II)).
- [251] N. Burkart, M. Huber and P. Faller, ‘Forcing interpretability for deep neural networks through rule-based regularization,’ in *ICMLA*, IEEE, 2019 (cit. on p. 31 (II)).
- [252] M. Soui, I. Gasmı, S. Smiti and K. Ghédıra, ‘Rule-based credit risk assessment model using multi-objective evolutionary algorithms,’ *Expert systems with applications*, 2019 (cit. on p. 31 (II)).
- [253] M. I. Rey, M. Galende, M. J. Fuente and G. Sainz-Palmero, ‘Multi-objective based fuzzy rule based systems (FRBSs) for trade-off improvement in accuracy and interpretability: A rule relevance point of view.,’ *KBS*, 2017 (cit. on p. 31 (II)).
- [254] L. Liao, H. Li, W. Shang and L. Ma, ‘An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks,’ *TOSEM*, 2022 (cit. on p. 31 (II)).
- [255] W. Romsaiyud, H. Schnoor and W. Hasselbring, ‘Improving k-nearest neighbor pattern recognition models for privacy-preserving data analysis,’ in *Big Data*, IEEE, 2019 (cit. on p. 31 (II)).
- [256] L. Liu, J. Yu and Z. Ding, ‘Adaptive and efficient GPU time sharing for hyperparameter tuning in cloud,’ in *ICPP*, 2022 (cit. on p. 31 (II)).
- [257] D. K. Jain, A. K. Dutta, E. Verdú, S. Alsubai and A. R. W. Sait, ‘An automated hyperparameter tuned deep learning model enabled facial emotion recognition for autonomous vehicle drivers,’ *Image and Vision Computing*, 2023 (cit. on p. 31 (II)).
- [258] Y. N. Kunang, S. Nurmaini, D. Stiawan and B. Y. Suprpto, ‘Attack classification of an intrusion detection system using deep learning and hyperparameter optimization,’ *JISA*, 2021 (cit. on p. 31 (II)).
- [259] L. Wu, G. Perin and S. Picek, ‘I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis,’ *Trans. Emerg. Topics Comput.*, 2022 (cit. on p. 31 (II)).
- [260] R. K. Batchu and H. Seetha, ‘A generalized machine learning model for DDoS attacks detection using hybrid feature selection and hyperparameter tuning,’ *Computer Networks*, 2021 (cit. on p. 31 (II)).
- [261] S. B. Feroz, N. Sharmin and M. S. Sevas, ‘An empirical analysis of hyperparameter tuning impact on ensemble machine learning algorithm for earthquake damage prediction,’ *Asian Journal of Civil Engineering*, 2024 (cit. on p. 31 (II)).
- [262] V. Perrone, M. Donini, M. B. Zafar, R. Schmucker, K. Kenthapadi and C. Archambeau, ‘Fair bayesian optimization,’ in *AIES*, 2021 (cit. on p. 31 (II)).

- [263] X. Gao, J. Zhai, S. Ma, C. Shen, Y. Chen and Q. Wang, ‘FairNeuron: Improving deep neural network fairness with adversary games on selective neurons,’ in *ICSE, 2022* (cit. on p. 31 (II)).
- [264] I. Dagan, R. Vainshtein, G. Katz and L. Rokach, ‘Automated algorithm selection using meta-learning and pre-trained deep convolution neural networks,’ *Information Fusion, 2024* (cit. on p. 31 (II)).
- [265] J. Bossek, P. Kerschke and H. Trautmann, ‘A multi-objective perspective on performance assessment and automated selection of single-objective optimization algorithms,’ *ASC, 2020* (cit. on p. 31 (II)).
- [266] S. Shahoud, M. Winter, H. Khalloof, C. Duepmeier and V. Hagenmeyer, ‘An extended meta learning approach for automating model selection in big data environments using microservice and container virtualization technologies,’ *IoT, 2021* (cit. on p. 31 (II)).
- [267] R. Trajanov, S. Dimeski, M. Popovski, P. Korošec and T. Eftimov, ‘Explainable landscape analysis in automated algorithm performance prediction,’ in *EvoStar, Springer, 2022* (cit. on p. 31 (II)).
- [268] W. Hutiri, A. Y. Ding, F. Kawsar and A. Mathur, ‘Tiny, always-on, and fragile: Bias propagation through design choices in on-device machine learning workflows,’ *TSEM, 2023* (cit. on pp. 31 (II), 31 (II)).
- [269] M. Hort, Z. Chen, J. M. Zhang, M. Harman and F. Sarro, ‘Bias mitigation for machine learning classifiers: A comprehensive survey,’ *JRC, 2023* (cit. on p. 31 (II)).
- [270] R. Ghani, K. T. Rodolfa, P. Saleiro and S. Jesus, ‘Addressing bias and fairness in machine learning: A practical guide and hands-on tutorial,’ in *SIGKDD, 2023* (cit. on p. 31 (II)).
- [271] S. Jain and P. Kumar, ‘Cost effective generic machine learning operation: A case study,’ in *ICDSNS, IEEE, 2023* (cit. on p. 31 (II)).
- [272] Z. Chen, J. M. Zhang, F. Sarro and M. Harman, ‘A comprehensive empirical study of bias mitigation methods for machine learning classifiers,’ *TSEM, 2023* (cit. on p. 31 (II)).
- [273] M. Miceli, J. Posada and T. Yang, ‘Studying up machine learning data: Why talk about bias when we mean power?’ *Human-Computer Interaction, 2022* (cit. on p. 31 (II)).
- [274] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak and F. Herrera, ‘A survey on data preprocessing for data stream mining: Current status and future directions,’ *Neurocomputing, 2017* (cit. on p. 31 (II)).
- [275] M. C. Rendleman, J. M. Buatti, T. A. Braun *et al.*, ‘Machine learning with the TCGA-HNSC dataset: Improving usability by addressing inconsistency, sparsity, and high-dimensionality,’ *BMC bioinformatics, 2019* (cit. on p. 32 (II)).
- [276] K. Shivashankar and A. Martini, ‘Maintainability challenges in ML: A systematic literature review,’ in *SEAA, IEEE, 2022* (cit. on pp. 32 (II), 36 (II)).

- [277] N. A. Hikal and M. Elgayar, 'Enhancing IoT botnets attack detection using machine learning-IDS and ensemble data preprocessing technique,' in *ITAF*, 2020 (cit. on p. 32 (II)).
- [278] M. A. Bouke and A. Abdullah, 'An empirical study of pattern leakage impact during data preprocessing on machine learning-based intrusion detection models reliability,' *ESA*, 2023 (cit. on p. 32 (II)).
- [279] C. V. G. Zelaya, 'Towards explaining the effects of data preprocessing on machine learning,' in *ICDE*, 2019 (cit. on p. 32 (II)).
- [280] M. M. Basha and P Kuppusamy, 'F-DDPT: An efficient fuzzy-based automated preprocessing technique to support explainability,' in *ICCDN*, Springer, 2022 (cit. on p. 32 (II)).
- [281] Y. Sun, F. Haghghat and B. C. Fung, 'Trade-off between accuracy and fairness of data-driven building and indoor environment models: A comparative study of pre-processing methods,' *Energy*, 2022 (cit. on pp. 32 (II), 32 (II)).
- [282] H. S. Obaid, S. A. Dheyab and S. S. Sabry, 'The impact of data preprocessing techniques and dimensionality reduction on the accuracy of machine learning,' in *IEMECON*, IEEE, 2019 (cit. on p. 32 (II)).
- [283] S. Sajid, B. M. von Zernichow, A. Soylu and D. Roman, 'Predictive data transformation suggestions in grafterizer using machine learning,' in *MTSR*, Springer, 2019 (cit. on p. 32 (II)).
- [284] S. Oppold and M. Herschel, 'A system framework for personalized and transparent data-driven decisions,' in *CAiSE*, Springer, 2020 (cit. on p. 32 (II)).
- [285] N. B. Ding and E. Mit, 'A framework of data quality assurance using machine learning,' in *CITA*, IEEE, 2023 (cit. on p. 32 (II)).
- [286] E. Seo, H. Kim and T.-M. Chung, 'Profiling-based classification algorithms for security applications in Internet of Things,' in *ICIOT*, IEEE, 2019 (cit. on p. 32 (II)).
- [287] W. Epperson, V. Gorantla, D. Moritz and A. Perer, 'Dead or alive: Continuous data profiling for interactive data science,' *Trans. Vis. Comput. Graph*, 2023 (cit. on p. 32 (II)).
- [288] S. Tverdal, A. Goknil, P. Nguyen *et al.*, 'Edge-based data profiling and repair as a service for IoT,' in *IoT*, 2023 (cit. on p. 32 (II)).
- [289] S. Barney, K. Petersen, M. Svahnberg, A. Aurum and H. Barney, 'Software quality trade-offs: A systematic map,' *IST*, 2012 (cit. on pp. 36 (II), 4 (III)).
- [290] V. Mohan and L. B. Othmane, 'Secdevops: Is it a marketing buzzword?-mapping research on security in devops,' in *ARES*, IEEE, 2016 (cit. on p. 36 (II)).
- [291] M. Staron, *Action research in software engineering*. Springer, 2020 (cit. on p. 37 (II)).

- [292] J. Melegati and F. Kon, ‘Early-stage software startups: Main challenges and possible answers,’ in *Fundamentals of Software Startups: Essential Engineering and Business Aspects*, Springer, 2020, pp. 129–143 (cit. on p. 2 (III)).
- [293] N. Khalid, A. Qayyum, M. Bilal, A. Al-Fuqaha and J. Qadir, ‘Privacy-preserving artificial intelligence in healthcare: Techniques and applications,’ *Computers in Biology and Medicine*, 2023 (cit. on p. 3 (III)).
- [294] H. Pan, M. Luo, J. Wang, T. Huang and W. Sun, ‘A safe motion planning and reliable control framework for autonomous vehicles,’ *T-IV*, 2024 (cit. on p. 3 (III)).
- [295] M. Khemka and B. Houck, ‘Toward effective ai support for developers: A survey of desires and concerns,’ *Queue*, 2024 (cit. on p. 3 (III)).
- [296] L. Shan, B. Sangchoolie and P. e. a. Folkesson, ‘A survey on the applicability of safety, security and privacy standards in developing dependable systems,’ in *SAFECOMP*, 2019 (cit. on p. 3 (III)).
- [297] S. Larsson, ‘Ai in the eu: Ethical guidelines as a governance tool,’ *The European Union and the technology shift*, 2021 (cit. on p. 3 (III)).
- [298] S. Kim, D.-K. Kim, L. Lu and S. Park, ‘Quality-driven architecture development using architectural tactics,’ *JSS*, 2009 (cit. on pp. 4 (III), 4 (III)).
- [299] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach*. Pearson Education, 2000 (cit. on p. 5 (III)).
- [300] C. Gilbertson, M. Mundt, J. Teves, S. Toribio and R. Milewicz, ‘Towards evidence-based software quality practices for reproducibility: Practices and aligned software qualities,’ in *ACM-REP*, 2024 (cit. on p. 5 (III)).
- [301] S. Mäkinen, H. Skogström, E. Laaksonen and T. Mikkonen, ‘Who needs mlops: What data scientists seek to accomplish and how can mlops help?’ In *WAIN*, 2021 (cit. on p. 5 (III)).
- [302] R. Ranawana and A. S. Karunananda, ‘An agile software development life cycle model for machine learning application development,’ in *SLAAI-ICAI*, 2021 (cit. on p. 5 (III)).
- [303] M. Saenz, E. Revilla and C. Simón, *Designing AI systems with human-machine teams*. 2020 (cit. on p. 5 (III)).
- [304] S. Hove and B. Anda, ‘Experiences from conducting semi-structured interviews in empirical software engineering research,’ in *METRICS*, 2005 (cit. on p. 9 (III)).
- [305] M. Corrente and I. Bourgeault, *Innovation in transcribing data: Meet otter. ai*. SAGE Publications, Ltd., 2022 (cit. on p. 9 (III)).
- [306] P. Elger and E. Shanaghy, *AI as a Service: Serverless machine learning with AWS*. Manning Publications, 2020 (cit. on p. 13 (III)).
- [307] G. Van den Broeck, A. Lykov, M. Schleich and D. Suciuc, ‘On the tractability of shap explanations,’ *JAIR*, vol. 74, pp. 851–886, 2022 (cit. on p. 13 (III)).

- [308] K. Vlaanderen, S. Jansen, S. Brinkkemper and E. Jaspers, ‘The agile requirements refinery: Applying scrum principles to software product management,’ *IST*, 2011 (cit. on p. 14 (III)).
- [309] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner *et al.*, ‘Hidden technical debt in machine learning systems,’ in *NeurIPS Press*, MIT Press, 2015 (cit. on pp. 2 (IV), 2 (IV)).
- [310] G. A. Lewis, I. Ozkaya and X. Xu, ‘Software architecture challenges for ML systems,’ in *ICSME*, 2021 (cit. on pp. 2 (IV), 2 (IV)).
- [311] H. Muccini and K. Vaidhyanathan, *Software architecture for ML-based systems: What exists and what lies ahead*, 2021 (cit. on pp. 2 (IV), 2 (IV), 8 (IV)).
- [312] F. Kumeno, ‘Software engineering challenges for machine learning applications: A literature review,’ *Intelligent Decision Technologies*, 2020 (cit. on p. 2 (IV)).
- [313] K. R. Thórisson, ‘Integrated AI systems,’ *Minds and Machines*, 2007 (cit. on p. 2 (IV)).
- [314] U. S. D. of Defence, ‘Component models,’ *DODAF Report*, 2023 (cit. on pp. 2 (IV), 6 (IV)).
- [315] F. Wedyan and S. Abufakher, ‘Impact of design patterns on software quality: A systematic literature review,’ *IET Software*, 2020. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2018.5446> (cit. on pp. 2 (IV), 2 (IV), 9 (IV), 9 (IV), 42 (IV)).
- [316] V. Indykov, R. Wohlrab and D. Strüber, ‘Quality trade-offs in ML-enabled systems: A multiple-case study,’ in *SIGAPP SAC*, ACM, 2025 (cit. on pp. 3 (IV), 15 (V)).
- [317] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 2012 (cit. on p. 3 (IV)).
- [318] International Organization for Standardization (ISO), ‘System and software quality models,’ ISO, International Standard, 2011 (cit. on pp. 4 (IV), 4 (IV), 4 (IV), 4 (IV)).
- [319] V. Indykov, D. Strüber and R. Wohlrab, ‘Architectural tactics to achieve quality attributes of machine-learning-enabled systems: A systematic literature review,’ *JSS*, 2025 (cit. on pp. 4 (IV), 4 (IV), 4 (IV), 4 (IV), 5 (IV)).
- [320] K.-K. Lau, ‘Software component models,’ in *ICSE*, 2006 (cit. on p. 6 (IV)).
- [321] E. Gamma, R. Helm, R. Johnson and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994 (cit. on pp. 6 (IV), 21 (IV)).
- [322] P. Bourque, R. E. Fairley and I. C. Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 4.0*. IEEE Computer Society Press, 2024 (cit. on pp. 6 (IV), 12 (IV)).

- [323] V. Garousi, M. Felderer and M. V. Mäntylä, ‘Guidelines for including grey literature and conducting multivocal literature reviews in software engineering handbook-of-research-synthesis,’ *Information and Software Technology*, 2019 (cit. on pp. 6 (IV), 7 (IV), 7 (IV), 11 (IV)).
- [324] V. Garousi, M. Felderer and M. V. Mäntylä, ‘The need for multivocal literature reviews in software engineering,’ in *EASE*, 2016 (cit. on p. 6 (IV)).
- [325] G. T. G. Neto, W. B. Santos, P. T. Endo and R. A. Fagundes, ‘Multivocal literature reviews in software engineering: Preliminary findings from a tertiary study,’ in *EASE*, 2019 (cit. on p. 6 (IV)).
- [326] H. Cooper, L. Hedges and J. Valentine, ‘The handbook of research synthesis and meta-analysis 2nd edition,’ in *The Hand. of Res. Synthesis and Meta-Analysis, 2nd Ed.* Russell Sage Foundation, 2009 (cit. on p. 7 (IV)).
- [327] J. Schöpfel, ‘Towards a prague definition of grey literature,’ in *International Conference on Grey Literature*, 2010 (cit. on p. 7 (IV)).
- [328] R. J. Adams, P. Smart and A. S. Huff, ‘Shades of grey: Guidelines for working with the grey literature in systematic reviews for management and organizational studies,’ *International Journal of Management Reviews*, 2017 (cit. on p. 7 (IV)).
- [329] B. Kitchenham, L. Madeyski and D. Budgen, ‘How should software engineering secondary studies include grey material?’ *TSE*, 2023 (cit. on p. 7 (IV)).
- [330] V. Garousi, M. Felderer, M. V. Mäntylä and A. Rainer, ‘Benefitting from the grey literature in software engineering research,’ in Springer, 2020 (cit. on p. 7 (IV)).
- [331] N. Maslej, L. Fattorini and E. Brynjolfsson, *Artificial intelligence index report*, 2023 (cit. on pp. 7 (IV), 12 (IV)).
- [332] F. P. B. Jr, *The Mythical Man-Month, Essays on software engineering.* Addison-Wesley, 1972 (cit. on p. 8 (IV)).
- [333] D. E. Perry and A. L. Wolf, ‘Foundations for the study of software architecture,’ *SIGSOFT Softw. Eng. Notes*, 1992. [Online]. Available: <https://doi.org/10.1145/141874.141884> (cit. on p. 8 (IV)).
- [334] D. Garlan and M. Shaw, ‘An introduction to software architecture,’ Carnegie Mellon University, Tech. Rep., 1994 (cit. on p. 8 (IV)).
- [335] E. Woods and N. Rozanski, ‘Using architectural perspectives,’ in *WICSA*, 2005 (cit. on p. 8 (IV)).
- [336] E. Woods, ‘Software architecture in a changing world,’ *IEEE Software*, 2016 (cit. on pp. 8 (IV), 8 (IV)).
- [337] A. Vogelsang and M. Borg, ‘Requirements engineering for machine learning: Perspectives from data scientists,’ in *REW*, 2019 (cit. on p. 8 (IV)).

- [338] B. Bafandeh Mayvan, A. Rasoolzadegan and Z. Ghavidel Yazdi, ‘The state of the art on design patterns: A systematic mapping of the literature,’ *JSS*, 2017 (cit. on pp. 8 (IV), 9 (IV), 42 (IV), 43 (IV)).
- [339] Q. Lu, L. Zhu, X. Xu, J. Whittle, D. Zowghi and A. Jacquet, *Responsible AI pattern catalogue: A collection of best practices for ai governance and engineering*, 2023 (cit. on p. 8 (IV)).
- [340] J. Juziuk, D. Weyns and T. Holvoet, ‘Design patterns for multi-agent systems: A systematic literature review,’ in *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Springer, 2014 (cit. on p. 8 (IV)).
- [341] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E. B. Fernandez and N. Yoshioka, ‘Landscape of architecture and design patterns for IoT systems,’ *IoT*, 2020 (cit. on p. 8 (IV)).
- [342] D. Taibi, V. Lenarduzzi and C. Pahl, ‘Architectural patterns for microservices: A systematic mapping study,’ in *CLOSER*, 2018 (cit. on p. 8 (IV)).
- [343] H. Washizaki, H. Uchida, F. Khomh and Y.-G. Guéhéneuc, ‘Studying software engineering patterns for designing machine learning systems,’ in *IWESEP*, 2019 (cit. on pp. 9 (IV), 9 (IV)).
- [344] H. Washizaki, F. Khomh, Y. Gueheneuc *et al.*, ‘Software-engineering design patterns for machine learning applications,’ *Computer*, 2022 (cit. on pp. 9 (IV), 9 (IV), 16 (IV), 41 (IV), 41 (IV), 41 (IV)).
- [345] L. Heiland, M. Hauser and J. Bogner, *Design patterns for AI-based systems: A multivocal literature review and pattern repository*, 2023 (cit. on pp. 9 (IV), 9 (IV), 16 (IV), 16 (IV), 20 (IV), 21 (IV), 23 (IV), 26 (IV), 27 (IV), 27 (IV), 27 (IV), 28 (IV), 41 (IV), 41 (IV), 41 (IV), 42 (IV)).
- [346] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley, 1996 (cit. on p. 9 (IV)).
- [347] G. Márquez and H. Astudillo, ‘Actual use of architectural patterns in microservices-based open source projects,’ in *APSEC*, 2018 (cit. on pp. 9 (IV), 9 (IV)).
- [348] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin and J. Ocharán-Hernández, ‘Patterns related to microservice architecture: A multivocal literature review,’ *Programming and Computer Software*, 2020 (cit. on pp. 9 (IV), 9 (IV)).
- [349] M. Rahman, M. S. H. Chy and S. Saha, ‘A systematic review on software design patterns in today’s perspective,’ in *SeGAH*, 2023 (cit. on p. 9 (IV)).
- [350] E. Eriksson, J. Olausson, V. Indykov, D. Strüber and R. Wohlrab, *Supplementary artifact*, doi.org/10.6084/m9.figshare.28023146, 2025 (cit. on pp. 11 (IV), 16 (IV), 17 (IV)).

- [351] Google, *Why your google search results might differ from other people*, Experience Report, 2024. [Online]. Available: <https://support.google.com/websearch/answer/12412910> (cit. on pp. 12 (IV), 13 (IV)).
- [352] Y. Wu, P. Gupta, M. Wei, Y. Acar, S. Fahl and B. Ur, ‘Your secrets are safe,’ in *World Wide Web Conference*, 2018 (cit. on p. 13 (IV)).
- [353] S. Hove and B. Anda, ‘Experiences from conducting semi-structured interviews in empirical software engineering research,’ in *METRICS*, 2005 (cit. on p. 17 (IV)).
- [354] A. Singh, *Comprehensive guide for ensemble models*, 2023 (cit. on p. 20 (IV)).
- [355] X. H. Vu, X. D. Hoang and T. H. H. Chu, ‘A novel model based on ensemble learning for detecting DGA botnets,’ in *KSE*, 2022 (cit. on pp. 20 (IV), 21 (IV), 25 (IV)).
- [356] Z. Peng, J. Yang, T.-H. Chen and L. Ma, ‘A first look at the integration of machine learning models in complex autonomous driving systems: A case study on apollo,’ *ACM*, 2020 (cit. on pp. 22 (IV), 24 (IV)).
- [357] M. Abirami, R. Anand and A. M. Bhaskaran, ‘Facial image-based age and gender classification to enhance recommendations in a smart TV environment,’ in *ICNWC*, 2023 (cit. on p. 23 (IV)).
- [358] S. M. Anjum, K. M. Malik, H. Soltanian-Zadeh, H. Malik and G. Malik, ‘Saccular brain aneurysm detection and multiclassifier rupture prediction using digital subtraction and magnetic resonance angiograms,’ in *BBE*, *ACM*, 2018 (cit. on p. 24 (IV)).
- [359] I. Guyon, S. Gunn, M. Nikravesh and L. Zadeh, *Feature Extraction: Foundations and Applications*. Springer, 2008 (cit. on p. 25 (IV)).
- [360] S. Khalid, T. Khalil and S. Nasreen, ‘A survey of feature selection and feature extraction techniques in machine learning,’ in *2014 Science and Information Conference*, 2014 (cit. on p. 25 (IV)).
- [361] A. D. Jain and C. M. Hera, *Noise reduction using ML*, Patent Documentation, 2018. [Online]. Available: <https://patents.google.com/patent/US20190122689A1/en> (cit. on p. 26 (IV)).
- [362] I. Awaad and B. León, ‘XPERSIF: A software integration framework and architecture for robotic learning by experimentation,’ Ph.D. dissertation, Bonn-Rhein-Sieg High School, 2008 (cit. on p. 27 (IV)).
- [363] F. Petroni, N. Raman, T. Nugent *et al.*, ‘An extensible event extraction system with cross-media event resolution,’ in *SIGKDD*, *ACM*, 2018 (cit. on p. 28 (IV)).
- [364] L. Dorard, *9 components of a real-world machine learning system*, Experience Report, 2020. [Online]. Available: <https://www.linkedin.com/pulse/9-components-real-world-machine-learning-system-louis-dorard/> (cit. on p. 28 (IV)).

- [365] Y. Zi, Y. Luo, Z. Guang, L. Qi, T. Wu and X. Zhang, ‘Anomalous taxi route detection system based on cloud services,’ in *Cloud Computing, Smart Grid and Innovative Frontiers in Telecommunications*, 2020 (cit. on p. 29 (IV)).
- [366] Commonwealth Scientific and Industrial Research Organisation (CSIRO), *Multi-model decision maker*, Service Documentation, 2023. [Online]. Available: <https://research.csiro.au/ss/science/projects/responsible-ai-pattern-catalogue/multi-model-decision-maker/> (cit. on p. 29 (IV)).
- [367] A. W. Services, *ML lifecycle architecture diagram*, Service Documentation, 2024. [Online]. Available: <https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/ml-lifecycle-architecture-diagram.html> (cit. on p. 30 (IV)).
- [368] S. Sheikhi and S. M. Babamir, ‘A predictive framework for load balancing clustered web servers,’ *Journal of Supercomputing*, 2016. [Online]. Available: <https://doi.org/10.1007/s11227-015-1584-8> (cit. on p. 30 (IV)).
- [369] P. Runeson and M. Höst, ‘Guidelines for conducting and reporting case study research in software engineering,’ *ESE*, 2009 (cit. on pp. 41 (IV), 42 (IV), 42 (IV)).
- [370] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on pp. 41 (IV), 42 (IV), 42 (IV)).
- [371] V. Indykov, D. Strüber and R. Wohlrab, ‘Architectural tactics to achieve quality attributes of machine-learning-enabled systems: A systematic literature review,’ *JSS*, 2025 (cit. on pp. 2 (V), 15 (V)).
- [372] D. A. Seale and A. Rapoport, ‘Sequential decision making with relative ranks,’ *Organ. Behav. Hum. Decis. Process*, 1997 (cit. on p. 2 (V)).
- [373] A. J. London, ‘Artificial intelligence and black-box medical decisions: Accuracy versus explainability,’ *Hastings Center Report*, 2019 (cit. on p. 3 (V)).
- [374] M. Moshref, M. Yu and R. Govindan, ‘Resource/accuracy tradeoffs in software-defined measurement,’ in *SIGCOMM*, ACM, 2013 (cit. on p. 3 (V)).
- [375] M. M. John, H. H. Olsson, J. Bosch and D. Gillblad, ‘Exploring trade-offs in MLOps adoption,’ in *APSEC*, IEEE, 2023 (cit. on pp. 3 (V), 9 (V)).
- [376] M. Hanna, L. Pantanowitz, B. Jackson *et al.*, ‘Ethical and bias considerations in artificial intelligence,’ *Modern Pathology*, 2024 (cit. on p. 3 (V)).
- [377] G Bou Ghantous and A. Gill, ‘DevOps: Concepts, practices, tools, benefits and challenges,’ *PACIS*, 2017 (cit. on p. 5 (V)).

- [378] V. Casola, A. De Benedictis, M. Rak and G. Salzillo, ‘A cloud SecDevOps methodology: From design to testing,’ in *QUATIC*, Springer, 2020 (cit. on p. 5 (V)).
- [379] J. Storment and M. Fuller, *Cloud FinOps*. O’Reilly, 2023 (cit. on p. 5 (V)).
- [380] H. Atwal, ‘Practical DataOps,’ *Practical DataOps*, 2020 (cit. on p. 5 (V)).
- [381] Y. Dang, Q. Lin and P. Huang, ‘AIOps: Real-world challenges and research innovations,’ in *ICSE*, IEEE, 2019 (cit. on p. 5 (V)).
- [382] M. Testi, M. Ballabio, E. Frontoni, G. Iannello and S. Moccia, ‘MLOps: A taxonomy and a methodology,’ *IEEE Access*, 2022 (cit. on p. 5 (V)).
- [383] W. Hummer, V. Muthusamy, T. Rausch *et al.*, ‘ModelOps: Cloud-based lifecycle management for reliable and trusted AI,’ in *IC2E*, IEEE, 2019 (cit. on p. 6 (V)).
- [384] J. Diaz-De-Arcaya, J. López-De-Armentia, R. Miñón, I. L. Ojanguren and A. I. Torre-Bastida, ‘Large language model operations (LLMOps): Definition, challenges, and lifecycle management,’ in *SpliTech*, IEEE, 2024 (cit. on p. 6 (V)).
- [385] A. Melnikov and H. Birkholz, *IOT operations*, IETF, 2024 (cit. on p. 6 (V)).
- [386] X. Zhang and J. Jaskolka, ‘Conceptualizing the secure machine learning operations (SecMLOps) paradigm,’ in *QRS*, IEEE, 2022 (cit. on p. 6 (V)).
- [387] M. Zeller, T. Waschulzik, R. Schmid and C. Bahlmann, ‘Toward a safe MLOps process for the continuous development and safety assurance of ML-based systems in the railway domain,’ *AI and Ethics*, 2024 (cit. on p. 6 (V)).
- [388] Y. Billeter, P. Denzel, R. Chavarriaga *et al.*, ‘MLOps as enabler of trustworthy AI,’ in *SDS*, IEEE, 2024 (cit. on p. 6 (V)).
- [389] S. Gupta, W. Zhang and F. Wang, ‘Model accuracy and runtime tradeoff in distributed deep learning: A systematic study,’ in *ICDM*, IEEE, 2016 (cit. on p. 10 (V)).
- [390] L. Ardito, R. Coppola, L. Barbato and D. Verga, ‘A tool-based perspective on software code maintainability metrics: A systematic literature review,’ *Sci. Program.*, 2020 (cit. on p. 10 (V)).
- [391] E. Milanzi, E. Njeru Njagi, L. Bruckers and G. Molenberghs, ‘Data representativeness: Issues and solutions,’ *EFSA*, 2015 (cit. on p. 10 (V)).
- [392] K. W. Al-Sabbagh, M. Staron and R. Hebig, ‘Improving test case selection by handling class and attribute noise,’ *JSS*, 2022 (cit. on p. 9 (V)).
- [393] T. R. Hoens, R. Polikar and N. V. Chawla, ‘Learning from streaming data with concept drift and imbalance: An overview,’ *AI*, 2012 (cit. on p. 9 (V)).

- [394] J.-M. Horcas, D. Strüber, A. Burdusel, J. Martinez and S. Zschaler, ‘We’re not gonna break it! consistency-preserving operators for efficient product line configuration,’ *IEEE TSE*, 2022 (cit. on p. 11 (V)).
- [395] G. Varoquaux and O. Colliot, ‘Evaluating machine learning models and their diagnostic value,’ in *Machine learning for brain disorders*, Springer, 2023 (cit. on p. 11 (V)).
- [396] R. Kazman, M. Klein and P. Clements, *ATAM: Method for architecture evaluation*. Carnegie Mellon University, USA, 2000 (cit. on p. 17 (V)).

Part II

Appended Papers



**Component-based Approach to Software  
Engineering of Machine Learning-enabled  
Systems**

**V. Indykov**

Proceedings of the 3rd IEEE/ACM International Conference on AI  
Engineering (CAIN), 2024, pp. 250-252

# Abstract

Machine Learning (ML) - enabled systems capture new frontiers of industrial use. The development of such systems is becoming a priority course for many vendors due to the unique capabilities of Artificial Intelligence (AI) techniques. The current trend today is to integrate ML functionality into complex systems as architectural components. There are a lot of relevant challenges associated with this strategy in terms of the overall system architecture and in the context of development workflow (MLOps). The probabilistic nature, crucial dependency on data, and work in an environment of high uncertainty do not allow software engineers to apply traditional software development methodologies. As a result, there is a community request to systematize the most relevant experience in building software architectures with ML components, to create new approaches to organizing the process of developing ML-enabled systems, and to build new models for assessing the system quality. Our research contributes to all the mentioned directions and aims to create a methodology for the efficient implementation of ML-enabled software and AI components. The results of the research can be used in the design and development in industrial settings, as well as a basis for further studies in the research field, which is of both practical and scientific value.

# Chapter 1

## Introduction

*Machine learning (ML) engineering* is an emerging field of research that lies at the intersection of *software engineering* and *artificial intelligence (AI) development*. Great interest in this area from the scientific community, practitioners, and industries has been observed in recent years due to the dynamic spread of AI techniques in various fields. This PhD project strives to obtain the most relevant experience in the field of ML engineering, which includes an analysis of grey and white literature.

The current trend today is to integrate ML functionality into complex systems as architectural components [9]. Following this trend, many practitioners face a number of challenges due to AI specifics. There are several still unresolved issues associated with both the unstable quality of architectural solutions and the relatively low efficiency of the development process [50].

To address architectural challenges, the main emphasis of the research is put on *Component-based Software Engineering (CBSE) of ML-enabled systems*. The *CBSE* promotes software development through construction from existing software components, the development of components as reusable entities, and system evolution realization by the customization and replacement of components [51]. The operational side of software engineering of ML-enabled systems, known as *Machine Learning Operations (MLOps)*, is also of particular interest for this research. *MLOps* is a set of principles and practices adopted from Development Operations (DevOps) and applied to the development of machine learning systems [52].

Previous studies were conducted to identify current issues in the CBSE of AI-enabled software. There were a systematic literature review [53], a multi-vocal literature review from leading AI companies [54], and case studies in Swedish companies [55]. These works identified a wide range of challenges that can be classified into several categories from technical to organizational: problems in mixing data and code in version management and dependency management, prediction non-linearity, problems in the inability to ensure quality assurance, unreliable project planning, etc. Based on the results, it was concluded that several challenges have been experienced, but the overall solution for seamless continuous development, integration, deployment, operations, and evolution is

still missing. The goal of this PhD project is to address this gap.

**The main research hypothesis:** “The approaches from component-based software engineering (both technical and operational) can be tailored for solving the architectural challenges created by AI specifics”.

## 1.1 Related work

Nowadays, we are witnessing the dynamic growth of scientific activity in the field of ML-enabled software engineering. Some works describe how the engineering of AI-based systems is implemented in separate companies. Amershi et. al [56] shared the experience of Microsoft and highlighted the need for detailed studies of CBSE for ML-enabled systems, Google software engineers [57] presented their observations and emphasized the crucial role of hidden technical debt consideration in MLOps, SAP and Polytechnique Montreal [58] reported some recommendations for building ML applications and highlighted “a growing need for a consolidated set of guidelines regarding software engineering for machine learning”.

Some authors conducted systematic literature reviews and concluded that “mature tools and techniques are missing to engineer ML systems” [59], the application of software design patterns to AI-based systems needs deeper investigation [60] and the existing fundamental differences between the development of traditional and ML-enabled software requires systematic study of corresponding software architectures [61]. An analysis of related work in the research area has shown the emergence of an increasing number of high quality papers, that identify practical and theoretical challenges in constructing AI-based systems and describe special cases of their solution [24], [62]–[65]. However, most studies continue to highlight the need for in-depth research continuation and systematic exploration of the problems and possible decisions associated with the design of ML-enabled system architectures. There is an in-demand request for system-level common recommendations that would be relevant for the majority of ML-enabled solutions or divided by certain types of ML solutions (e.g. Deep Learning (DL)-based, Reinforcement Learning (RL) - based etc.).

Therefore, **the research problem** is formulated as “the lack of systematic work that brings together relevant scientific and practical experience in building architectures for ML-enabled systems and organizing the process of ML-enabled software development”.

# Chapter 2

## Contributions

The main result of the research will be a new multifaceted approach to the design and development of ML-enabled systems. The project examines ML-enabled software engineering from different angles, however, all research contributions are united by one goal - increasing the quality of the development. Moreover, each subsequent contribution uses the results of the previous one as an input to obtain a new result. The approach will include the identification of common indicators for quality assurance (qualitative side); the design of a quality-driven reference architecture to achieve quality attributes identified above built on the most relevant practical and scientific experience and tested in industrial settings (architectural side); and the development of guidance for ML-enabled software engineers built on the best practices from experts for effective implementation of reference architectures (operational side). The description of contributions can be found below:

**Common Quality Model of ML-enabled systems.** A systematic white literature review in the field of ML-enabled software architectures identified a critical need to create a model that describes the quality of such systems and considers their specificity. More than 60 scientific papers were analyzed to exclusively identify quality attributes of ML-enabled systems. It is planned to test the relevance and adequacy of this model in industrial conditions through an expert interview and assessment. An initial evaluation of this model was conducted during its demonstration at the Swedish Requirements Engineering Meeting (SIREN 2023) and got positive appraisal from practitioners. The resulting model was descriptive according to DAP classification [66]. It was included in a scientific paper entitled “Architectural Tactics to Achieve Common Quality Attributes of Machine-Learning-Enabled Systems” that was submitted to an International Conference in December 2023 [67].

**Quality-driven Software Reference Architecture.** To achieve the identified quality attributes, various architectural tactics (ATs) were studied. In the absolute majority of cases, quality trade-offs arose (for example, architectural tactics can improve the model accuracy, but worsen resource efficiency). The study of ATs and trade-offs was also included in the systematic literature review mentioned earlier [67]. In the course of further research, it is necessary to

explore design patterns and component models of existing AI-based solutions, as well as their relationship with quality attributes. The results of the study will be combined into reference architecture of ML-enabled systems, which can serve as the primary template for software engineers to architecturally achieve certain qualities. In the context of our project, the reference architecture corresponds to “Type 1” (Low-level) according to classification by Angelov et. al [68]. It remains to be explored whether it can be domain- and model-type-independent. If it is not, the development of several reference architectures is possible. The resulting reference architecture(s) will be evaluated by interviews with experts and relevant case studies.

**Modelling Guidance for Software Architects.** A significant part of the studied industrial experience can describe not only the architectures but also the process of their creation. It includes several organizational decisions, guidelines, and tooling. The collected experience can be compiled into guidance for software architects in the field of ML-enabled systems. This guidance is planned to be presented as a collection of best practices, which will give the architects flexibility in organizing the process. They will be able to use separate best practices as components of their own project without forcing themselves into a strict process model. In addition, various possible implementations of automated tools for supporting software architects based on the guidance are considered: modeling assistants, automated techniques for enforcing the correct implementation of reference architecture, and validation of conformance of the implemented system’s behavior to architectural decisions based on run-time data. The results will be evaluated by a focus group of practitioners.

## Chapter 3

# Conclusion

The results of this research can be considered as an extension to the Software Engineering Body of Knowledge (SWEBoK) [16] that explores the specifics of machine learning development in the context of overall system design. This PhD project contributes to the following SWEBoK's Knowledge Areas: Software Design, Software Engineering Management and Software Engineering Process, Software Quality, and Software Engineering Professional Practice. Other knowledge areas are left out of scope due to the time frames, however, they have great potential for further studies.

**Architectural Tactics to Achieve Quality  
Attributes of Machine-Learning-Enabled  
Systems: a Systematic Literature Review**

V. Indykov, D. Strüber, R. Wohlrab

Journal of Systems & Software (JSS), 2025, vol. 223, no. 112373, pp. 1-20

# Abstract

Machine-learning-enabled systems are becoming increasingly common in different industries. Due to the impact of uncertainty and the pronounced role of data, ensuring the quality of such systems requires consideration of several unique characteristics in addition to traditional ones. This range of quality attributes can be achieved by the implementation of specific architectural tactics. Such architectural decisions affect the further functioning of the system and its compliance with business goals. Architectural decisions have to be made with attention to possible quality trade-offs to prevent the cost of mitigating unintended side effects. A related work analysis revealed the need for a thorough study of existing architectural decisions and their impact on various quality attributes in the context of machine-learning-enabled systems. In this paper, to address this goal, we present comprehensive research on the quality of such systems, architectural tactics, and their possible quality consequences. Based on a systematic literature review of 206 primary sources, we identified 11 common quality attributes, and 16 relevant architectural tactics together along with 85 potential quality trade-offs. Our results systematize existing research in building architectures of ML-enabled systems. They can be used by software architects and researchers at the system design stage to estimate the possible consequences of decisions made.

# Chapter 1

## Introduction

Machine-learning-enabled (ML-enabled) systems [69] are currently in high demand among various spheres. The design, development, and implementation of such systems are widespread now since ML technologies allow organizations to reach results that are difficult to achieve through traditional solutions. Machine learning systems typically work with large volumes of data, adapt, learn, search for, and process complex correlations. The development of AI-based systems is an extremely relevant strategy for the world’s largest vendors: Meta is implementing ML components for content moderation and feed personalization, Microsoft is focused on developing the AI companion called “Microsoft Copilot”, the use of large language models is conquering new frontiers. However, designing such systems remains a non-trivial and non-standardized task due to the lack of detailed system-level guidelines and instructions for constructing appropriate architectures with a consideration of system specifics.

The construction of ML-based software architecture starts with the collection of requirements, particularly, non-functional ones, also known as *quality attributes*. They must be considered at the stage of architectural design to align the system with the intended goals. As Monson stated: “*You don’t drive the architecture, the requirements do. You do your best to serve their needs*” [70].

There are several detailed specifications and standards (e.g., ISO/IEC 25010 [71] and ISO/IEC 45010 [72]) for traditional software that makes it possible to predetermine the fundamental quality attributes of the designed system without conducting any deep research. Some of their quality characteristics can be adapted, updated, and extended to directly meet the needs of ML-based software due to its unique characteristics compared to traditional ones. Specifically, ML-enabled systems operate in environments of high uncertainty and depend on the quality and quantity of data used for model training, validation, and testing. This fact and its relevance were confirmed by the ISO/IEC 25059 [73] issued in June 2023, which adjusted some of the existing qualities from ISO/IEC 25010 to ML contexts and additionally considered several ML-specific aspects (e.g., ethics, transparency). While this new standard presents an important initiative for addressing the specific quality aspects of ML-enabled systems, it has not been investigated to which extent it characterizes the relevant aspects

of this domain exhaustively. Such an investigation could be supported by a systematic study, as we perform in this work.

Quality attributes can be achieved by architectural and non-architectural tactics. Non-architectural ways of achieving quality are based on organizational non-technical management and on technical decisions that do not affect software architecture. Such decisions are too dependent on the system specifics and are out of our scope. Architectural tactics, on the contrary, are general design decisions. They are designed to improve one specific target quality attribute. In practice, it is very common that architectural tactics entail unanticipated tradeoffs on other quality attributes. To raise awareness of the consequences that come with a selection of architectural tactics, it is important to make these tradeoffs explicit. This is one of the contributions of this paper and will enable architects to more deliberately select architectural tactics for ML-enabled systems in the future.

For example, there is an architectural tactic to implement a real-time data monitoring module. In the context of ML-enabled systems, by implementing this architectural tactic, the operator gets an opportunity to monitor all the data used for model training, testing, and validation as well as dynamic input data. Such a decision can increase fairness, reliability, maintainability, accuracy, and security. However, the main trade-off after the implementation of this tactic appears in terms of resource efficiency when operating with big data [74], [75], [76].

In this paper, we give a comprehensive picture of architectural tactics for the engineering of ML-enabled systems along with quality attributes affected by them. We report on the results of a systematic literature review, in which we extracted information from 206 primary sources about these aspects. Specifically, we made the following contributions:

1. We propose a quality model for ML-enabled systems, focused on the most commonly reported quality attributes in the literature, and compare it to the relevant standards of ISO/IEC 25010 [71] and ISO/IEC 25059 [73].
2. We present a range of architectural tactics that can help achieve identified common quality attributes.
3. We present an analysis of the quality trade-offs of the identified architectural tactics, summarized as an impact matrix.

This paper is accompanied by a supplementary artifact<sup>1</sup> which contains search queries and data extraction sheets.

The rest of this paper is structured as follows: Section 2 discusses related work and introduces the used terminology. Section 3 describes our research methodology. Section 4 presents our results, including the identification and analysis of quality attributes, architectural tactics, and quality trade-offs. Section 5 discusses implications concerning specific attributes, other quality standards, and threats to validity. Section 6 concludes and outlines future work.

---

<sup>1</sup>Supplementary Artifact: <https://doi.org/10.6084/m9.figshare.25673643.v1>

# Chapter 2

## Background

### 2.1 Context

A *quality attribute (QA)* is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders [77]. In ISO/IEC 9126-1:2001, quality attributes are described as a “*checklist to determine software quality*” [78]. According to Lundberg et al. [79], the quality attributes should guide the design of the software architecture. While stakeholders, usage contexts and, therefore, relevant quality attributes differ from one system to another, one can identify the most widespread quality attributes applied to systems of different natures. In the context of this work, we call them the “*common quality attributes*” (*CQAs*).

Quality attributes are related to the term “architecturally significant requirement(s)”. However, the latter is entirely specialized to a particular system, based on the needs of certain stakeholders, technical capabilities, internal regulations, etc. “*In gathering [architecturally significant requirements], we should be mindful of the business goals of the organization*” [77]. In this paper, we seek to generalize existing experience, putting the specifics of individual systems aside.

An *architectural tactic (AT)* is a “*technique an architect can use to achieve the required quality attributes*” [77]. By definition, the connection between tactic and certain *quality attributes* is implied. However, our study goes further and analyzes the impact of its influence on all identified common quality attributes. Balance or compromises between them are called *quality trade-offs*.

### 2.2 Related work

The study of software quality for ML-enabled systems is an in-demand topic among researchers and practitioners [14], [26]. Despite the relatively small number of studies published at the time of writing the current paper, a steady positive trend in this domain was noted. The space for interpreting the quality of AI systems has only been partially explored and a conclusive view is yet to

form, which is proved by the emergence of different quality models based on industrial experience [27], [28], [29]. Such studies work with non-functional requirements relevant to a certain system and most often receive them from domain experts. The generalizability of such models can be debatable due to context dependence. Their systematization and the identification of the most common quality attributes is a way to build a more generalized picture based on real examples. Such a strategy supports a collection of the most recent materials and makes current research more independent from external inputs.

There are also plenty of review papers on architectural issues in the context of AI-based systems [30], [31], [32]. These papers explore a collection of existing architectural design decisions without a clear reference to system qualities or with a focus on the impact of decisions on individual quality attributes and their metrics in isolation from the overall quality picture of the system. As a result, possible trade-offs often remain unnoticed. In contrast with such studies, we strive to investigate the effects of architectural tactics (ATs) on all the identified quality attributes to provide insights at the architectural level.

# Chapter 3

## Methodology

The methodology of *systematic literature review (SLR)* allowed us to work with a large amount of scientific information, find common approaches to different systems, and effectively extract information from different sources. Such opportunities suit the research in the chosen domain. We decided to perform an SLR according to Kitchenham's guidelines [80] as we found them most detailed and highly applicable to the current study of software architectures.

### 3.1 Review Questions

To achieve the research objectives, three fundamental review questions (RQs) were identified.

**RQ1:** *What are the most frequently reported quality attributes for ML-enabled systems?* This question aims to identify the most often emphasized QAs in scientific literature.

**RQ2:** *What architectural tactics have been reported to be effective for ML-enabled systems?* This question aims to identify ATs to achieve quality attributes defined in RQ1. If the quality attribute can not be satisfied by any AT, then it is out of scope for RQ2 and RQ3.

**RQ3:** *For each architectural tactic, what is the reported impact on all the identified quality attributes?* This question aims to identify quality trade-offs when ATs defined in RQ2 are implemented.

### 3.2 Inclusion and Exclusion Criteria

Only scientific literature was analyzed in this work, leaving grey literature outside the scope of this study. We used the following *inclusion criteria*:

1. Research scientific papers containing lists of QAs for specific or general ML-enabled system(s);
2. Research and review scientific papers with the description of ATs and their influence on the QA(s) of specific or general ML-enabled system(s).

We used the following *exclusion criteria*:

1. Grey literature;
2. Scientific papers about QAs of non-ML-enabled systems;
3. Scientific papers about ATs in non-ML-enabled systems;
4. Scientific papers about applying ML to address software quality concerns of non-ML-enabled systems;
5. Scientific papers about applying ML to address architectural concerns of non-ML-enabled systems;
6. Exclusively for RQ1: secondary research (literature reviews).

For our investigation of RQ1, in which we counted the number of occurrences of specific quality attributes in the literature, we deliberately excluded secondary studies. This is to avoid bias that would arise if the same primary study and its contained quality attributes are considered several times: through considered secondary studies and through our own data collection. For RQ2 and RQ3 we found it reasonable to leave secondary research included to expand the search and collect architectural tactics as much as possible. The limitation on grey literature is justified by the availability of a sufficient amount of “white” literature for the current study.

### 3.3 Data Sources

Our search procedure was targeted to enable precise investigation of the identified research questions. To this end, we selected appropriate digital libraries and determined a suitable publication time frame.

*Literature databases.* To build up a high-quality review, only publications from journals and conference proceedings indexed by at least one globally significant citation database (e.g., Scopus, Web of Science, etc.) were analyzed. The five most popular and largest online digital libraries were the sources for this research: IEEE Xplore ([ieeexplore.ieee.org](http://ieeexplore.ieee.org)), ACM Digital Library ([dl.acm.org](http://dl.acm.org)), Springer ([springerlink.com](http://springerlink.com)), Elsevier ([sciencedirect.com](http://sciencedirect.com)), Wiley ([onlinelibrary.wiley.com](http://onlinelibrary.wiley.com)).

*Time frame.* Since this study seeks to explore the most relevant experience in the field of ML-enabled system design, we decided to limit the number of papers with the earliest date of publication of 2011. This decision was made also in connection with the release of the most recent version of the ISO/IEC 25010, which dates to 2011 [71]. This standard is important for the study since this research seeks, in some sense, to clarify the list of quality attributes from it with a consideration of the ML-enabled specifics and recent research experience. Thus, this review is based on the papers from 2011 to 2024 (the year of writing).

### 3.4 Data Collection

Each review question implies its own objective. The architectural tactics are often not mentioned in works related to software quality and the trade-offs are often not considered in the works on a certain architectural tactic. Thus, we slightly moved away from the standard approach to a systematic literature review with only one query for all review questions and divided our search strategy into three queries, each of which corresponded to its own RQ.

The research under RQ1 works with a set of scientific papers that contains a list of QAs specific to ML-enabled system(s). In the literature, they can be represented explicitly as a list (e.g., a study of Habibullah et al. [81]) or addressed when describing a certain problem or proposing a solution on a system level (e.g., a study of Vojivr et al. [82]). Preliminary research has shown that in the literature on deep learning systems, neural networks, or artificial intelligence systems, the term “machine learning” may not be explicitly stated in the text of the work. Therefore, we decided to expand the query with the above terms to cover a larger number of papers. The introduction of other ML-related terms (such as “MLOps”, “ML Engineering” etc.) could potentially shift focus from architectural scope to a more operational one, while the introduction of other software engineering terms (such as “software quality”) could exclude certain papers that did not explicitly mention them. Therefore, we decided not to include those keywords. The resulting query for RQ1 is presented below:

```
("machine learning" OR "deep learning" OR "artificial intelligence" OR "neural network" OR "AI" OR "ML" OR "DL") AND ("system" OR software) AND ("quality attribute*" OR "quality characteristic*" OR "non-functional requirement*" OR "nonfunctional requirement*" OR "quality model" OR "quality requirement*")
```

Answering RQ2 identifies architectural tactics that improve certain quality attributes. We used search queries based on the results obtained from RQ1, which included common quality attributes (for example, security), together with their sub-characteristics (respectively, privacy). The difficulty of this task is that relevant tactics are not easily identified, since developers might introduce an architectural tactic without referring to it as such. To address this challenge we also included the terms “design pattern” and “architectural decision” in the query. However, we still consider this challenge as a threat to validity and can not argue that the list of collected architectural tactics is complete. Search queries for RQ2 were built according to the template presented below with changing parameters of quality attributes together with their sub-characteristics:

The resulting query for RQ1 is presented below:

```
("machine learning" OR "deep learning" OR "artificial intelligence" OR "neural network" OR "AI" OR "ML" OR "DL") AND (system" OR "software") AND ("common quality attribute" OR "subcharacteristic[1]" OR... OR "subcharacteristic[n]") AND ("*architectur* tactic*" OR "design pattern*" OR "*architectur* design decision*" OR "*architectur* decision*")
```

The research under RQ3 implies the study of all possible impacts (predominantly positive, predominantly negative, or ambivalent) of the identified architectural tactics from RQ2 on the common quality attributes identified in RQ1. For RQ3 we wrote 16 queries (equal to the number of identified architectural tactics). We expected that the connections between some ATs and some QAs would not be addressed, however, the papers that brought some insights are of special usefulness for the current research. The structure of the search queries corresponds to the template presented below and includes all of the studied common quality attributes and their sub-characteristics together with a changing parameter of architectural tactic:

```
("machine learning" OR "deep learning" OR "artificial intelligence" OR "neural network" OR "AI" OR "ML" OR "DL") AND ("common quality attribute[1]" OR ... OR "common quality attribute[n]" OR "subcharacteristic[1]" OR... OR "subcharacteristic[m]") AND ("architectural tactic[i]") AND ("trade-off*" OR "trade off*" OR "tradeoff*" OR "compromise*")
```

We executed the queries sequentially. The results of data extraction from the sources found with the RQ1-query became the input data for the RQ2-queries, the results of which, similarly, became the input for the RQ3-queries. The full search queries for RQ1, RQ2, and RQ3 as well as the process of data collection are presented in the supplementary artifact<sup>1</sup>.

Overall, applying the search procedure with the described queries as well as exclusion and inclusion criteria led to the identification of 206 papers, 37 of which were studied under RQ1, 73 were under RQ2, 96 were under RQ3, and 7 were found for RQ2 but were also found for RQ3 and used to address it.

### 3.5 Data Synthesis

We now discuss the dedicated data synthesis strategies used for each research question as well as our measures taken for ensuring consistency of the data synthesis process.

**RQ1.** The coding strategy for RQ1 is based on content analysis [83] together with basic frequency analysis and taxonomic analysis [84]. First, we employed content analysis to scrutinize the full-text papers to identify possible quality attributes relevant to ML-enabled systems. In the context of our research, content analysis is a manual research method that examines full texts and

---

<sup>1</sup>Supplementary Artifact: <https://doi.org/10.6084/m9.figshare.25673643.v1>

concepts of scientific papers, allowing us to comprehensively detect relevant quality attributes across the studies. In order to extract a certain characteristic mentioned in a paper as a quality attribute, we introduced two main conditions: “*the characteristic must be explicitly mentioned in the paper*” and “*the characteristic must describe the quality of the overall system*” (not a certain algorithm or component).

In parallel, we detected that the number of identified attributes was going to be quite large, however, some of them were mentioned only in a few papers. This fact introduces a threat to the generalizability of our findings since such attributes can potentially describe the specifics of only one specific system. To avoid this threat, we made a scoping decision based on the hypothesis: *The more often an attribute is mentioned in different independent papers, the more cases it covers, and therefore the more generalizable it is.* To count those mentions we employed a basic frequency analysis. Our basic frequency analysis can be considered as a form of coding, where the code of a quality attribute is defined as the number of papers mentioning it. It is worth noting that all the papers had equal weight when extracting attributes. One quality attribute could be mentioned explicitly either once or several times in the text of the one paper, however, it did not affect the calculated frequency. This algorithm was applied to all papers found, resulting in a ranked list of quality attributes. Based on the resulting counters, a dividing line was drawn between the frequently mentioned and less frequently mentioned quality attributes. The latter were not included in the common quality model.

We noticed that several frequently mentioned attributes were semantically closely related (e.g., reliability and trustworthiness) or by definition can be deemed a superset of several other quality attributes (e.g., maintainability usually covered concerns connected to testability, transparency, and maintainability itself). This observation motivated us to employ taxonomic analysis and group quality attributes by semantic similarity to structure the resulting quality model. First, we formulated high-level definitions that discarded the specifics of individual papers, while retaining the fundamental meanings of attributes. Where it was possible, we directly referred to ISO standards ([71],[73]). In other cases we analyzed extra literature to build proper definitions of found attributes. In the studied papers the definitions of quality attributes usually were not mentioned explicitly. Therefore, we analyzed the selected articles again and checked whether our definitions corresponded to the attribute meanings that were implied in them and whether they were relevant in the context of these papers. When the definitions were formulated in a way that satisfied all the cases, we systemized them. We distinguished quality attributes of two levels based on a principle: “*If one quality attribute covers related concerns with certain other attributes and by definition is broader than them, then such an attribute was considered a (“top-level”) common quality attribute, while the other associated attributes were deemed “sub-characteristics”.*” We note that during the research under subsequent RQs, both common quality attributes and their sub-characteristics are included in search queries. Therefore, the main goal of the taxonomic analysis was to build a clearer perception of the resulting quality model, which is presented graphically as a two-level diagram.

**RQ2.** Data synthesis and coding strategies for RQ2 were based exclusively on content analysis. Our goal was to explore all relevant ATs we could find with our search strategy for the scope of common quality attributed as determined in RQ1. Therefore, we did not introduce frequency analysis or taxonomic analysis for RQ2. We thoroughly analyzed full-text papers and followed three conditions for extracting data as ATs: the decision must be explicitly mentioned in a paper, the decision must be architectural in nature (it has an impact on the architectural design principle or can be implemented as a part of the overall system architecture) and the decision must be used to improve some quality attribute(s). Those conditions were introduced with a direct connection to the definition of AT used in this research (see Section 2). If an AT is described as effective in achieving multiple quality attributes, it is associated with all affected attributes. To increase generalizability and eliminate bias, the degree of “significance” of an architectural tactic for a particular attribute was out of scope. For example, if the literature found for RQ2 confirms that the architectural tactic of “containerization” significantly improves both maintainability and portability, then the tactic will be assigned to both attributes, without investigation of which indicator is improved more significantly.

We noticed that some collected tactics only affect the *training system* (e.g., federated learning is usually referred to as a way of organizing model training exclusively), while others can additionally affect other parts of the *deployed system* (e.g., componentization can be the approach to overall system design or be used only to break down the ML pipeline or even the model into components) or be applied to the model when the system is already deployed (e.g., automated bias mitigation usually monitor the outputs of model when it operates with certain inputs). In this context, the *training system* is a system associated with the ML pipeline, which operates with data for model training, testing, and verification; while *deployed system* is a produced ML-enabled system that operates with certain inputs (e.g. real-time data).

ML-enabled systems may include the training system into the overall architecture to introduce continuous retraining and improvement based on new data [85]. However, in some cases, the training system can be relatively independent. Therefore, we decided to introduce a classification of the identified tactics depending on which system they affect: training or deployed. Our findings were presented in tabular format.

It is important to note that the results obtained to some extent generalize the experience described in the literature, which means if a tactic was described as effective for at least one type of ML-enabled system (for example, an IoT system), it was included in the table. Consequently, we cannot guarantee with full certainty optimal efficiency for other types of machine learning systems, which is also considered in the analysis of threats to validity.

**RQ3.** For RQ3, we employed content analysis to identify trade-offs that indicate the impact of implementing architectural tactics on quality attributes. A full-text analysis of the papers identified through our search strategy was performed. We reported an impact of a tactic on a quality attribute if at least one source indicated that applying the tactic influenced metrics or other

indicators for that attribute. When all sources agreed on the impact’s direction, either *predominantly positive* or *negative*, we reported it as such. If sources reported both predominantly positive and negative impacts for the same tactic-quality attribute combination, depending on conditions of the environment or domain, we marked the impact as *ambivalent*. In cases where no evidence of a correlation between an AT and a QA was found, we noted this absence of evidence.

**Data extraction consistency.** Towards ensuring data extraction consistency, we took three measures.

First, we followed the specific advice from the Kitchenham guidelines for performing SLRs [80]. According to them, it sufficient to conduct “*a test-retest process where the researcher performs a second extraction from a random selection of primary studies*”. This second extraction was conducted by Author 1 on a random sample of *10 papers* for RQ1, *15 papers* for RQ2, and *20 papers* for RQ3. The results of this extraction round were identical to the previous attempt for all RQs.

Second, we continuously discussed the data synthesis and its results in the group of authors. Author 1 strictly followed selected search and data synthesis strategies for RQ1, RQ2, and RQ3 sequentially. Whenever a synthesis of results for a particular RQ was completed, a group discussion with all authors was organized. Author 2 and Author 3 based on their expertise provided feedback on whether the search strategy was executed correctly and whether extracted QAs, ATs, or trade-offs corresponded to selected definitions and conditions for their extraction. At each meeting, the review protocol was presented and updated based on the results of the discussion.

Third, the used literary sources are shared in the publicly available supplementary artifact allowing other researchers to follow our algorithm and analyze selected papers. This also enhances the reproducibility of this research.

## 3.6 Results Verification

All the results should be verified by the experts and practitioners to check their relevance for industrial use. We followed several scenarios of validation depending on the contribution.

Our findings for RQ1 which were compiled in the format of the quality model were verified through:

1. *Expert Validation.* The model was presented at the Swedish Requirement Engineering meeting (SiREN 2023). This event brought together academic and practical experts with a background in the field of requirements engineering and machine learning. An assessment was organized in a focus-group setting with oral feedback. Six experts were surveyed sequentially on three main questions:
  - If the proposed model is ‘emphcomplete, i.e. the identified quality attributes exhaustively characterize the quality of ML-enabled systems.
  - If the proposed model is *general*, i.e. the identified quality attributes

are applicable to all types of ML-enabled systems, not only to a certain one.

- If the proposed model is *relevant*, i.e. the identified quality attributes respond to current challenges in ML-enabled software quality assurance.

2. *Practitioner Validation.* The model was presented to four ML engineers from Swedish AI software companies. They checked the proposed model against the key quality characteristics used in their enterprise when designing AI-based systems. The validation used the same evaluation parameters as in the case of expert assessment: completeness, generalizability, and relevance.

The findings for RQ2 which were combined in the final list of architectural tactics and associated quality attributes were verified through practitioner validation. The list of ATs was presented to four ML engineers from Swedish AI software companies. They assessed the applicability of architectural tactics to solve problems encountered in the design of AI-based systems within their company, as well as their theoretical validity for improving system qualities.

The findings for RQ3 which were summarized in the resulting table of trade-offs were verified through internal peer-reviewing, where each co-author checked the plausibility of the identified impact (or absence of such) based on their expertise. This review step did not result in any changes to the findings. An additional verification by practitioners and experts is desirable, however, it is overly laborious for the current study due to the large number of impacts identified. In Section 5, we propose and discuss a strategy for such validation in future work.

# Chapter 4

## Results

### 4.1 RQ1: Identification of Common Quality Model

We examined 37 scientific sources to obtain a comprehensive list of quality attributes that characterize various ML-enabled systems. Table 4.1 provides a list of all quality attributes found and the number of their occurrences in all the sources studied. The list is sorted in descending order of occurrences (*#oc*) of the quality attribute in the papers.

Table 4.1: All retrieved quality attributes of ML-enabled systems

QA	#oc	QA	#oc	QA	#oc	QA	#oc
Fairness	19	Efficiency	12	Ethics	6	Completeness	2
Safety	19	Usability	11	Data quantity	6	Consistency	2
Security	18	Accuracy	10	Traceability	4	Compatibility	2
Explainability	18	Testability	10	Legal	3	Accountability	1
Privacy	17	Correctness	9	Reusability	3	Justifiability	1
Reliability	16	Func. suitability	8	Interoperability	3	Autonomy	1
Performance	16	Interpretability	8	Reproducibility	2	Modifiability	1
Transparency	14	Trustworthiness	8	Integrity	2	Elasticity	1
Robustness	13	Scalability	8	Repeatability	2	Resilience	1
Data Quality	13	Adaptability	6	Retrainability	2		
Maintainability	12	Portability	6	Modularity	1		

**Studies based on interviews and questionnaires.** Several works built models based on the results of interviews, questionnaires, and surveys with experts.

The work of Habibullah et al. [81] contains the most complete list of quality indicators among all the papers studied. The set of QAs was formed through interviews with practitioners in the field of developing ML-enabled systems. The authors collected 37 quality attributes (system non-functional requirements) relevant to product operation, product revision, and product transition, such as efficiency, usability, portability, etc.

Vogelsang [86] identified the structure of common requirements for ML-

enabled systems: functional and non-functional based on the interview results of several data scientists. The group of non-functional requirements (= quality attributes) included: explainability, freedom from discrimination (= fairness), legality, data quantity, and data quality.

To build sustainable AI architectures Kästner et al. [87] indicated six main characteristics of quality assurance based on expert assessment: performance, data quality, testability, safety, security, and fairness.

Agca et al. [88] conducted a comprehensive survey on trusted distributed artificial intelligence. The focus of that paper was not on creating a certain quality model, however, the research addresses such quality attributes as performance, robustness, and transparency.

Various quality models have been proposed by other authors: based on an interview study with ML-project stakeholders [89], [90], industry experts [91], [92], and based on the mixture of qualitative and quantitative studies including a survey of practitioners [61].

**Studies based on expert assessments.** There is a group of work presenting the quality characteristics of ML-enabled systems axiomatically, i.e. the authors list them as relevant or discuss their relevance based on their own expertise. Since we are examining exclusively scientific “white” literature, we consider the authors as experts and find it reasonable to include such works in the list as well.

Yap [93] stated that ML systems have unique requirements arising from the interaction with humans such as fairness, privacy, safety, and security (covering the ML component and overall system security). The key quality requirement in that context was trustworthiness.

Ozkaya [94] pointed out that all the knowledge and experience in designing and reasoning about software systems does not immediately apply to AI-system engineering. The author suggested security, usability, privacy, explainability, data quality, and quantity, testability, and robustness as the critical attributes in the successfully designed structure and behavior of AI-enabled systems.

Zhang et al. [95] provides a comprehensive survey of techniques for testing machine learning systems. Authors defined quality attributes as testing properties, which included correctness, memory and energy efficiency, robustness, and others.

Truong [96] suggested applying the author’s R3E approach to evaluate the state of end-to-end ML systems. The R3E approach consists of robustness, reliability, resilience, and elasticity.

Kuwajima et al. [97], [28] state that all quality attributes from the standard SQuaRE model could and should be applied to the development of ML-enabled software with additional quality attributes from ethics guidelines for trustworthy AI from the European Commission.

Horkoff [98] summarized a selection of quality attributes presented previously in the work of Habibullah et al. [81] and created another quality model consisted of eight general quality attributes: accuracy, performance, fairness, transparency, security, privacy, testability, and reliability.

Various quality models have been proposed by other authors: particularly for AI-chatbots [99], ML-based systems for Automotive OEM [100], Deep

Learning Systems [101], IoT systems [102], Regression-Based ML-systems [103], and other types of AI-based systems [104], [53], [105], [106], [107], [108], [109].

**Studies based on the other methodologies.** Some papers stem from methodologies, for example, experience reports from particular companies, design science research of particular solutions, applications of common standards to specific cases, or studies of community trends.

Based on the priorities of a particular company in ML-enabled systems development, Cysneiros et al. [110] identified several key quality attributes: trust (a.k.a. trustworthiness), ethics, and transparency.

Washizaki et al. [22] collected “good/bad” software engineering design patterns for ML techniques to provide developers with a comprehensive classification of such patterns. Their patterns are implemented to directly affect quality attributes, such as performance, reliability, accuracy, and others.

Ahmad et al. [111] noticed that industry practices use tools that do not enforce requirements engineering for AI and that there are gaps between research and practices in RE for AI. They conclude that the engineering of AI-systems introduced new specs that did not exist in traditional software, which include data quality, data quantity, accuracy, and explainability.

Felderer et al. [112], [113] brought together best practices written by software engineers and data scientists. Key quality attributes according to the studies above were: data quality, system accuracy, correctness, interpretability, etc.

Arseniev et al. [114] applied fundamental software engineering principles to AI systems. They analyzed how various software teams build software applications with customer-focused AI features and which main problems they meet. The authors claimed that a substantial amount of effort is usually spent on data collection and data preparation. Data quality characteristics also reflect the quality of the AI system. In addition to data quality and quantity, the authors worked with the reliability, scalability, and convenience of accompaniment (in the context of the research equals maintainability).

Other practical-oriented solutions described in the scientific literature were a bug benchmark [115] with key affected attributes of testability, traceability and functional suitability, quality assessment and criteria analysis for AI image recognition software [116] with an emphasis on data quality and robustness, compositional approach to creating architectural frameworks for distributed AI systems [65], explaining models in AI [117], ontology-based modeling and analysis of trustworthiness [118], ensuring dataset quality [119] and other works that operated with quality attributes [120], [121].

**Synthesized quality model.** A contextual cut-off line between “frequently mentioned” and “infrequently mentioned” attributes was drawn based on the number of their mentions in the scientific literature according to the rule: “If the number of occurrences was greater than 4 then the quality attribute was recognized as frequently mentioned, otherwise, as infrequently mentioned”.

The next step was to combine semantically similar frequently mentioned attributes into common quality attributes (CQAs).

Some authors used the term “performance”, which implied “system accuracy” or “resource efficiency” depending on the context and to avoid misunderstand-

ings, we divided this term into the above two groups during the data extraction process.

We noticed that the papers that mentioned the quality attribute of “efficiency” used it to describe four different cases: efficiency in terms of running time, efficiency in terms of memory costs, efficiency in terms of energy consumption, or accuracy of system output. The first three cases were exceptionally considered as subcharacteristics of resource efficiency, while the last one was also included as the “system accuracy”.

A review of the literature showed that “correctness” and “functional suitability” were often used as contextual synonyms; “usability” was a separate attribute from any other; terms “trustworthiness”, “reliability”, “safety”, “robustness” and “scalability” were of the same nature; “privacy” was presented as a subset of “security”, “maintainability” consisted of “system transparency”, “testability” and “maintainability” itself; “portability” and “adaptability” were the attributes of the same nature; “explainability” and “interpretability” described by highly-related spectrum of issues, “fairness” and “ethics” were used as synonyms with the rare exceptions of non-standard terminology when “fairness” characterized the “trustworthiness” of model’s predictions; “data quantity” was often considered as a special characteristic of the overall “data quality”.

The result of the semantic unification of frequently mentioned attributes into common quality attributes and their sub-characteristics is presented in Figure 4.1.

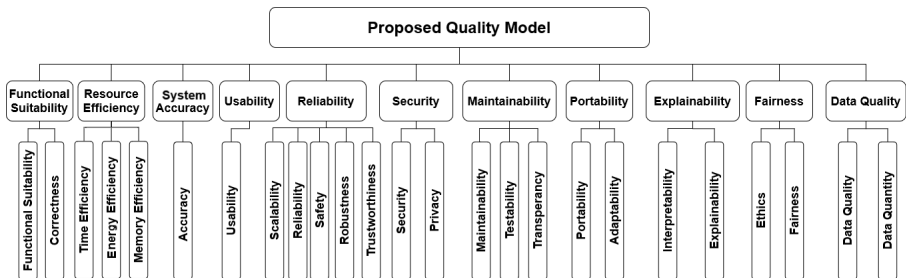


Figure 4.1: Proposed Quality Model for ML-enabled Systems

Our quality model comprises the following high-level common quality attributes: *Functional suitability* is the degree to which a system corresponds to functional requirements. *Resource efficiency* is the degree to which a system fulfills a given functionality within an existing amount of hardware capacities. *System accuracy* is the degree to which a system performs and analyzes the contextual environment and refers to the performance of the entire system in real-world conditions, including model inference and data post-processing. Particularly, system accuracy is different from model accuracy, which specifically measures the predictive performance of the trained machine learning model on a given dataset. *Usability* is the degree to which a system can be employed by end users to achieve specified goals. *Reliability* is the degree to which a system performs specified functions under specified conditions in the fixed

domain. *Security* is the degree to which a system protects information and data. *Maintainability* is the degree to which a system can be modified and supported by developers and maintainers to achieve specified goals. *Portability* is the degree of effectiveness with which a system can be transferred from one domain, software, or hardware basis to another. *Explainability* is the degree to which the behavior of a system (primarily, the behavior of ML models) and its output can be explained by humans. *Fairness* is the degree to which a system can detect and prevent an algorithmic bias created by a model. *Data quality* is the degree of integrity and sufficiency of data for model training, testing, and validation, including the reliability of the related data sources.

**Verification of the quality model.** The developed quality model was presented to six experts at the Swedish Requirements Engineering Meeting (SiREN 2023) in Gothenburg, Sweden organized by Chalmers University of Technology. Four experts out of six participants rated the model as fully complete, general, and relevant. One participant pointed out that the proposed model lacks the “compatibility” attribute (with reference to ISO 25010:2011 [71]). According to the used methodology, this attribute was rarely reported in the literature (only 2 times). This fact did not allow us to include it in the final model. This fact was considered as a threat to validity in Section 5. The last expert noted that the selection of terms for quality characteristics is not as important as specific metrics and ways to achieve them, however, they fully supported the proposed model in terms of completeness, generalizability, and relevance.

Further, the model was presented to four ML engineers from Swedish AI software companies. They conducted a theoretical comparison of the quality attributes we found with the system qualities considered by them when designing real AI solutions. Three of the practitioners concluded that the resulting model fully exhausts the quality of all the systems developed in their company and is relevant nowadays. The last expert noted the importance of compliance of the developed solutions with all the quality attributes we identified, as well as with the business goals of stakeholders. According to our findings, meeting business requirements can be expressed in both functional and non-functional requirements. Each NFR can be ranked according to the identified quality attributes, while strict adherence to the functional requirements corresponds to the identified “functional suitability” attribute. After clarifying the context and terminology, the expert agreed that the proposed model was complete, general, and relevant.

## 4.2 RQ2: Architectural Tactics

In Table 4.2 we present our findings under RQ2 and provide a correspondence in the format of “quality attribute - architectural tactic(s)”. We highlight that the table presents only common quality attributes without sub-characteristics, but they were considered in the search strategy, the details of which can be found in the supplementary artifact<sup>1</sup>. Each AT is characterized by its scope

---

<sup>1</sup>Supplementary Artifact: <https://doi.org/10.6084/m9.figshare.25673643.v1>

as either training system (TS), deployed system (DS) or both. Note that the TS can generally be included as a subsystem in the DS (e.g., in systems that support retraining); in such cases, DS refers to the tactic being applicable to *other* subsystems than the TS.

We note that the “functional suitability” attribute was not included in the summary table for two reasons: this attribute is general in meaning (all ML-enabled systems must follow functional requirements), but not general in the ways of its achievement (for each system there is a unique set of functional requirements), and also because we were unable to find common architectural tactics to satisfy this attribute with the selected methodology.

Table 4.2: Architectural Tactics to Achieve Common Quality Attributes (CQA)

CQA	Tactic	Scope	CQA	Tactic	Scope
<b>Resource Efficiency</b>	Distributed Learning	TS	<b>Data Quality</b>	Data Preprocessing	TS/DS
	Federated Learning	TS		Data Profiling	TS
	Data Reduction	TS/DS			
<b>System Accuracy</b>	Hyperparameter Tuning	TS	<b>Explainability</b>	LIME	TS/DS
	Algorithm Selection	DS		Rule-based Models	TS/DS
<b>Usability</b>	HitL	TS/DS	<b>Maintainability</b>	Componentization	TS/DS
				Containerization	DS
<b>Security</b>	Intrusion Detection	TS/DS	<b>Fairness</b>	Bias Mitigation	DS
	Data Encryption	TS/DS			
	Federated Learning	TS			
<b>Reliability</b>	Data Versioning	TS/DS	<b>Portability</b>	Containerization	DS
	Human-in-the-Loop	TS/DS		Componentization	TS/DS

TS = Training System; DS = Deployed System

Below we provide detailed explanations of all the tactics found. To introduce a basic understanding of how each tactic can be implemented, we supplement all the tactics with one example of its implementation from the literature.

**Tactics associated with Resource Efficiency.** The ATs to increase resource efficiency were aimed at distributing and reducing resource-intensive processes. The first architectural tactic we found was an approach to distribute the powers for model processes among several computers as known as *Distributed Learning*. Shi et al. [122] described distributed deep learning as a time and resource-efficient approach to designing the machine learning process. Considering the context in which this tactic is useful, Rao et al. [123] based on the experiments concluded that distributed learning is effective when there is a need to allocate training loads evenly. The implementation of distributed logic itself (building extra connections) does not sufficiently affect the overall resources consumed. Other papers mentioned that traditional centralized architectures are time-consuming in the domains of biomedical images [124], photonic nanostructures [125] processing and could be replaced with the distribution of loads.

For example, Rao et al. [123] developed a distributed learning mechanism that enables self-adaptive resource provisioning by treating each virtual machine as a highly autonomous agent that submits resource requests based on its benefit.

The next explored tactic was *Federated Learning (FL)*. In contrast with

the centralized approach, FL shifts the computational load to the user equipment. While distributed learning involves training models collaboratively across decentralized nodes with mostly shared data, federated learning specifically focuses on training models mostly on local data. Considering application contexts, FL can be profitable when there is a need to lighten the load on the server [126] during model training.

For example, Brisimi et al. [127] developed a federated learning framework that can learn predictive models through peer-to-peer collaboration without raw data exchanges solving a binary supervised classification problem to predict hospitalizations based on clients' data.

Finally, we found the tactic of *automated data reduction* relevant for resource efficiency. This tactic can be used in the ML pipeline to reduce training data [128] or to reduce large amounts of input data in real-time when the system is deployed and operates [129]. Considering application context, this tactic is useful in systems with massive amounts of data, such as those in the Internet of Things (IoT) [128] domain. The main perspectives of big data reduction are noise-cleaning and addressing the “curse of dimensionality” caused by millions of variables in big datasets [129]. In terms of limited resources, this tactic can be the only way to run the system [130].

For example, Rehman et al. [129] collected different methods of automated data reduction in the form of big data compression algorithms, dimension reduction methods, and redundancy elimination.

**Tactic associated with Usability.** With our search strategy we could identify only one AT for increasing the usability of ML-enabled systems, which was the integration of a human as a system component, so-called “*human-in-the-loop*” (*HitL*). Petrelli et al. [131] stated that usability indicators of AI-based systems were sufficiently improved after the integration of so-called “interaction designers” into the processes. They acted as experts to coordinate reciprocal understanding. Winter et al. [132] and Kröll et al. [133] viewed close interaction and co-integration between users and the model as mutually beneficial. The user becomes more proficient in using machine learning for their tasks and gets a clearer picture of basic AI principles, while their feedback can serve as a basis for improving the system in terms of usability as well as fairness, explainability, and data quality. According to Heimerl et al., [134] integration of experts in the training system (when the experts manually evaluate and update training datasets) is as beneficial in terms of usability as their integration in the deployed system (when they evaluate system outputs). Sperrle et al. [135] proposed a human-centered approach to the evaluation of ML-enabled systems and highlighted the necessity of HitL to improve system usability. Considering the application context, the tactic is especially relevant for accessibility-critical systems [131] or systems primarily oriented toward the end users [134].

For example, Gomez et al. [136] noticed that the ML-enabled system in a specific case overlooked human factors (such as human workload or timing) which were critical for end-users. To address this issue, the model was replaced with an adaptive one to consider parameters provided by specific users.

**Tactics associated with Reliability.** To address system reliability or its sub-characteristics we found an AT of *Data Versioning*. Lewis et al. [12] conducted

a complex study on architectural challenges in ML-enabled systems and among other fundamental conclusions, stated that the technique of data versioning can be effective in improving reliability and robustness that serves as a safety net in case of unexpected failures during data processing. This aspect covers mostly the training system. However, in addition to versioning of datasets, it is also important for architects to consciously version related artifacts, such as parameters for model training, data for model evaluation, and evaluation results (“co-versioning”) [137]. Warnett et al. [64] proposed the versioning of output data when the system is operating to ensure the traceability and integrity of results, allowing for the accurate tracking of changes in outputs over time. Considering the application context, such a tactic is relevant for all systems where multiple versions of a model could be deployed, either over time or even in parallel [138].

For example, Van et al. [137] used data versioning in the format of saving and storing different versions of end-to-end machine learning pipelines (including datasets for data processing pipelines and model coefficients for model training pipelines) to ensure that multiple versions of a pipeline can run in parallel.

Another tactic to address reliability and safety concerns was previously introduced *Human-in-the-Loop (HitL)*. Rajendran et al. [138] stated: “The involvement of humans during the training phase can play a crucial role in mitigating some safety issues of autonomous systems”, although it can also lead to extra expenses for the vendor. Considering application contexts, integrating expert users is reasonable to validate solutions that need to be available at sufficient capacity [138].

For example, Rajendran et al. [138] explored different integrations of experts as components to improve the reliability of deep learning autonomous systems such as “learning from demonstration”, “learning from intervention”, and “learning from evaluation” to deal with unforeseen circumstances and define safer policies.

**Tactics associated with Security.** The most frequently considered AT for improving security was the integration of *Intrusion Detection*. Sanju [139] claimed: “The protection of IoT systems from attacks and the assurance of their security posture is ensured by intrusion detection systems”. Liu et al. [140], Qu et al. [141], Laqtib et al. [142], Rashid et al. [143], Balbali et al. [144] listed several fundamental benefits of using intrusion detection as microservices in the architecture of industrial IoT for enhancing security. Intrusion detection is mainly used for preventing data poisoning (where exclusively the training system is under attack)[143] and adversarial attacks (where minor malicious changes in input data can cause the model to make incorrect predictions) [141]. In comparison with traditional software, intrusion detection in ML-enabled systems is more flexible and adapted to changes due to possible different outputs produced by the model over time or new behavior of the model caused by retraining. Considering the application context, the tactic is useful for systems operating with large amounts of real-time input data or big datasets consumed by training system [142], including IoT systems [139].

For example, Sanju et al. [139] introduced a hybrid metaheuristics-deep learning approach with an ensemble of recurrent neural networks to detect and

prevent intrusions in real-time data processing in an operating IoT system.

Another identified AT was *Automated Data Encryption*. McGraw et al. [145], Bekri et al. [146] when analyzing risks for the security of ML-enabled systems, highlighted the important role of encryption of training and testing datasets to protect from several threats primarily. Wu et al. [147] found big data encryption efficient for system security, however, some encryption methods may not be optimal also for privacy by default. Kantarcioglu et al. [148] along with several non-architectural decisions, found data encryption as one of the ways to satisfy security requirements in industrial solutions when it comes to ensuring the security of all data flows within a deployed system. In addition to the encryption of datasets and input data, the encryption of the model parameters and weights should be conducted. Considering the application context, the implementation of this tactic is especially relevant for highly sensitive systems that operate continuously [146], including privacy-preserving deep learning systems [149].

For example, Phong et al. [149] proposed additively homomorphic encryption to increase the privacy and security of neural networks by allowing computations to be performed on encrypted data without decrypting it, thus protecting sensitive information throughout the processing stages.

Finally, an effective tactic to architecturally increase privacy and security is an implementation of *Federated Learning (FL)*. In addition to the benefits of this tactic for resource efficiency, it is commonly used to “train a massive amount of data privately due to its decentralized structure” [150]. Zhou et al. [151] proved the statement above: “The emerging federated learning (FL) offers a feasible solution for the privacy preservation of users’ sensitive data in training AI models”. In other words, federated learning allows the benefits of data privacy without the need for data to be shared with a central server [152], [153]. Considering the application context, this tactic is useful for privacy-critical systems [152], [153], including personalized big data systems [151].

For example, Zhou et al. [151] implemented a federated learning algorithm that ensures that sensitive data is not disclosed during model training together with a user-level personalized differential privacy mechanism.

**Tactics associated with Maintainability.** For both traditional and ML-enabled systems, the architectural tactic of *Containerization* has shown its effectiveness in terms of maintainability and system transparency. Rovnyagin et al. [154] explicitly claimed the positive effect of containerization along with related tools (such as docker or orchestrator) on system maintainability and operability. Kolltveit et al. [155] stated: “Models packaged in containers are simply run directly as standalone services” which contributes to the enhancement of maintainability. According to Openja et al. [156]: “Docker (which is a containerization service) allows for convenient deployment of websites, databases, applications’ APIs, and machine learning (ML) models with a few lines of code”. Finally, containerization allows applied scientists without advanced knowledge to deploy models and access High-Performance Computing (HPC) [157]. Considering the application context, this tactic is useful for systems that require more isolated dependencies and simplified updates [154]–[156]

For example, Openja et al. [156] identified 21 major categories representing

the purposes of existing ML projects using Docker, including those specific to ML models, which in turn reduces the complexity of managing ML models.

Another efficient tactic for enhancement of system maintainability was *Componentization*, which obviously contributes to more transparent testing [158]. The componentization can be applied to the overall deployed system architecture, or exclusively to the training system, or even exclusively to a model. Singravel et al. [159] stated that "Component-Based Machine Learning (CBML) enhances the capabilities of the monolithic ML models" in terms of transparency. Considering the application context, this tactic is relevant for systems that require constant monitoring and updates from the side of developers or maintainers.

For example, Singravel et al. [159] transformed the monolithic ML model in the domain of space exploration into a set of connected components which simplified its further maintenance.

**Tactics associated with Portability.** According to the literature review, both tactics associated with maintainability were also profitable for portability.

*Componentization* is a relevant architectural tactic for increasing portability [160]. Shadab et al. [161] reported that the development of AI logic in the format of reusable components could be an adequate solution to increase portability, however, it also may introduce new risks since "currently there is no framework that guides the selection of necessary information to operate in a system different than the one for which the component was originally purposed". Geyer et al. [162] concluded that components instead of one monolithic model extend reusability and generalization, which in the context of our research directly contributes to the quality of portability. In terms of portability, both componentization of the overall architecture as well as dividing the ML model into components are profitable. Considering the application context, this tactic is useful for cross-platform compatible systems that are partly or fully transferred from one software or hardware base to another [160] as well as to systems that are transferred from one environment to another [162].

For example, Geyer et al. [162] replaced monolithic models with a component-based approach that develops machine learning models not only for the parameterized design of the whole buildings in the construction domain but also for the design of its semi-independent parts on the lower level of abstraction.

The tactic of *Containerization* also improves the portability of ML-enabled systems by encapsulating all dependencies and configurations [163]. Considering the application context, this tactic is useful for cross-platform compatible systems, including cloud-based services [164].

For example, Naydenov et al. [163] studied different container technologies used in ML-enabled systems, such as K8s, K3s, Docker, Rancher, and others allowing the systems to run consistently across different platforms, such as local machines or cloud servers.

**Tactics associated with Explainability.** *Local Interpretable Models (LIME)* are a tool for solving issues of explainability (XAI) and interpretability [165], [166], by approximating the behavior of a complex underlying model around a specific prediction using a simpler local model that can *explain* the prediction. The intended application context of LIME is a system with a "black-box" model

that is inherently non-interpretable, such as a neural network obtained by deep learning [167]. Such complex models cannot be avoided in domains that deal with particularly complex phenomena, such as weather forecasting [168] and intrusion detection [169], where LIME is particularly useful. LIME is model-agnostic, which means it can be applied to any ML model without knowing its internals as it only requires access to prediction probabilities. Integrating LIME into a training system can help explain how the model comes to conclusions before deployment and help with the selection of the final model to be used in production. It can be also used in an already deployed system to explain the behavior of the existing model [170].

For example, Saadatfar et al. [170] proposed a LIME algorithm that generates more focused data samples close to the decision boundary and simultaneously close to the original data point in comparison with different LIME implementations, such as BayLIME, SLIME, and LS-LIME.

Another widespread tactic found was the usage of *Rule-based Models* [171], [172]. While LIME explains individual predictions of complex models by fitting simpler models locally around specific instances, rule-based models directly encode prediction rules that are clear for a human [173]. The intended application context is one where that permits such rules to be formulated, which then inherently leads to explainability and interpretability, and can make rule-based models favorable over more complex ones, especially for non-complicated tasks [174]. Rule-based models can also be used for rule-based approximation and visualization [175]. This AT can be also used in both training and deployed systems [176].

For example, Rajapaksha et al. [176] developed a model-agnostic rule-based approach that obtains k-optimal association rules from a neighborhood of the instance to be explained.

**Tactics associated with System Accuracy.** *Automated Hyperparameter Tuning* can be especially profitable in increasing model(s) accuracy resulting in the enhancement of the overall system accuracy. It is well-known that the accuracy of machine learning models relies on hyperparameter tuning [177]. Daviran et al. [178] stated: "The predictive accuracy of models can significantly increase when the optimized hyperparameters are predefined and then adjusted to training procedure", which, in this tactic, is an automated process. Considering the application context, this tactic is especially valuable for systems with complex models or large datasets where manual tuning is ineffective, including deep learning systems [179].

For example, Ottoni et al. [179] proposed a framework for automated hyperparameter tuning and based on the experimental results proved that this tactic sufficiently improved different accuracy metrics of an image recognition deep learning system.

Another tactic for system accuracy is an *Automated Algorithm Selection*. Kerschke et al. [180] proposed their implementation of automated algorithm selection from a pre-defined set of algorithms and noted that the choice might be made not only to maximize the accuracy but also based on other contextual priorities. Pise et al. [181] listed several key factors that must be considered in a proper algorithm selection tactic. Such a tactic fundamentally improves

accuracy in healthcare and medical systems as well [182], [183]. Considering the application context, this tactic is generally useful for the systems in which high accuracy is particularly important [180], [181], [184], and more specifically in systems operating in constantly changing environments [184].

For example, Alissa et al. [184] proposed a technique for automated algorithm selection, applicable to certain optimization domains in which implicit sequential information is encapsulated in the data. Specifically, they trained two types of recurrent neural networks to predict a packing heuristic.

**Tactic associated with Fairness.** *Automated Bias Mitigation* is a common term for a set of algorithms developed to increase the fairness of the deployed ML-enabled system outputs. Lee et al. [185] conducted a review of so-called fairness toolkits with the analysis of their relevance in improving system outputs from the ethical perspective. Ferrara et al. [186] suggested that "building specific methods and development environments, other than automated validation tools, might help developers treat fairness throughout the software lifecycle". Other algorithms for automated bias detection and mitigation were proposed by Agarwal et al. [187], Castelnovo et al. [188] and Zhang et al. [189]. Considering the application context, this tactic is primarily useful for the systems operating with personal data and sensitive parameters [167], in particular those that are not inherently interpretable, such as deep learning systems [167].

For example, Maan et al. [167] proposed a method that evaluates the fairness of deep learning model behavior against sensitive attributes (i.e. age, race, gender, sex, and so on) to help mitigate biases without compromising much on accuracy.

**Tactics associated with Data Quality.** The tactic of *Automated Data Profiling* is considered an effective tool to increase data quality in the domain of ML-enabled systems [190]. Data profiling contributes to the training system allowing the identification of missing values and the detection of outliers and anomalies. Considering the application context, this tactic is generally useful for systems that deal with heterogenous and complex data and, therefore, require comprehensive evaluation to ensure sufficient data quality for training a high-quality model, including domains such as cybersecurity [191], digital twins [192], and healthcare systems [193].

For example, Pansara et al. [194] proposed to employ extra machine learning algorithms to automatically profile and cleanse master data for complex model training operating in the domain of environmental sustainability.

While data profiling implies examining the structure and the content of data to understand its features, *Automated Data Preprocessing* includes data cleaning and data transforming to prepare it for analysis. Gawhade et al. [195] and Ramkumar et al. [196] proposed computerized data preprocessing algorithms to primary process input data before it enters the ML model in the deployed system. Santos et al. [197] suggested another implementation of automated data preprocessing used for the preparation of training datasets in supervised machine learning. In terms of ML-enabled systems, data preprocessing includes data splitting (dividing data into training, testing, and validation datasets). Considering the application context, this tactic is necessary for all types of ML-enabled systems that are going to be trained on unprepared datasets [197]

or that operate with raw data on the input [196].

For example, Bilal et al. [198] proposed an automated pipeline for advanced data preprocessing steps of target discretization and sampling which are validated using RandomForest.

**Definitions of Architectural Tactics.** Below we present brief contextual descriptions of all architectural tactics found. These definitions were built based on the experience from literature and brought to the common format (relevant for all studied papers despite specifics and context).

*AT1: Distributed Learning* is an architectural approach to machine learning aimed at parallelizing computing powers among several computers [123].

*AT2: Automated Data Reduction* is an automated process aimed at minimizing the complexity and size of datasets while preserving their essential information (widespread in IoT) [128].

*AT3: Federated Learning* is an architectural approach to machine learning aimed at training on local heterogeneous datasets [126].

*AT4: Human-in-the-Loop (HitL)* is an architectural approach where a human (expert) is integrated into the ML-enabled system as a separate component aimed at monitoring and improving the system’s behavior [131].

*AT5: Automated Data Versioning* is an automated process aimed at the creation, tracking, and management of different versions or iterations of datasets used for model training, testing, and validation [64].

*AT6: Intrusion Detection* is a tactic for complex systems (primarily, IoT systems) aimed at the detection and classification of network intrusions and anomalies [139].

*AT7: Automated Data Encryption* is an automated process aimed at securing sensitive data used for training, inference, or model deployment to protect it from unauthorized access [145].

*AT8: Containerization* is an architectural approach aimed at packaging an entire system or model (incl. its dependencies and runtime environment), into a standardized unit called a container [154].

*AT9: Componentization* is an architectural approach aimed at breaking down a software system into modular components or building blocks that can be independently developed, tested, and deployed [160].

*AT10: Local Interpretable Models (LIME)* is an approach to machine learning aimed at explaining black boxes by approximating the behavior of a complex model around a specific prediction using simpler (more interpretable) models [165].

*AT11: Rule-based Models* is a type of model that relies on explicit rules (i.e. if-then) that are designed and specified by humans or domain knowledge to approximate complex model behavior [172].

*AT12: Automated Hyperparameter Tuning (or Hyperparameter Optimization)* is a method aimed at searching for the best hyperparameter values for the model based on certain criteria [179].

*AT13: Automated Algorithm Selection (or Algorithm Configuration)* is an automated process aimed at searching the most appropriate method(s) for a certain task or in certain circumstances [180].

*AT14: Automated Bias Mitigation* is an automated process aimed at identifying and reducing bias in algorithms, models, and datasets by their evaluation through fairness metrics or "sensitive" feature monitoring [186].

*AT15: Automated Data Preprocessing* is an automated process aimed at preparing raw data for analysis and model training [190].

*AT16: Automated Data Profiling* is an automated process aimed at analyzing and summarizing the characteristics of a dataset to gain insights into its structure, quality, and distribution [195].

**Verification of the architectural tactics.** All authors of this article were conducting constant peer-reviewing of the resulting list of ATs. During weekly meetings, architectural tactics were discussed against identified quality attributes based on the expertise of each co-author. During the validation, issues related to the architectural nature of the identified artifacts and the advisability of classifying them as tactics were discussed. In other words, based on the studied literature we checked if the artifacts affected the overall principle of architectural design (e.g., componentization) or could be integrated as constituent parts into the overall system architecture (e.g., automated bias mitigation module). In this paper, we presented a list of ATs agreed upon by all co-authors of this work.

Further, the list of ATs was shared with four ML engineers from Swedish AI software companies. One practitioner stated that he had experience with all of the suggested tactics to "a greater or lesser extent" except federated learning. He concluded that, based on his experience, all of the tactics presented were relevant to the quality attributes associated with them with an exception for federated learning. Due to insufficient expertise, the interviewee could not confirm or deny this connection. The second practitioner had a similar background and experience with all of the tactics listed except federated learning. He concluded that the list of tactics was accurate and consistent with quality attributes, however, he noted that the list was not complete. He proposed supplementing the list with the tactic of "code versioning" to improve reliability and maintainability. Using our methodology, we were unable to find this tactic relevant in the literature. However, from a practical point of view, we see the importance of this remark. It requires additional assessment and refinement of the search string. The other two experts confirmed their experience in employing all the listed ATs and found all of them relevant in the context of corresponding QAs. They provided several organizational decisions on how to improve resource efficiency and fairness (e.g., evolution of developing culture), however, they struggled to propose any additional ATs to this list.

To enhance the generalizability of verification results, it is preferable to continue validating the list of ATs by experts and practitioners. We anticipate that the list of architectural tactics will expand as we receive feedback from experts. We see great potential in updating the list of tactics and further exploring new entries.

### 4.3 RQ3: Trade-off analysis

Table 4.3 represents the summary of our findings obtained during the systematic literature review to answer RQ3.

Table 4.3: Trade-off analysis: impact of Architectural Tactics on Quality Attributes

QA	AT1	AT2	AT3	AT4	AT5	AT6	AT7	AT8	AT9	AT10	AT11	AT12	AT13	AT14	AT15	AT16
RE	+	+	+	0	0	-	-	-	0	+	0	-	-	-	a	0
U	+	0	0	+	0	0	0	0	0	0	0	0	0	0	+	+
R	0	0	a	+	+	+	+	+	-	0	+	+	+	a	-	+
S	a	0	a	a	0	+	+	-	0	0	0	+	0	0	+	+
M	0	0	0	+	+	0	0	+	+	+	0	0	+	0	+	a
P	+	0	0	0	0	0	0	+	+	0	0	+	0	+	0	0
E	-	0	0	+	0	-	0	0	+	+	+	0	-	0	+	0
SA	0	-	-	0	0	0	-	0	+	-	-	+	+	-	a	+
F	-	0	-	+	0	0	0	0	0	+	0	+	0	+	a	0
DQ	0	-	-	+	0	+	+	+	0	0	0	0	0	a	+	+

+ = predominantly positive impact, - = predominantly negative impact  
a = ambivalent impact, 0 = insufficient evidence either way

Resource Efficiency (RE), Usability (U), Reliability (R), Security (S), Maintainability (M), Portability (P), Explainability(E), System Accuracy(SA), Fairness (F), Data Quality (DQ)

Below, we provide an analysis of the papers that investigate quality trade-offs of the identified architectural tactics.

**AT1: Distributed Learning.** Distributed learning can have positive side-effects on system usability by enabling learning across multiple nodes, thereby enhancing responsiveness and adaptability to diverse user needs [199]. However, the impact of distributed learning on privacy (in the current context: on security as well) is controversial. On the one hand, due to their distributed nature such systems are more stable in terms of security since they do not rely only on one server [200] and they can be profitable to preserve privacy due to the essence of decentralized nodes without a necessity to share sensitive information centrally [201]. On the other hand, issues with data confidentiality, security breaches, and potential misuse of personal information are connected to increased exposure of sensitive data across those decentralized nodes [202], [203]. The positive impact of distributed learning on portability is proved by its ability to transfer and deploy trained models across different computing environments and devices [199]. The negative impact of distributed training on explainability arises from the difficulty of tracking and understanding how individual augmented data from different nodes influence the final results of the model [204]. Finally, representation across decentralized nodes can lead to biased model outputs and unequal treatment of different demographic classes [205].

**AT2: Automated Data Reduction.** In addition to the obvious improvement in reducing the load on system resources, automated data reduction tactics also have some limitations. Any interventions in datasets can be risky, especially for complex (low-explainable) models. First of all, such a tactic may decrease system accuracy due to the potential loss of important data during the reduction process [206]. The risks of incorrect perception of data by the algorithm and

classification of useful data as noise or outlier is an obvious risk for data quality and quantity when implementing this tactic [207], [208].

**AT3: Federated Learning.** In RQ2 we identified that federated learning is often used for increasing security and privacy particularly, however, some papers found for RQ3 by Shen et al. [209], Jeong et al. [210], Jagarlamudi et al. [211] and Shin et al. [212] point to the practical insecurity of existing implementations of federated learning and noted severe vulnerabilities associated with data leakage or inference attacks during the decentralized model training across multiple devices: "existing federated learning protocol designs have been shown to be vulnerable to adversaries within or outside of the system, compromising data privacy" [213]. Therefore, based on the development level of federated learning at the time of writing the current paper, its impact on security and privacy is recognized as controversial. The reliability of federated learning systems can be considered ambivalent. On the one hand, "federated learning resulted in a reliable strategy for model development" [214] due to its capability to incorporate diverse and decentralized data sources. On the other hand, potential communication bottlenecks and data heterogeneity across devices lead to severe challenges in terms of robustness [213], [215], [216]. Some risks of federated learning are also connected to system accuracy due to the aggregation of diverse and potentially noisy local data from distributed devices and fairness due to insufficient diversity of data collected [217]. Such challenges also affect overall system data quality.

**AT4: Human-in-the-Loop (HitL).** The effect of HitL on security and privacy is controversial. On the one hand, integration of human intelligence as a system component can bring the benefit in guiding the XAI-enabled system and generate refined solutions in terms of vulnerability detection [218], [219], and on the other hand, "the involvement of humans results in an external and unpredictable element that increases security concerns" [220]. Human-in-the-Loop is a unique element that plays a crucial role in the human-centered system qualities such as maintainability by intelligently tracking changes and intermediate results over time [221], explainability by leveraging bidirectional symbiotic sensing feedback [222], [223], [224] and fairness by identifying sensitive data and parameters [225], [226], [227]. Finally, data quality can be significantly improved based on the feedback constantly provided by analysts and engineers [228].

**AT5: Automated Data Versioning.** Automated data versioning can enhance the maintainability of ML-enabled systems by ensuring reproducibility and traceability of model training and inference, which simplify debugging and model updates [229], [230].

**AT6: Intrusion Detection.** Intrusion detection in IoT systems can negatively affect resource efficiency due to the high computational and memory requirements of deep neural networks [231], [232]. While "an intrusion detection system is a promising automotive security enhancement", it also improves anomaly detection capabilities, thereby improving overall system robustness by reducing the risk of non-security-related failures and errors [233]. The inherent complexity of deep neural networks for intrusion detection negatively affects the explainability of the overall often low-explainable IoT systems [234],

[235]. Finally, identifying and removing unnecessary data from the datasets significantly contribute to the data quality on a system level [144], [163].

**AT7: Automated Data Encryption.** Data encryption in ML-enabled systems can negatively impact resource efficiency by increasing computational overhead and latency due to the additional processing requirements for encryption and decryption [236], [237]. The positive side-effect of data encryption in terms of reliability appears due to ensuring data integrity and reducing the risk of data corruption [238]. Wang et al. [239] noted a slight decrease in the system accuracy of the encrypted model in comparison with non-encrypted solution. Any data protection tactic also makes a significant contribution to overall data quality [240].

**AT8: Containerization.** The main challenge of containerization in terms of resource efficiency arises from the potential resource overhead of container orchestration and virtualization [241], [156]. However, such a tactic has a positive side effect on system reliability by providing a consistent and isolated runtime environment [241]. Figueroa et al. [242] claimed: "Combined with IoT, containerization allows efficient allocation, fast execution, and deployment of hardware resources". According to Joraviya et al. [243]: "Containerization has introduced new security challenges including cloud data breaches in ML-enabled systems", which is also accompanied by increased attack surface and potential misconfigurations including increasing risks of data breaches, model theft, and adversarial attacks due to shared resources, image vulnerabilities, and insufficient isolation, making strict access control and monitoring essential. Finally, it has a positive effect on data quality due to its ability to facilitate consistent data handling and processing environments [244].

**AT9: Componentization.** With our search strategy we could not find any evidence that componentization has any crucial impact on resource efficiency. However, we found a positive impact of this tactic on the reliability of IoT systems [245] by enabling the implementation of safety-critical components with clear interfaces and well-defined behaviors. At the same time, this tactic is reasonable to isolate specific model components to improve explainability and interpretability [246]. Finally, based on the experimental results Heisele et al. [247] concluded that "the component system clearly outperformed global systems on all tests in terms of accuracy".

**AT10: Local Interpretable Models (LIME).** The work of Kumarakulasinghe et al. [248] significantly contributed to the analysis of trade-offs when applying local interpretable models. According to this study, LIME provides interpretable and simple local explanations without a need for resource-intensive global model explanations, which in some cases is really profitable for resource efficiency. This tactic of simplification also contributes to improvements in maintainability. However, it goes in contrast with reliability, where LIME brings the risks of potentially incorrect local explanations that do not accurately reflect the overall behavior. Mori et al. [249] also found this fact a reason for misleading interpretations and decisions (which is considered a negative impact on system accuracy). At the same time, LIME can be used to assess a classifier's fairness and to determine the sensitive features to remove [250].

**AT11: Rule-based Models.** The only impact of rule-based models found

with our search strategy was on system accuracy, which can suffer from limited adaptability to complex and dynamic data patterns when scenarios lie outside the predefined rules [251], [252], [253].

**AT12: Automated Hyperparameter Tuning.** When automated hyperparameter tuning (HPT) is aimed at increasing system accuracy, the trade-off with performance efficiency occurs most often [254], [255]. Liu et al. [256] claimed: “The current resource provisioning approaches for HPT are unable to adjust resources adaptively according to the upward trends of HPT accuracy at runtime. On the other hand, dynamic resource provisioning approaches based on checkpointing are inefficient for HPT, because of the high overhead of context switching and job restarting”. HPT can enhance reliability by optimizing model generalization, reducing the risk of overfitting [257], [258]. The positive impact of HPT is also noted in terms of security [259], [260] by improving resistance against adversarial attacks. Feroz et al. [261] claimed that HPT can improve not only system accuracy and system reliability but also the adaptability and portability of the system in different real-life scenarios. Finally, HPT can be used in the form of optimizing model parameters to reduce bias and consider different demographic groups or sensitive attributes [262], [263].

**AT13: Automated Algorithm Selection.** Automated algorithm selection like HPT often brings a trade-off between resource efficiency and system accuracy [264], [265]. The main possible benefit of the automated algorithm selection component lies in the recommendation of a promising learning algorithm based on meta features computed from a given dataset [266]. Such analysis can be too complicated and time-consuming for human data scientists and delegation of those responsibilities to ML is considered a valuable contribution to system maintainability. Also, it in some sense minimizes human factors in algorithm selection, which has a positive effect on reliability as well. However, existing automated algorithm selection methods for increasing accuracy rarely consider explainability as a factor for selection, which leads to the complexity of automatically chosen algorithms [267].

**AT14: Automated Bias Mitigation.** Hutiri et al. [268] found the risks of computational overhead due to the complexity of bias detection and correction algorithms in the context of IoT systems. The impact of this tactic on reliability is ambivalent. On the one hand, such methods improve system stability when working with diverse datasets, however, they can also lead to potential unintended model changes with risks of system failures [269], [270]. Increased adaptability of the ML-enabled system also influences the common attribute of portability [271]. The potential distortion or removal of relevant patterns in the data introduced by this tactic harms system accuracy [268], [272]. This fact also affects the attribute of data quality [273].

**AT15: Automated Data Preprocessing.** This tactic has a controversial impact on resource efficiency. According to Ramirez et al. [274]: on the one hand, the introduction of automated data preprocessing contributes to a faster and more precise learning process which can potentially save resources, on the other hand, when it comes to big data systems such tactic can lead to resource overload due to the large volumes of data being processed. Rendleman

et al. [275] proposed a method to increase the usability of a certain module when data preprocessing is conducted according to the priorities of end users. Automated preprocessing offers certain benefits in terms of model training, such as lowering the manual effort required for data preparation and enhancing maintainability by structuring and formatting data [276], while it can also protect models against malicious inputs and data poisoning [277], [278]. It also improves explainability by ensuring consistent and standardized data transformation, which makes model behavior and decision insights clearer to humans [279], [280]. The impact of this tactic on system accuracy is ambivalent since it can be either improved by standardizing input data and noise cleaning or harmed by potentially introduced biases or distortions by the algorithms [281], [282]. The complexity of automated data preprocessing in the context of human-centered learning can also have different implications for fairness due to the same reasons [281].

**AT16: Automated Data Profiling.** With our search strategy we were unable to find any evidence that automated data profiling has any crucial impact on resource efficiency as we expected. However, the personalization of certain data (which is a subset of data profiling) can significantly increase usability according to the needs of certain users [283], [284]. Data quality improvements provided by automated data profiling modules play a crucial role in overall system reliability [285]. In the context of IoT, data profiling can detect data vulnerabilities and privacy risks as an improvement of system security and upgrade feature understanding as an improvement of system accuracy [286]. Finally, the impact of data profiling on maintainability is controversial since it can reduce manual effort on data management, but brings the risk of over-reliance on automated processes, which can be “black boxes” if they are executed by complex ML-algorithms [287], [288].

**Verification of the trade-off matrix.** The resulting table of quality trade-offs was constantly being peer-reviewed by co-authors of this paper based on their independent expertise. This paper presents a version of the table agreed upon by all authors of the article.

All identified trade-offs are supported with literature references, however, more expert validation is desirable. Due to the large number of identified impacts, it would be complex and lead to significant effort. We present a strategy for such assessment in Section 5.

# Chapter 5

## Discussion

### 5.1 Observations regarding quality attributes.

This study proposed a common quality model for ML-enabled systems. This model took a broader look at ML-enabled specific nature and suggested considering attributes related to “data quality” and ML-unique “explainability”, “system accuracy” and “fairness” along with a standard set of attributes.

The attribute of system accuracy is a complex indicator that goes beyond model accuracy itself. In the context of quality, it is used to understand whether the system can operate effectively in the existing context with the existing accuracy (incl. metrics of precision, recall, F-score, etc.).

Thirteen papers referred to data quality as an attribute of the overall system quality. This attribute characterizes the quality of data sets used for model training and testing, as well as the ways this data was obtained and the sources from which this data was collected from at the system level. Working with unreliable or incomplete data causes a high risk of system failure due to its incorrect or insufficient perception of contextual reality as well as legal issues.

Quality attributes rarely addressed in the reviewed literature may still deserve further study. For any attribute named at least one time in RQ1, we can assume that it is relevant for some specific ML-enabled system(s). For example, the retrainability attribute can be very important for systems operating in especially dynamically updated environments, and the autonomy attribute could be relevant for systems that, for a number of reasons, need to be isolated from all external influences.

Remarkably, the standard quality attribute of compatibility is emphasized in the literature as relevant to the quality of ML-enabled systems only two times. For this reason, this quality attribute was not included in the proposed model. According to studied papers, we noticed that the considered works typically sought to study characteristics connected to external entities: resource efficiency as a result of interaction with available resources, usability as the result of interaction with end-users, maintainability - with developers and maintainers, system accuracy - with context, fairness - with society, portability - with new environments, etc. However, the interaction of ML-enabled systems with other

software systems (which is the basis for compatibility) remains poorly described in the scientific literature and likely requires additional attention.

## 5.2 Comparison to ISO standards.

In 2023, the International Organization for Standardization (ISO) issued a new standard ISO 25059:2023 [73]. This standard offers a quality model for AI systems, which can be seen as a possible alternative answer to RQ1. A comparison of our quality model with ISO 25059:2023, as well as with the traditional SQuaRE quality model (ISO 25010:2011 [71]), is presented in Table 5.1. The table maps all high-level attributes from three quality models grouped by semantic similarity. We now compare our obtained quality model to those from two relevant ISO standards. Importantly, our goal is not to present a new, competing standard, but to reflect the results of our literature review in comparison to existing quality models, in particular, those from the standards.

Factually, neither of the previous standards considers system accuracy and data quality on the system level, whereas we found in RQ1 that the literature frequently mentions them as system-level concerns and in RQ2 identified appropriate architectural tactics to address them. Moreover, ISO 25059:2023 considers the ethical perspective (related to fairness in the terminology of our research) as a high-level quality attribute and combines transparency with explainability. In our model, we decided to separate explainability and transparency. While system transparency refers to the openness and accessibility of a system’s internal processes (which is mostly important for developers and maintainers), system explainability focuses on how clearly the system’s decisions and processes can be understood and interpreted by humans (which is more important for users, operators, and experts). The release of ISO 25059:2023 confirmed the relevance of RQ1 and highlighted the need to consider ML-specifics in assessing the quality of software systems.

Table 5.1: Comparison of proposed quality model and ISO standards

<b>Proposed Quality Model</b>	<b>ISO 25059 (AI)</b>	<b>ISO 25010 (Trad. soft.)</b>
Functional Suitability	Functional Correctness	Functional Suitability
Resource Efficiency	-	Performance Efficiency
Usability	User Controlability	Usability
Reliability	Robustness	Reliability
Security	-	Security
Maintainability	Intervenability	Maintainability
Portability	Functional Adaptability	Portability
Explainability	Transparency	-
System Accuracy	-	-
Fairness	Ethical	-
Data Quality	-	-
-	-	Compatibility

## 5.3 Observations regarding trade-offs.

The analysis of trade-offs proposes a comprehensive mapping of different impacts after the implementation of certain ATs reported by different researchers. The most frequently reported trade-offs appeared between system accuracy and resource efficiency, system accuracy and explainability, fairness, and resource efficiency. Also, notable positive “side-effects” include the facts that: architectural enhancement of security can often increase data quality and reliability, enhancing system accuracy positively affects reliability, and tactics to improve data quality can have a positive impact on usability and security.

We met a remarkable situation with AT3 of Federated Learning. While investigating RQ2, we found its wide application to increase resource efficiency and security. However, during the work on RQ3, we identified several significant vulnerabilities in existing implementations of federated learning, which motivated us to characterize its impact on security as controversial.

Context analysis is critical when applying architectural tactics of distributed learning in terms of security, automated data preprocessing in terms of accuracy and fairness, and automated bias mitigation in terms of reliability and data quality.

## 5.4 Threats to validity.

The main threat to external validity is that the generalizability of our architectural findings can be limited as we restrict our analysis to available literature, specifically, scientific papers. While the analyzed literature stems from diverse domains and methodologies, including those that analyze practical experiences from companies, we cannot claim generalizability to all possible systems and sub-domains. A planned mitigation is to conduct a grey literature study investigating non-scientific literature such as blog entries and repository documentation. Another threat to external validity is caused by the complexity of the search process for RQ2. We searched for architectural tactics with corresponding keywords, however, it is possible that some architectural tactics were not referred to as such in the literature. To mitigate this we also included related terms (e.g., “design decisions”) in the search query. However, we still cannot state that the list of found ATs is complete and to mitigate this threat, we plan to conduct some more interview studies with practitioners and experts. Finally, the mapping of quality trade-offs for RQ3 was complex and should take sufficient resources for its validation from the side of practitioners. The possible mitigation is to follow the proposed verification strategy described further.

The main threat to internal validity associated with our approach to consider all quality attributes of equal importance as well as architectural tactics of equal value. Therefore, we do not weigh the magnitude of the trade-offs between quality attributes and the scale of the consequences of AT implementation. It can be mitigated by in-depth research of each AT with the study of the specific contexts, leaving generalizability behind.

The main threat to construct validity is our focus on commonly reported quality attributes in the literature, which we implicitly assume to be correlated

with their generalizability and the need to include these attributes in a common quality model for the domain of ML-enabled software. Clearly, some of the less commonly reported attributes might still be important in particular domains and use cases. To mitigate this threat it is possible to run a separate study of such attributes.

## 5.5 Implications for practitioners

Our study has several implications for practitioners and researchers. Considering practitioners, our findings can be used in a checklist-like manner during the system requirements and design stages. During requirements elicitation, our quality model can guide practitioners in identifying relevant non-functional requirements for the overall system that then need to be addressed by the system architecture. The list of architectural tactics provides them with insights into how to achieve those quality attributes architecturally. We especially highlight our description of context and examples we provide for each tactic, which allows practitioners to match the tactics to their particular context at hand. Finally, the table of trade-offs raises awareness of possible unintended consequences of decisions made. These insights can be especially valuable for start-ups and SMEs with limited resources for hiring ML engineering experts.

Our findings are also informative for the emerging area of *MLOps*, which focuses on the integration of DevOps principles and practices into the development and maintenance of machine learning systems [52]. While a dedicated literature study of quality attributes and tactics for MLOps is outside our scope, we identify the following potential applications of our findings in an MLOps context. First, the identified quality attributes can help align the engineering process with business goals and operational needs and contribute to more sustainable software development. For example, the consideration of maintainability at the design stage can guide the optimization of planned resources for further maintenance, support, and updates of the deployed system [276]. Second, treating training and deployed systems as aspects of a single complex software architecture along with appropriate tactics during the design phase, can improve operations associated with system retraining [85]. Finally, recognizing trade-offs can help allocate resources and prioritize tasks within the common workflow [289]. For example, if a team uses containerization to improve maintainability, our findings emphasize potential drawbacks for security, which need to be addressed specifically within the roles and responsibilities of the DevOps setting, for example, by actively planning and estimating the effort arising for a security team within the company [290].

## 5.6 Implications for researchers

We suggest several directions for future work within the existing architectural perspective. First, our list of trade-offs can be informative for researchers to develop new architectural tactics. Given that our numbers of identified tactics per quality attribute are between one and three, there is a potential for new tactics that complement the existing ones to find a different “sweet spot” within the trade-offs, possibly addressing specific domains and application

contexts. Second, more generally, while our study was broadly focused on ML-based systems, it would be worthwhile to conduct additional studies that explore the relevance and applicability of our findings in different domains (e.g., automotive vehicles, healthcare systems, etc.). Third, our study of trade-offs is based on literature sources; yet, the results could benefit from follow-up research that verifies and refines the resulting table of trade-offs based on insights in industrial settings.

While we conducted a first evaluation of our results with experts, there is room for further evaluating our findings in complementary ways, focusing on their real-world applicability in specific contexts. As part of future work, we propose to conduct such evaluations with empirical methods, such as case studies, controlled experiments, and surveys. As a promising direction, we highlight *action research* [291], which is dedicated to deploying solutions in a specific real-world context—for example, in our case, specific quality attributes and tactics in an industrial MLOps environment with multiple teams. In such an environment, we aim to investigate if the introduction of a quality model helps teams to allocate their priorities more efficiently, if the identified architectural tactics can be applied to systems of different natures and sizes and be generalizable to new contexts, and how identified trade-offs affect the decisions made by MLOps teams. Such an evaluation may shift the focus from our current architectural perspective to a more operational one, which can be especially relevant in the context of MLOps.

# Chapter 6

## Conclusion

This work contributes to the methodology of building software architecture for ML-enabled systems from the perspective of quality, offering ways to define it and achieve it through common architectural tactics with a consideration of possible side effects. Our focus is on theory-building, as we systematically identified and synthesized information from 206 research papers.

There are several worthwhile directions for future work. First, while our contributions are informed and validated by empirical insights from published literature, additional validation, for example, through expert feedback, is possible. Second, the selection of literature for analysis can be expanded by including gray literature. As a result, the quality model and proposed architectural tactics can be refined. Third, one can conduct complementary forms of evaluation highlighting the applicability of our findings in specific real-world contexts, through action research and other empirical methods. The emerging area of MLOps provides a particularly promising avenue for such evaluations. Finally, we suggest follow-up research to further investigate the role of quality aspects mentioned infrequently in the literature (e.g., portability) and to systematically study the impact of the identified tactics on all quality aspects.

**Quality Trade-Offs in ML-Enabled Systems: a  
Multiple-Case Study**

V. Indykov, R. Wohlrab, D. Strüber

Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing  
(SAC), 2025, pp. 1730 - 1737

# Abstract

When building a machine-learning-enabled system, quality objectives are achieved through architectural and non-architectural tactics, including general ones as well as specific ones that address machine learning specifics, such as the focus on data. However, implementing these tactics typically compromises other quality attributes that are not the primary focus of the tactic at hand. Previous research has investigated quality aspects and tactics for machine-learning-enabled systems, but there is a lack of detailed insights on quality trade-offs observed in industrial practice, and how companies address them. A study in this direction could especially help start-ups and SMEs to benefit from the insights of other companies, and academics to develop improved tactics addressing these trade-offs in alternative, potentially more effective ways.

In this paper, to fill this gap, we present a multiple-case study of four companies in the AI sphere. As AI solution providers, all companies are faced with a variety of quality priorities, tactics, and trade-offs in their addressed application domains. We find that our subject companies consistently address a common set of core quality priorities, encompassing *reliability*, *functional suitability*, and *resource efficiency*, which they address with recurring architectural tactics such as the use of *cloud-based components* for resource efficiency, and non-architectural ones such as *Scrum practices* for functionality suitability. Finally, we find a variety of trade-offs appearing in different companies with several recurring ones, two of them—*efficiency vs. reliability*, and *system accuracy vs. explainability*— manifesting themselves in three out of the four companies.

# Chapter 1

## Introduction

As machine learning (ML) becomes increasingly integrated into software systems across various industries, the demand for high-quality ML-based solutions continues to grow [26], [14]. However, the concept of quality is not universal; it varies significantly depending on the specific business goals. It depends on the domain of companies that own and use AI-based systems and on the strategic objectives of companies that deliver such systems. While the use of ML-enabled software in different domains is usually associated with a spectrum of business-oriented tasks, the development, integration, and maintenance of such systems create several complex challenges and dilemmas from a technical perspective.

Different quality priorities give rise to different approaches to achieving them by implementing architectural and non-architectural tactics [31], [30], [32]. Architectural tactics may involve decisions around system structure and component interaction, while non-architectural tactics can include workflow organization and process optimizations. Despite the efforts of companies to implement them most effectively, they usually encounter unavoidable trade-offs. Improving one quality attribute may necessitate compromises in another. For instance, developing complex models (i.e. deep learning models) usually results in high system accuracy, but also excessive computational resource consumption.

These trade-offs generate significant dilemmas for companies aiming to balance multiple, sometimes conflicting, quality objectives. As a result, there is no universal approach to developing high-quality ML-enabled systems and the consideration of context is required. This complexity sets a high entry threshold for ML and software engineering startups and small and medium-sized enterprises (SMEs) in the early stages, that are in dire need of best practices from larger successful players [292]. However, there is a lack of generally reported practical experience in quality-driven development of ML-based software, in particular, considering the inseparable connection between priorities (quality attributes), decisions to achieve them (tactics), and consequences of their implementation (trade-offs).

In this paper, to fill this gap, we present the results of a multiple-case

study with four companies that deliver AI-based solutions, in short, *AI solution providers*. AI solution providers are particularly interesting for studying quality trade-offs: On the one hand, they cater to the needs of their clients, whose particular application domains usually load to a specific view on quality priorities. For example, for ML-based healthcare systems, the focus is usually on privacy and system accuracy [293], whereas for autonomous vehicles, major attention is paid to reliability and safety [294]. On the other hand, as we observe in our study, AI solution providers align the priorities of their customers with their own strategic goals that can vary significantly (e.g., depending on their available resources [295], internal standards on security and privacy [296], ethical guidelines [297]). We report on the overall quality priorities, how they achieve them using tactics, what quality trade-offs they encounter, and how they balance these trade-offs.

Specifically, we address and answer three research questions:

**RQ1:** *Which system quality attributes are relevant for companies that provide AI solutions?* By answering this question, we identify the main quality priorities for certain companies.

**RQ2:** *What tactics are used to achieve those quality attributes?* To address this question, we share the main practical insights of how to achieve previously identified priorities.

**RQ3:** *What major quality trade-offs are encountered by practitioners?* By answering this question, we share quality dilemmas encountered in practice by our subject companies.

The synthesis of answers to all the RQs provides a focused collection of concentrated practical experiences, which can be applied by early-stage startups as best practices and used by academics to develop new solutions that address the observed trade-offs in alternative, possibly more effective ways.

## Chapter 2

# Background

A *quality attribute (QA)* is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders [77]. In the context of this research, we use the term “quality priorities” which are the most important quality attributes for specific companies, conditioned by strategic goals and external requirements.

An *architectural tactic (AT)* is a “*technique an architect can use to achieve the required quality attributes*” [77]. Architectural tactics affect the overall system architecture in one way or another [298]. We consider the tactics that do not directly affect system architectures as “non-architectural” in the context of the current study. This division allows us to establish a line between organizational and technical solutions, which can be especially useful in assessing resources for their adoption. By definition, the connection between tactics and certain *quality attributes* is implied [298].

A *quality trade-off* is a compromise between several quality attributes [289]. *Balancing trade-offs* ensures that appropriate levels of satisfaction are found for all involved quality attributes.

# Chapter 3

## Related Work

### 3.1 Quality Priorities

The study of software quality for ML-enabled systems is an in-demand topic among practitioners and researchers [26], [14]. Different common quality models were developed by independent researchers [27], [28]. In 2023, the International Organization for Standardization issued a standard quality model for AI systems [73] that considered the specific nature of machine learning. While such studies are crucial to understanding the overall quality picture, in practice, it is barely possible to fully achieve all of them simultaneously [299]. As a result, AI development companies introduce quality priorities depending on their goals, external requirements (provided by clients), and internal requirements (dictated by strategic goals) [300]. These are the most important attributes that primarily reflect the overall quality of their solutions, while other characteristics may be subject to partial sacrifice. The decisions in this regard vary among companies, therefore, it is necessary to primarily identify the main priorities of the considered companies to understand their context.

### 3.2 Architectural and Non-Architectural Tactics

There are several review papers on architectural issues in the context of AI-based systems [31], [30], [32]. These papers explore a collection of existing architectural design decisions to address certain challenges without a clear reference to system qualities or with a focus on individual quality attributes and their metrics in isolation from other characteristics. As a result, possible trade-offs often remain unnoticed. The same situation with non-architectural tactics that are based on certain MLOps principles [301], agile-implementations for the development of ML-based software [302] and other relevant organizational decisions [303]. Using such knowledge is certainly important in the early stages of startups, however, without consideration of possible trade-offs it can lead

to undesirable consequences. The research on industrial experience in this area can be beneficial and alert aspiring start-ups and SMEs about complex dilemmas that may arise.

### **3.3 Quality Trade-offs**

Existing research actively explores quality trade-offs arising from the implementation of selected tactics in ML-enabled systems. For instance, between energy efficiency and system accuracy [4], performance and interpretability [34], performance and privacy [35]. While such studies provide deep investigations of specific cases, they leave the identification of the most crucial trade-offs for industries aside and focus on one particular trade-off. Moreover, related work on trade-offs is often performed from a fundamental academic perspective [67], without a clear rooting in practice. When it comes to strategic prospects of startups and early-stage SMEs, the need for high-level studies with insights from the industry arises, which is the focus of this work.

# Chapter 4

## Methodology

To investigate quality trade-offs in ML-enabled systems in industry, we performed a multiple-case study. This method allowed us to gain a deep understanding of how companies handle quality objectives, implement tactics, and navigate trade-offs in practice, by focusing on several cases of interest. We selected our cases by focusing on a particular business segment of companies that provide AI-based solutions for other companies, in short, *AI solution providers*, as this would allow us to benefit from particular rich insights over different application domains. The data collection and analysis processes are described below.

### 4.1 Case Selection

The companies were selected based on the following criteria:

1. *AI Solution Providers*: Each company has to be involved in the development, deployment, integration, or maintenance of ML-enabled systems or ML components.
2. *Diverse Application Domains*: Each company has clients from different domains.
3. *Focus on High-Quality Solutions*: Each company has well-established processes for handling quality attributes like reliability, resource efficiency, maintainability, accuracy, and explainability in their systems.

An overview of our subject companies is provided in Table 4.1. All companies have their headquarters located in Sweden.

*C1*: This company develops customized ML-based systems for clients from different domains, including the healthcare sector, the financial sector, and the retail sector. The company provides solutions of different types, for instance, computer vision systems, natural language processing (NLP) systems, and ML-based business intelligence systems.

*C2*: This company integrates ML functionality in existing systems of clients from different domains, including the manufacturing sector, the real estate

Table 4.1: Subject companies

Company ID	Domain	ML Team Size
C1	Custom Development of ML-based Software	approx. 30
C2	Integration of ML Components	approx. 10
C3	AI-based Content Creation & Machine Translation	approx. 20
C4	ML-based Software Support and Maintenance	approx. 30

sector, and the governmental and public sectors. The company focuses on the integration of deep learning and large language models in workflows and data processing.

*C3*: This company develops AI-based content creation systems and shares them in the form of services. It delivers services to clients from different domains, including the financial sector, the manufacturing sector, and the retail sector. The company focuses on natural language processing and machine translation.

*C4*: This company develops and maintains customized ML-based systems for clients from different domains, including the construction sector, the manufacturing sector, and the public sector. The company provides different services for the development of ML-based control systems and embedded systems, however, the main focus is on the maintenance and support of existing ML-based systems, including AI model updates, proactive monitoring, and automated testing.

Our selected companies illustrate the wide-ranging application of AI and ML across various sectors, highlighting the importance of both development and maintenance for quality-driven solutions. Each company brings specific expertise and capabilities, enabling them to represent diverse needs and priorities.

## 4.2 Data Collection

Data collection was conducted through semi-structured interviews with key employees of each company. Participants were selected based on their role in the development, design, and maintenance of ML-based systems, ensuring that they had direct experience with the decision-making processes related to system quality. Our interviewees were software architects, ML engineers, project managers, and team leaders. Interviews with 6 participants were conducted (1-2 interviewees per company).

The semi-structured format maintains consistency in the core objectives by being structured around a set of predefined questions, while still providing flexibility in exploring aspects in detail using *ad hoc* follow-up questions.

This format is particularly suitable for discussions of artifacts that may be subject to different interpretations. For instance, architectural and non-architectural tactics can have varied levels of abstraction among different interviewees, as well as the same system qualities can be interpreted differently.

Hence, during the interviews, we paid not so much attention to the terminology used by the participants, but to the underlying phenomena they described.

For RQ1, interviewees were initially provided with full freedom to define their quality priorities, however, further, their responses were aligned in real-time with a predefined list of quality attributes to provide consistent terminology. For instance, we agreed with an interviewee to introduce the term “resource efficiency”, when they had previously reported concerns associated with computational and labor resources. According to all respondents, the introduced terminology accurately characterized their priorities. For RQ2, interviewees reported architectural tactics to address previously identified priorities in a free form. For RQ3, interviewees provided the list of the most crucial quality trade-offs they encountered in practice and some experience of how to balance them. We note that during their answers on RQ3, they also provided certain architectural and non-architectural tactics to balance trade-offs that were not mentioned previously for RQ2. We decided not to update the answer to RQ2, but to retain these experiences as part of the answer to RQ3 to maintain the consistency of the interviews.

During the planning and conducting of the interviews, Hove and Anda’s recommendations for semi-structured interviews in empirical software engineering research were followed [304]. Each interview lasted approximately 45 minutes. All interviews were recorded and transcribed for subsequent analysis. We make the interview questions publicly available in our Supplementary Artifact <sup>1</sup>, furthering reproducibility and providing a basis for further investigations. Due to privacy concerns, we omit to publish the full interview recordings and transcripts.

## 4.3 Data Extraction

The recordings of the interviews were transcribed with the help of the Otter.ai platform [305]. Manual verification of the transcribed interviews confirmed the high quality of transcription and full factual correspondence with everything said in the video recordings. The next step was a deep analysis of the phenomena discussed. As mentioned earlier, interviewees could operate with different terminologies but describe the same things. Clarifying questions and refining our hypotheses with participants during the interviews allowed us to identify which quality attributes, tactics, and trade-offs they described. Further, all of the extracted data was structured in the resulting tables associated with RQs. For RQ1, quality priorities were grouped by companies reported them; for RQ2, tactics were grouped by quality priorities they addressed and the companies reported them; for RQ3 companies were grouped by trade-offs they reported and augmented with brief descriptions of cases.

---

<sup>1</sup><https://doi.org/10.6084/m9.figshare.27074683.v1>

# Chapter 5

## Results

### 5.1 Quality Priorities

We now report the quality priorities provided by our subject companies, showing an overview in Table 5.1. All four companies reported functional suitability, resource efficiency, and reliability as their key quality priorities.

According to C1, their main quality priority is to follow the goals of their clients strictly. They conduct deep investigations of customer needs that further evolve into functional requirements. Compliance of the system with these requirements is called *functional suitability*. While those requirements can be dictated by the specifics of their clients, the company follows four more quality priorities that are not usually provided by them, which are universal for all their projects. The first one is *resource efficiency*. The company operates with limited labor and computational resources, which often creates a need to optimize certain processes and algorithms, re-balance existing powers, and offer feasible alternatives to unachievable solutions in terms of budget. They conduct constant workload monitoring and functioning capacity analysis. The second one is system *usability*. In addition to high client orientation, the company strives to deliver systems that are also efficient to the end users in terms of ergonomics and user interface optimization. The third priority is *maintainability*. Since the company also provides the services of post-production maintenance and technical support, it follows the strategy of “highly maintainable” solution development to avoid technical debts. It usually requires extra resources at the stage of design and development, however, it also brings serious resource savings strategically. Finally, *reliability* is one of the key priorities. Machine learning can never be completely stable due to evolving data, model updates, and changing environments, however, mitigation of this instability is one of their main objectives.

According to C2, the quality of the system is characterized primarily by the data it operates with. Crucial attention is usually paid to *data quality*. Improvement of data quality is especially resource-intensive when projects use dynamically updated data. This also highlights a pronounced role of *resource efficiency* monitoring. Another important factor is *system reliability*.

Table 5.1: Quality Priorities per Company

Company ID	Quality Priorities
C1	Functional Suitability, Usability, Reliability, Resource Efficiency, Maintainability.
C2	Functional Suitability, Reliability, Data Quality, Resource Efficiency, Fairness.
C3	Functional Suitability, Reliability, Resource Efficiency, Maintainability.
C4	Functional Suitability, Maintainability, Resource Efficiency, System Accuracy, Explainability, Reliability.

In the understanding of C2, reliability consists of two main sub-priorities: the availability of the system in changing conditions and the safety of model usage. Finally, C2 reports that the model and system must always take into account ethical considerations. This is a “*fairness*” priority.

C3 considered system *reliability* as the main quality priority. The main current focus of C3 is on AI-based content generation, perspectives of which are quite promising, however, clients stay wary and conservative. The low reliability of developed components may call into question the current possibilities of AI to generate high-quality insightful content and motivate potential clients to ignore this option and continue to create textual content manually. It in turn can affect the overall success of the company. Further, *resource efficiency* is also a key priority. Each project requires individual assessment of the applicability and expediency of ML integration. Finally, *maintainability* of the developed ML functionality is required to be on a high level since the company also takes on the responsibility for supporting delivered solutions.

The main business strategy of C4 is to support and maintain existing systems built by their company as well as by third-party developers. From this follows the key priority of high system *maintainability*. The main goal is to improve this attribute for all systems within their scope to save resources strategically. Another important attribute is *system accuracy*. The company has a lot of requests from clients to enhance the accuracy of system outputs to obtain certain objectives. Another wide range of requests is related to the improvement in predictability that is achieved through high *explainability* of the system’s behavior. Finally, *resource efficiency* and *system reliability* are also priorities that are constantly monitored and seriously influence the decisions made.

## 5.2 Tactics to Achieve Quality Priorities

To address their quality goals, our considered companies employ both architectural and non-architectural tactics. We now first present architectural tactics, before moving to non-architectural ones. Our considered companies

Table 5.2: Used Architectural Tactics

Quality Priorities	Companies Reported	Architectural Tactics
Resource Efficiency	C1, C2, C3, C4 C4 C4	Cloud-based Components Model Pruning Data Caching
Maintainability	C1, C4 C1, C4 C1 C1, C4	Microservices Containerization Experiment Tracking Dependency Tracking
Reliability	C1, C4 C1, C4 C1, C4 C1 C2, C3 C3 C4	Model Verification Module Code Versioning Data Versioning Pipeline Management Human-in-the-Loop Rule-based Models User Feedback Module
Data Quality	C2	Data Source Evaluation
Fairness	C2, C4 C4	Bias Mitigation Module Feature Engineering
Explainability	C4 C4	SHAP Explanations Local Interpretable Models
System Accuracy	C1 C1 C1, C4 C4	Data Preprocessing Feature Engineering Hyperparameter Tuning Data Postprocessing
Security	C1	Data Encryption

employ the following architectural tactics, as presented in Table 5.2.

In company C1, teams implement diverse architectural tactics depending on the project specifics, however, some tactics are used in almost all projects to follow the internal quality priorities. To improve resource efficiency, the company often resorts to *cloud-based tools and components*. It saves significant resources, however, it is only possible with the consent of the clients. For instance, these are third-party cloud storage services, ETL (Extract, Transform, Load) tools, and serverless computing tools. To improve maintainability, C1 usually uses tactics of *microservice architecture* (which means the separation of the whole system into smaller independent parts, with each part having its realm of responsibility) and *containerization* (packaging an application and its dependencies into lightweight, isolated units, allowing for consistent and efficient deployment across different environments). *Experiment tracking* (managing and monitoring of model configurations, hyperparameters, metrics, and outputs) and *dependency tracking* (managing and monitoring of the libraries, frameworks, and external components) are also widespread tactics to ensure maintainability.

To achieve high reliability of developed solutions, C1 usually uses *code versioning* (managing and monitoring of changes in both ML and non-ML specific code), *data versioning* (managing and monitoring of changes in datasets

used by ML models) and *pipeline management* (managing and monitoring of the dataflows and processes in a series of interconnected stages). Those tactics are usually implemented through experiment management tools. Finally, C1 developed its own model verification module (component designed to assess whether a model meets specific reliability requirements) that is usually integrated into safety-critical systems.

The basic set of architectural tactics such as *feature engineering* (selecting, modifying, or creating variables (features) from raw data), *data preprocessing* (cleaning, transforming, and organizing raw data into a suitable format) and *hyperparameter tuning* (searching for the optimal set of hyperparameters) are always applied to increase metrics of system accuracy.

Finally, *data encryption* algorithms are always introduced for security-critical and privacy-preserving systems. While security was not mentioned as a quality priority of the C1, it is often introduced for certain cases (e.g., for the healthcare domain).

C2 reported 3 architectural tactics. To address the “data quality” priority, the company usually conducts deep analysis and evaluation of data sources’ reliability. According to C2, if the sources are trustworthy, it solves fundamental issues associated with data used by models. Further, to improve system reliability the company introduces internal or external experts to verify the most safety-critical deliverables provided by ML. Finally, C2 developed *automated bias mitigation module* to exclude a set of sensitive parameters used by the model at different stages.

C3 also shared the 3 most frequently used architectural tactics. Similarly to C1, the company sometimes uses different *cloud-based* components in their systems and development processes, e.g., cloud databases, pre-built ML models, and AWS AI service [306]. The company sometimes integrates domain *experts in the loop*, who are introduced to guide the model with more appropriate decisions for the most complex cases to ensure system reliability. With the same motivation, supporting *rule-based models* (to handle categorical data) are introduced.

C4 shared 5 more tactics in addition to some of the ones mentioned previously. To save computational resources, the company introduces *model pruning* (removing unnecessary parameters from a machine learning model to reduce its size and complexity) and *data caching* (temporarily storing frequently accessed data in an operative storage layer, e.g., model outputs). To ensure reliability, the company integrates *user feedback module*, where it is possible, to receive operative information from end users on system faults and errors. Unlike C1, C4 primarily uses *feature engineering* to control the fairness of the system (e.g., absence of bias based on sensitive parameters, excluding outputs that violate privacy considerations). The unique priority of *explainability* is achieved by the integration of *Local Interpretable Model-Agnostic Explanations* (approximating behavior of the complex model with a simpler, interpretable model) and Shapley Additive Explanations (assigning each feature an importance value based on cooperative game theory) [307].

Our subject companies reported the following non-architectural tactics

Table 5.3: Used Non-architectural Tactics

Quality Priorities	Companies Reported	Non-architectural Tactics
Functional Suitability	C1	Regular Meetings with Clients
	C1, C3, C4	Knowledge Transferring Scrum Practices
Usability	C1	Early Involvement of End Users
Maintainability	C2, C3 C4	Supporting Documentation Knowledge Base
Security	C2, C4	Data Governance
Resource Efficiency	C3, C4	Cross-Functional Teams

presented in Table 5.3.

To ensure the full functional suitability of the system, C1 holds regular meetings with stakeholders. These meetings help to regularly adjust the budget, coordinate tasks, negotiate functional requirements, and allocate resources. To successfully fulfill functional requirements, knowledge transferring among teammates is conducted regularly in an operative manner. Finally, the overall development process is usually organized in accordance with main Scrum principles [308] which provide the most possible flexibility in conditions of constantly changing client’s priorities. A remarkable fact about C1 is its tactic to introduce end users in the early stages of system design and development to ensure the high usability of their solutions. According to C1, end users often differ from the formal clients. The company strives to always consider their needs as well.

To achieve high maintainability of the components integrated by C2, sufficient resources are invested in the development of supporting documentation explaining technical details and non-obvious dependencies. To address security and privacy issues C2 provides a data governance framework of policies, procedures, and standards together along with an integrated component to make data flows transparent to the clients.

C3, similarly to C1, emphasized the effectiveness of Scrum in successfully achieving project goals. Similarly to C2, the company provides detailed support documentation for its systems and components. C3 is considered a cross-functional team (team members have diverse expertise and skills from different functional areas) the most labor-resource-efficient way of organization in their domain.

C4 developed a knowledge base (centralized repository of information, resources, and expertise) for internal employees with a collection of best practices based on previous experience in development and maintenance. Ready-made insights are applicable to standard cases and sufficiently simplify maintenance. Like other companies, C4 follows Scrum practices, data governance principles, and cross-functional team organization.

The answer to RQ2 presented different perspectives on the use of quality-

driven tactics in ML-enabled systems. The most widespread architectural tactic was the use of *cloud-based components* to save computational and labor resources, while the most widespread non-architectural tactic was the use of *Scrum practices* to improve the functional suitability of final solutions.

### 5.3 Reported Quality Trade-offs

Based on the quality criteria and tactics identified in the previous research questions, our subject companies reported the quality trade-offs that affect their decisions most significantly. They are presented in Table 5.4.

C1 reported three quality trade-offs it deals with most frequently. According to C1, the first step in balancing any trade-off is to analyze the client's objectives and project scope. When it comes to the trade-off between resource efficiency and model accuracy, brainstorming by the team and the client is organized to prioritize the tasks and set performance thresholds (minimally acceptable levels of accuracy). Lack of labor resources can be compensated by the involvement of outsourcing powers requiring project budget extension. Computational resources can be compensated by the introduction of extra cloud-based components. The trade-off between resource efficiency and security appears rarely, but when it does, it seriously affects the whole project planning. First of all, security requirements from the stakeholders are deeply investigated by the team. When it is not possible to just use lightweight security protocols and outsource encryption algorithms, encryption specialists are introduced, which can lead to a significant increase of the project budget. The company strives to deliver highly reliable solutions, however, when the client requires extra robust functioning, the team spends extra resources on testing and simulations. Sometimes it introduces the ML-based algorithms of predictive maintenance to predict hardware failures and prevent downtime.

C2 usually deals with three main trade-offs. The first one is between resource efficiency and reliability. Since the team is quite small, it does not always have an opportunity to build complex models from scratch. In such cases, they agree with clients on the use of ready-made cloud-based models. The client and the team decide to be dependent on the third-party solution availability, however, the team provides at least one support alternative, which is a model with lower accuracy and performance that replaces the main one in case of third-party system failure. The second trade-off is between reliability and system accuracy. The team noticed the phenomenon of model overfitting when the formal metrics of accuracy are unrealistically high, but the ability of the model to generalize inputs is quite low. In such cases, the team introduces additional cross-validation, model pruning, or expanding the model training dataset. Finally, more complex models usually provide higher accuracy, but less explainability of their decision-making process. This case is totally dependent on the client's priorities. Customers are rarely interested in the explainability of the models, they accept models as black boxes as long as they efficiently solve their tasks. However, in cases when they are concerned, the team can replace complex models with simpler ones or introduce a technique of model stacking (by combining highly interpretable models (e.g., decision trees) with more complex models (e.g., deep learning)).

C3 reported the six most common trade-offs in their experience. Regarding the trade-off between system accuracy and resource efficiency, the company selects appropriate models that provide acceptable accuracy but fit computational limits. To find a balance between system reliability and resource efficiency, the company involves automated testing, parallel testing, and test optimizations. C3 also reported cases when teams were focused on increasing system accuracy and optimizing resources, however, main functional requirements built on the needs of stakeholders were associated with other quality attributes. For instance, there was a case when the speech recognition module primarily required high speed of processing, but due to blurred functional requirements team spent a lot of effort on increasing accuracy but lost the focus on the speed of response. The only solution there according to C3 is to monitor and clarify client's needs constantly. The tactic of balancing explainability and system accuracy is the same as for C2. Finally, the integration of the expert to maintain a system is essential, however, when comes to the maintenance of complex models, humans can provide biases based on their subjective decisions. In the area of machine translation, it is especially relevant. Data and model versioning are used for systematic tracking of changes, allowing for identifying and correcting bias introduced in specific versions.

For company C4, the majority of trade-offs emerge between maintainability and other priorities. That is because the main business process for C4 is associated with the maintenance and support of ML-based solutions. When addressing these trade-offs, the companies usually do not aim for a balance between the different dimensions. Instead, explainability, security, and usability are typically prioritized over maintainability, based on requirements by specific customers. To meet these requirements, the company allocates extra resources for solution development and maintenance, which leads to higher costs and increased budget needs.

The answer to RQ3 presents the most common quality trade-offs occurring across different companies. The most frequently reported trade-off is between *resource efficiency and reliability*.

Table 5.4: Identified Trade-offs

Trade-off	Company	Description
Resource Efficiency vs. System Accuracy	C1, C3	Significant labor resources are spent on manual data preprocessing, feature engineering and hyperparameter tuning. Significant amounts of computational resources (GPU) can be used to train own models instead of using existing ones to improve system accuracy.
Resource Efficiency vs. Security	C1	Development of complex cryptographic algorithms requires significant labor resources, while encryption of large datasets requires significant computational powers.
Resource Efficiency vs. Reliability	C1, C3, C4	Sufficient resources (both computational and labor) are usually invested in running simulations, stress tests, and monitoring system behavior under various conditions.
	C2	Relying on third-party models often makes sense in terms of resources (financial, computational, & labor), but if a third-party system fails or the server goes down, there is no way to restore the process manually.
Resource Efficiency vs. Functional Suitability	C3	The obvious strategy to conserve resources can detract from the main needs of stakeholders and real strategic goals.
Resource Efficiency vs. Maintainability	C4	Prioritizing immediate resource savings usually leads to shortcuts in development processes, such as inadequate testing or documentation. It significantly complicates further maintenance.
System Accuracy vs. Functional Suitability	C3	The obvious strategy to increase system accuracy metrics can detract from the main needs of stakeholders and real strategic goals.
Reliability vs. System Accuracy	C2	High accuracy of the model sometimes connected to overfitting, when the model cannot sufficiently generalize input data.
System Accuracy vs. Explainability	C2, C3, C4	Integration of complex deep learning models (e.g., Large Language Models) significantly improves the accuracy of system outputs; however, this makes it barely possible to explain how the model comes to certain conclusions.
Fairness vs. Maintainability	C3	If the model is maintained by humans, they can introduce potential bias by the decisions they personally consider the most rational.
	C4	Ethical considerations differ from one client to another. It makes it complicated for maintainers to properly set up bias mitigation modules and conduct data processing.
Usability vs. Maintainability	C4	Providing users with customization options improves overall system usability but introduces too many unnecessary dependencies that are complicated to monitor and maintain.
Security vs. Maintainability	C4	Strict security and privacy policies of security-critical clients introduce significant limitations on possible decisions for proper maintenance.
Explainability vs. Maintainability	C4	Highly explainable models (and designed systems) limit flexibility when adapting to new requirements. It leads to significant rework in case of changed priorities (for example, in favor of system accuracy).

# Chapter 6

## Discussion

Our findings shed light on the challenges faced by companies in balancing various quality attributes when developing ML-enabled systems. We now discuss their contribution to the theory of software quality of ML-enabled systems, their implications for practitioners and academics, and threats to validity.

### 6.1 Theory Building

For RQ1, by identifying quality priorities that exist across multiple organizations, such as reliability, resource efficiency, and functional suitability, we provide a high-level understanding of the strategic goals of real industry. This observation proves the fact that companies can operate in different ways and serve different clients, however, they often address similar priorities when it comes to building ML-enabled systems. For instance, reliability is essential for the safe use of machine learning in industrial conditions, while resource efficiency is important due to the high computational demands of ML models.

For RQ2, we found that integration of cloud-based components is frequently used to architecturally achieve those priorities. However, while cloud-based architectures can enhance resource efficiency and scalability, they may decrease reliability, by introducing dependency on third-party services. Surprisingly, no company reported concerns associated with privacy and security, despite the fact that ML training might require user data, and its transferring to an external cloud provider might lead to privacy concerns. A possible explanation is the confidence of providers in the tooling used and transferred responsibility for making decisions about the possibility of using cloud tools and components to the clients. Another finding is the adoption of Scrum as a non-architectural tactic to manage resources, ensure functional suitability, and provide teams and clients with the necessary flexibility.

Finally, for RQ3, our study of trade-offs reported by companies leads to an understanding of what dilemmas are usually met by AI developers, how the decisions are made, and what techniques of balancing are used.

## 6.2 Implications

This research provides actionable insights for both startups and SMEs, which often lack the resources to experiment with various tactics and learn from trade-offs in the same way that larger organizations might. First, the startup should select key quality priorities to follow from a wide spectrum of possible attributes. Secondly, it should implement appropriate tactics to achieve those priorities consistently, such as those reported by our subject companies. Thirdly, they should be mindful of trade-offs that are likely to appear in the process of design and development, which can potentially save significant resources—as the proverb says, “forewarned is forearmed”.

Researchers can use our findings in several ways. First, our list of identified trade-offs (Table 5.4) can serve as inspiration to develop new and improved solutions for important challenges in practice. For example, three of our four considered companies report on *system accuracy vs. explainability* trade-offs—with the recent trend on large language models (LLMs), there is a dire need for LLM solutions that are able to provide explanations. Second, they can develop automated recommender systems for practitioners to identify quality priorities, tactics, and trade-offs. Third, they can further their knowledge of quality trade-offs by focusing on particular domains and scopes of quality goals.

## 6.3 Threats to Validity

The main threat to external validity is determined by the specifics of certain companies (i.e. set of clients they serve, their resources, and their budgets). While we see the potential of applying our findings to the startups and SMEs, the specifics mentioned previously may have a more severe impact than we expect. Furthermore, the fact that all the selected companies are headquartered exclusively in Sweden may reflect the priorities of AI developers in a particular country. To mitigate this threat, it is possible to conduct more interviews with full competitors of studied companies (i.e. that have clients from the same domains and perform the same set of activities) or deeply analyze the specifics of startups and SMEs when applying those findings.

Internal validity is associated with the selected methodology of multiple case studies. Despite the fact that interviewees are directly involved in the decision-making process and design of ML-enabled solutions, they still can represent their subjective understanding of the described processes. Other employees within the same development team may provide other insights on the same issues. To mitigate this threat we involved more than one interviewee per company where it was possible.

The main threat to construct validity is associated with our strategy to rely entirely on the responses of interviewees. The questions were formulated in a way to explicitly highlights our intention to study the most important and frequently addressed quality priorities, tactics, and trade-offs. However, there is still a risk that perspective on the importance of those is determined by the responsibilities of a specific interviewee. To mitigate this threat, we involved only interviewees with the roles of CEO, project manager, or team leader. From our perspective, such roles can provide the most strategic insights.

# Chapter 7

## Conclusion

The findings emphasize the need for a more nuanced approach to addressing trade-offs between key quality attributes, particularly in resource-constrained environments such as startups and SMEs. This study investigated 9 quality priorities followed by 4 independent ML developing companies, 24 architectural and 8 non-architectural tactics to achieve those priorities, and 12 quality trade-offs that appear in the context of different projects per company.

Significant future work directions are to develop improved solutions addressing the identified trade-offs, recommender systems that support companies in identifying quality goals, tactics, and ways to balance trade-offs, and to compare the results from related work with the results of this study.

**Extracting Design Patterns from Mined  
Component Models of ML-Enabled Systems**

E. Eriksson, J. Olausson, **V. Indykov**, D. Strüber, R. Wohrab

Proceedings of the 51st Euromicro Conference Series on Software Engineering  
and Advanced Applications (SEAA), 2025, pp. 113–129

# Abstract

Machine Learning (ML) is becoming a widespread part of software systems. This results in a number of common ML-specific challenges that arise and recur across different systems, calling for consolidated solutions. While for traditional software, design patterns such as client-server or peer-to-peer architectures are well-established, a coherent body of design patterns for ML-enabled systems still has to emerge. Besides identifying and implementing such patterns, the analysis of their influence on different quality aspects is of special importance. It allows an architect to evaluate the solution from different angles and detect potential consequences of decisions made. In this paper, we present a collection of 14 design patterns extracted from models of ML-enabled software, and explore their impact on different system-wide quality characteristics. To identify these patterns, we conducted a multivocal literature review of 80 primary sources, which brought forward 49 component models from which the patterns were derived in a systematic manual process. As a form of evaluating the identified patterns, we performed an interview study with 10 experts who assessed their impact on relevant system quality attributes. To support the application of our patterns in newly encountered contexts, we systematically report our patterns with their definitions, addressed problems, examples, and relevant background.

# Chapter 1

## Introduction

Artificial intelligence (AI) is booming in various industries. With this increase in popularity, more and more software systems include AI components, often based on Machine Learning (ML). In an ML-enabled system, ML components typically make up only a small fraction [309]. Developing the vast infrastructure that surrounds the ML components leads to several distinct challenges, originating from aspects such as data handling and processing, data drift, and an extended set of relevant quality attributes for ML systems, including aspects such as fairness [310]. Such concerns typically need to be addressed at the level of a software system’s architecture [309].

Despite the massive uptake of ML research, which is often primarily focused on different model types and algorithms, research on the architecture of ML-enabled systems is only in its infancy and leaves various open challenges [310]–[312] that recur regularly in different projects [313]. Some of the most crucial challenges include a lack of architecture frameworks, processes, and evolution strategies for developing such systems [311]. Therefore, we notice the fundamental value of studying the whole system architectures rather than ML components or pipelines in isolation.

The most common high-level visualization of system architecture is the *component model*. Component models summarize the set of selected architectural decisions and highlight the most important parts of the system as well as the connections between them [314]. Such a source meets our intention to investigate how ML-enabled systems are produced in practice. Particularly, the analysis of several component models allows us to detect common decisions made in independent projects. By understanding which decisions are made recurrently, to address the same problems in the context of different projects, we identify *design patterns* for the engineering of ML-enabled software systems, whose systematic documentation can positively contribute to the evolution of ML-enabled software [315].

The implementation of design patterns generally affects the quality of the resulting system in multiple dimensions [315]. However, while some quality attributes can be effectively improved with certain design patterns, others can be affected negatively. Hence, to support the architect during their primary

risk analysis, accounts of design patterns should be supplemented with an analysis of their impact on the general quality characteristics of the system [316]. By connecting design patterns to quality attributes, software architects can roughly estimate the consequences of pattern implementation and predict the associated quality attribute benefits or costs [317].

In this paper, we present the results of a study in which we extracted 14 design patterns from available component models of ML-based software systems, and analyzed their impact on various quality attributes. Our data collection was based on extracting 49 component models of ML-enabled systems from different sources, including scientific literature, blog posts, repositories, and patent documentation), and analyzing them in a systematic manual process to identify recurring decisions. We examined these decisions in terms of the problems they address and relevant context and background, which led to an overall catalog of 14 patterns.

To evaluate our identified patterns, we interviewed 10 experts who assessed the patterns in terms of their impact on different quality attributes. We report the collected insights from our interviews in the form of graphical overviews of patterns and their impacts, and detailed justifications and insights from the interviews.

Software architects can use our results to improve the quality of the systems they design while being aware of the potential risks. Researchers can use this study as a basis for further investigation of component models and existing design patterns and update our findings with up-to-date patterns in the future.

# Chapter 2

## Background

We now revisit the necessary background for our methodology and results in four main directions: quality models for ML-enabled software, component models, design patterns, and our considered types of literature reviews.

### 2.1 Quality Models for ML-enabled Software

As we strive to contribute to the creation of high-quality software systems, “*comprehensive specification and evaluation of the software quality is a key factor*” [318]. The standard ISO-9126 for software engineering product quality [318] defines a quality model with quality attributes (QAs) to clearly outline where software systems may falter in quality.

Since machine learning relies heavily on data, “systems require consideration of several unique characteristics in addition to traditional ones” [319]. In our previous work [319], we performed a systematic literature review on quality attributes mentioned in the scientific literature on ML-enabled systems, and based on the results, created a quality model [319].

The model contains all the quality attributes from ISO-25010 with an exception for a quality attribute of compatibility since it was not frequently mentioned in the studied literature. We decided to use this model as a logical continuation of existing research in the subject area, supplementing it with the compatibility attribute from ISO-25010 [318] to get the fullest possible perspective on system quality.

Table 2.1 shows all quality attributes included in the quality model that are used in this study, together with their definitions. They were taken from our previous study [319] and from ISO-25010 [318] to define an additional attribute of compatibility.

The quality attribute of *functional suitability* is directly tied to the functional requirements of the system. It is a critical quality attribute, however, it cannot be generalized, since functional requirements vastly vary from one system to another due to the business logic or resources available. Functional suitability is still a part of the quality model used, but will therefore not be a part of the following research.

<b>Quality Attribute</b>	<b>Definition</b>
Functional Suitability	The degree to which a system corresponds to a defined set of functional requirements.
Resource Efficiency	The degree to which a system fulfills a given functionality within an existing amount of hardware or energy capacities.
System Accuracy	The degree to which a system performs and analyzes the contextual environment and refers to the performance of the entire system in real-world conditions.
Usability	The degree to which a system can be employed by end users to achieve specified goals.
Reliability	The degree to which a system performs specified functions under specified conditions in the fixed domain.
Security	The degree to which a system protects information and data.
Maintainability	The degree to which a system can be modified and supported by developers and maintainers to achieve specified goals.
Portability	The degree of effectiveness with which a system can be transferred from one domain, software or hardware basis to another.
Explainability	The degree to which the behavior of a system (primarily, the behavior of ML models) and its outputs can be explained by humans.
Fairness	The degree to which a system can detect and prevent an algorithmic bias created by a model.
Data Quality	The degree of integrity and sufficiency of data for model training, testing, and validation, including the reliability of the related data sources.
Compatibility	The degree to which the system exchanges information and performs its required functions, while sharing the same hardware or software environment with other system(s).

Table 2.1: Definitions of Quality Attributes [319].

## 2.2 Component models

According to Crnković, “A *component model defines standards for properties that individual components must satisfy and methods for composing components.*” [25]. Component models are unique to the systems they represent since they depict how the specific components of that system are composed and assembled [320]. Another important aspect of component models is that they represent connections or relationships between different components within the systems [314]

While a component model may be non-visual, for the rest of this study, only visual component models (in the form of graphical figures) are considered. This decision significantly improves the reproducibility of findings. Component models may be visualized as *component diagrams*, standard notation for software architecture, and one of the 14 diagram types of UML. However, in this study, we use not only component diagrams, but also other visual representations of component models to cover more potential sources.

## 2.3 Design Patterns

*Design pattern* is an important term in software engineering. Gamma et al. define design patterns as “*general solution[s] to reoccurring problems in software*” [321]. The SWEBoK defines design patterns as “*essence[s] describing both higher-level architecture and organization of code as well as lower-level design and implementation details*” [322] In this paper, we follow both definitions, with a focus on the first-mentioned kind of pattern from the SWEBoK: we consider design patterns that address the higher-level architecture of the system and its decomposition into components, as can be expressed in models, the focus artifact of our work. The results presented in our paper were all assessed with regard to their compliance with this terminology.

## 2.4 Literature reviews: SLRs, multivocal, and grey literature

*Systematic literature review (SLR)* is a technique commonly used for finding all information about a certain topic [37]. The way SLRs are performed makes them fair and of high scientific value, because they are transparent and reproducible, with as little bias as possible. A set of well-established guidelines exist for performing SLRs in software engineering, created by Kitchenham et al. [37].

There is a subset of systematic literature reviews, called *Multivocal Literature Reviews (MLRs)*, where the grey literature is included along with white literature. It gives the benefits of introducing the experience of both researchers and practitioners [323]. In a rapidly growing area of research, the inclusion of grey literature in a review can be highly beneficial [324], [325]. It also has the advantage of bringing research and practice closer together, which is especially important in areas with high practitioner interest or lack of corroboration between research and practice. Garousi et al. have created a set of guidelines for conducting MLRs in software engineering which can be used as an extension

to Kitchenham et al.'s guidelines for conducting systematic literature reviews [323].

The most widely accepted definition of grey literature was the Luxembourg definition [326]. Further, an updated Prague definition has been established that adjusts the definition in regards to publishing on the web [327]. In the guidelines for conducting MLRs in software engineering, Garousi et al. also highlight the existence of multiple definitions of grey literature [323] and imply to use of an adapted version of a model by Adams et al. [328]. This model is built on tiers of outlet control and credibility. However, Kitchenham et al. point out that these tiered models are “largely defined by example, which is a weak method of definition” [329]. They instead suggest using the Prague definition because it imposes grey literature to be *collected and preserved*, making the systematic review reproducible. However, this definition might lead to potentially valuable sources being neglected, which would not be in line with our goal of determining patterns used in existing component models from the literature, which is best solved by looking at both research and practice. Henceforth in this study, grey literature refers to the following definition, given by Garousi et al. in a more recent publication [330]: “*any material about software engineering that is neither formally peer-reviewed nor formally published.*”

Since the domain of software engineering for ML-based systems is characterized by dynamically growing interest from researchers and practitioners [331], we consider it profitable to use the methodology of MLR, including grey literature.

# Chapter 3

## Related Work

We discuss related work in three main directions: software architecture for ML-enabled systems, design patterns for the latter, and studies on the connection between design patterns and quality attributes.

### 3.1 Software Architecture for ML-enabled Systems

As software systems become larger and more complex, the need for proper structure when designing software arose [332]. To fulfill this need Perry and Wolf published one of the first papers on software architecture in 1995, accurately named “Foundations for the Study of Software Architecture” [333]. This opened up the field of software architecture and allowed others to continue the work within the field to, for example, analyze common architectural styles [334], a precursor to architectural patterns. When the internet was introduced and systems became more connected, a new focus on the systems’ non-functional qualities and ways of achieving them arose [335].

Now, when more software systems integrate ML, there is a need to reconsider how important data and algorithms are [336] since they are fundamental building blocks of ML-based systems [311], [337] but “do not appear prominently in software architecture literature” [336]. Therefore, there is a need to investigate new architectural techniques as a response to emerging challenges.

### 3.2 Design Patterns for ML-enabled Systems

Design patterns have been investigated in software engineering for more than four decades [338]. Within AI specifically, design patterns for responsible AI (i.e. the ability of AI to behave ethically and responsibly) [339], and multi-agent systems (i.e. a subfield of AI involving the interaction of intelligent agents) [340] have also been studied for latest decade. There have been studies on design patterns for the Internet of Things [341] and microservices [342].

However, to our knowledge, only two secondary studies on design patterns for ML-enabled systems have been produced. Washizaki et al. conducted a multivocal literature review to find patterns for ML architecture and design, which they claimed was the first of its kind [343]. In total, they found 33 patterns and anti-patterns, with only two of them described in detail. Later, Washizaki et al. continued the work on these patterns and narrowed them down to 15, removing those that were vaguely defined or had questionable usefulness [344]. The identified patterns are either high-level architectural patterns (e.g., “Distinguish Business Logic from ML Models”), or more design-oriented (e.g., “Data Lake”). In 2023, Heiland et al. conducted an MLR with a similar purpose — identifying design patterns for AI systems [345]. They identified 70 patterns, 36 of which they classified as “traditional patterns”, i.e., patterns that can be found in non-AI contexts but have been adapted to an AI context, while the rest remained exclusively AI-specific. All patterns were categorized into different groups, for example, *deployment*, *implementation*, or *security and safety*. The category with the majority of patterns was *architecture* with 25 of them, which the authors say adds to the current understanding of the importance of architecture and its reuse. Design patterns from this category are of special interest for our paper. The methodologies used in the studies of Heiland et al.[345] and Washizaki et al.[343] were different from the one used in the present work and led to different results. Unlike both previous studies, we used a mining methodology, based on component models of existing systems as a main source, while the previous work relies on available textual descriptions of patterns. The obtained lists of patterns are different and complementary to each other. In addition, we supplement our list of design patterns with their evaluation by experts and investigation of their quality consequences.

### 3.3 Connection of Design Patterns to Quality Attributes

It is established that using design patterns correctly can significantly improve the quality of the overall system in a resource-efficient manner [315]. Therefore, we can assume that patterns have a direct impact on the quality attributes of a given system. This line of reasoning is not something unique [346], many previous studies found the connection between design patterns and QAs [315], [344], [347]–[349]. However, evaluating the connection of patterns to quality attributes remains a complicated task due to the context dependency, as stated by Mayvan et al. in a systematic mapping study on design patterns [338].

Some studies have attempted to answer the question of QAs’ connection to design patterns. A study on microservice patterns found a different distribution for which quality attributes were associated with the microservice patterns compared to what Wedyan and Abufakher found [347]. The methodology for identifying the QAs that were addressed by each architectural pattern was based on the in-depth analysis of the associated literature and the interviews with experts and practitioners.

Valdivia et al. [348] conducted a multivocal literature review to find patterns in microservices. Further, they identified connections to quality attributes by

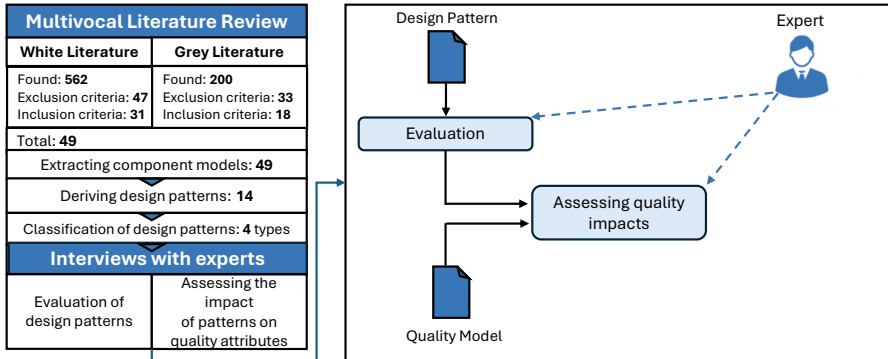


Figure 3.1: Overview of the employed methodology

reading the sources describing the patterns. This study has several similarities with current research with two exceptions: current research is focused on ML-enabled systems instead of microservices and contains an impact evaluation of the quality attributes through interviews with experts. These studies show that answering how the qualities of a system are connected to design patterns is ambiguous and accessible to different points of view. In our study, we strive to focus on generalizable insights leaving the specifics of certain systems behind and involve experts to get primary assessment of the impacts of design patterns on the common quality attributes.

# Chapter 4

## Methodology

The aim of this study is to identify existing design patterns in ML-enabled software and explore their impacts on different system quality characteristics. We address and answer the following two research questions (RQs):

**RQ1:** *What common design patterns can be derived from existing component models of ML-enabled software systems?*

**RQ2:** *What impact do design patterns have on the common quality attributes of ML-enabled systems?*

Our research method for addressing these questions, outlined in Figure 3.1, has the following main components: To answer RQ1, a multivocal literature review was conducted to extract component models, with further derivation of design patterns from them. The answer to RQ2 is based on interviews with experts to evaluate found patterns and identify the quality consequences of their application.

### 4.1 Multivocal Literature Review

A multivocal literature review was conducted to systematically elicit data used to answer RQ1. We followed guidelines by Garousi et al. for conducting MLRs in software engineering [323]. As suggested in these guidelines, Kitchenham and Charters' guidelines for conducting SLRs in software engineering [37] were used, but with the former as variation points.

For the MLR, a review protocol was developed and tested with a pilot search. The review protocol can be found in the Supplementary Artifact [350].

### 4.2 Search Strategy

The search was performed in four databases for white and two databases for grey literature.

For white literature, the databases *IEEE Xplore Digital Library*, *Wiley Online Library*, *ACM Digital Library*, and *SpringerLink* were used. These were selected to get a wide coverage of literature within the software engineering field.

Grey literature was searched through the *Google Images Search Engine* as well as through the *Google Patents*. Since the main task of search strategy for RQ1 is to find graphical component models, Google Images is considered the most effective tool to find such type of content with links to the sources from which they were extracted [351]. However, as we noticed through the pilot search, this tool is of limited applicability to find the images from patent documentation. Therefore, we decided to run a separate search through Google Patents to find patent documentation of appropriate solutions and manually extract component models from it.

The search strings were derived by assembling a list of synonyms and similar phrases to “component model” and “machine learning” respectively. The Software Engineering Body of Knowledge (SWEBOK) [322] was used to aid in the former. For white literature, the search string was as follows:

```
("component model*" OR "architectural description language*" OR
"component diagram*" OR "class responsibility collaborator*"
OR "structure chart*" OR "component based design*") AND
("ml-system" OR "ml-enabled system" OR "machine learning" OR
"deep learning" OR "reinforcement learning" OR "unsupervised
learning" OR "supervised learning" OR "neural network") AND NOT
"artificial intelligence"
```

The wildcard operator \* specifies any number of unknown characters, which in this case is used to include inflections of some of the terms. There has been a huge rise in interest in artificial intelligence within the last decade, seen not least by the number of published publications on the matter [331]. Machine learning is a form of AI, which means that ML has followed a similar trend. To remove the most obsolete machine learning systems, all searches were filtered from 2014–2024. This should provide good chances that the source is still relevant, even in this rapidly growing field.

A side effect of the trend on AI is that a large part of the research is not relevant when looking for machine learning systems exclusively. We observed that papers about artificial intelligence tend to be too broad. For these reasons, papers that contained “artificial intelligence” as a key phrase were excluded. After the removal of “artificial intelligence”, the amount of search results was roughly halved. Since the amount of sources before this was very high, this was a way to make the study feasible.

Since Wiley and SpringerLink are both databases that are not computer science-specific, the pilot search showed a lot of sources focused on how to apply machine learning in different fields. Due to this both of these databases were filtered exclusively in the area of computer science.

The search string for Google Images had to be modified since there is a limit of 32 words for a query. Different search strings were tested to find the one that gave the most appropriate results after a pilot search was performed. One more important limitation is that Google Search and Google Image Search do not support parentheses in queries (i.e. complex queries), which results in the following search string:

```
"machine learning" AND "component model" OR "component diagram"
```

When searching grey literature in Google Images, a private browsing mode was utilized to prevent Google from tailoring the search results depending on the earlier activity [351], [352].

Finally, the following string was used to find patent documentation in Google Patents:

```
("ml-system" OR "ml-enabled system" OR "machine learning" OR  
"deep learning" OR "reinforcement learning" OR "unsupervised  
learning" OR "supervised learning" OR "neural network") AND NOT  
"artificial intelligence"
```

After a pilot search, we noticed that the patent documentation about machine learning often includes architecture in the form of component models, making that part of the string unnecessary to include, to the point where it dilutes the search results.

## 4.3 Study Selection

The following three inclusion criteria were applied for both white and grey literature, where each one has to be true for a source to be included:

- The focus of the source is on the *system itself*.
- The system described in the source is a *machine learning system*.
- The source contains one or more figures showing *the relationship between architectural components*.

The exclusion criteria partially differed between white and grey literature because of their nature. For white literature, the following exclusion criteria were applied:

- The source is *not written in English*.
- The source is *published before 2014*.
- The source is *not available in full text*.
- The source is *not a single chapter or paper*.

For grey literature, the following exclusion criteria were applied:

- The source is *not written in English*.
- The source is *published before 2014*.
- The source is *in video or audio format*.
- The source contains *component models or equivalent only as an example of what they are or how they are used*.

The stopping criteria for grey literature was selected to be effort-bound to 100 sources for each database. One hundred was arbitrarily chosen and puts a lot of faith in Google's search ranking <sup>1</sup> to give the most relevant results first.

Each source was assessed by one of the authors. If the source could not be assessed unambiguously, it was discussed with the other authors.

## 4.4 Data Extraction Strategy

All identified figures of component models are extracted, copied, and saved with an identification number that traces back to the source and put into a data extraction form. In this form, personal notes were written which included summaries of the component models. Further, extra columns were added for additional information about the system, for example, the type of machine learning that was presented (e.g., reinforcement learning, neural networks, deep learning). Each component model was also categorized into which type of system it depicts: either a whole system, a part of a system, or machine learning pipeline. Data extraction sheets can be found in the Supplementary Artifact.

To ensure the consistency of data extraction, two authors of this study conducted data extraction process on the same set of sources independently. Further, the third author was involved in conducting data extraction on the random sample of 10 sources found. As a results, no extraction conflicts were met, which according to Kitchenham's guidelines [37] is an appropriate way to prove sufficient levels of data extraction consistency.

## 4.5 Literature Review Process

All found sources were listed in a document with titles and links so that sources that appeared multiple times could be removed. Out of 562 white literature sources, 8 were duplicated (and were further removed). Further, 7 white literature sources were not available in full text and 9 were not a single chapter or paper and were therefore not considered because of the exclusion criteria.

The distribution of all sources can be seen in Figure 4.1. The two grey literature databases are up roughly one-fourth of all sources.

After inclusion criteria has been applied there were 49 sources left, from which an equal number of component models could be extracted.

The distribution for these 49 sources can be seen in Figure 4.2. The share of grey literature sources went from 26.6% before the inclusion criteria had been applied, to 36.7% after. This increase is quite high, and almost all of it is from Google Images. This highlights the importance of including grey literature for the chosen topic.

A summary of the number of sources for each database after each filtering step in the literature review process can be seen in Table 4.1. Out of the total 754 unique sources of white and grey literature, 49 component models were

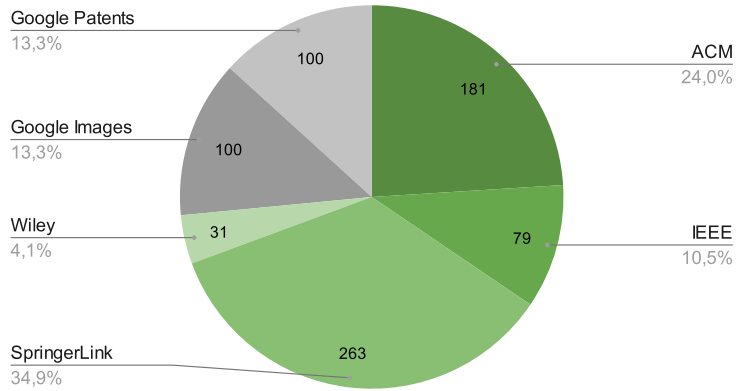


Figure 4.1: The distribution of the sources found per databases

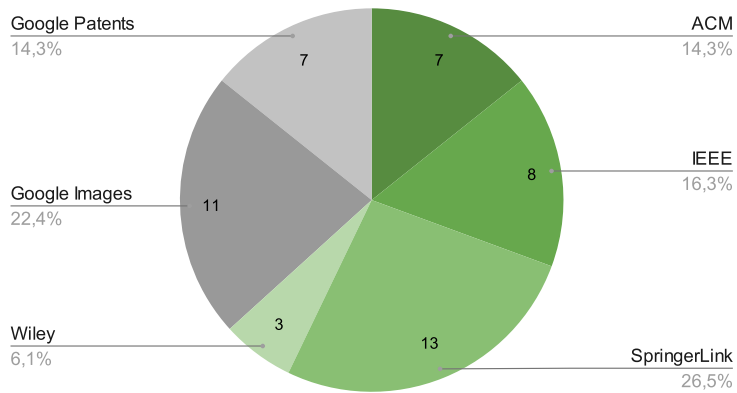


Figure 4.2: The distribution of the sources selected per databases

Step	ACM	IEEE	Springer	Wiley	Total White	Google Images	Google Patents	Grand Total
Init. search	183	82	266	31	562	100	100	762
Excl. duplic.	181	79	263	31	554	100	100	754
Excl. criteria	9	9	26	3	47	21	12	80
Incl. criteria	7	8	13	3	31	11	7	49

Table 4.1: Source selection process

extracted, giving a source inclusion rate of about 6.5%.

## 4.6 Pattern Elicitation

When the final set of component models was compiled the extraction of patterns from them had started. The first step was to find similarities between several component models. Similarities in this context were the same design decisions or implemented techniques made in two or more independent cases. Further, the previous literature reviews on ML or AI architectural design patterns were examined, i.e. the ones by Washizaki et al. [344] and Heiland et al. [345]. Furthermore, some of the identified similarities were found to be established patterns with an already given name, problem, and solution. This resulted in some extracted patterns being already established patterns, while some were brand new and identified exclusively by the current study.

For the already established patterns, the problems a pattern solves can be directly found through the related literature, e.g., Heiland et al.'s MLR [345]. For the new patterns, the problems a pattern solves were constructed by the authors based on their descriptions presented in the sources from which they were extracted.

To support the definition of patterns as solutions to recurring problems, our current study only considers essences that appear in at least two component models and address the same issues according to their descriptions in the associated literature. This creates a threat that the list of patterns is not entirely complete. In this study, we aim to elicit the most commonly used design patterns, rather than aiming for completeness. We describe the associated threats to validity in Section 7.

The pattern elicitation process was carried out in an iterative manner where each component model was printed on paper and analyzed twice, independently by two different authors, to reduce the risk of missing patterns and to ensure the consistency of data synthesis. From each component model, we independently extracted a list of architectural decisions presented there. We checked what decisions were mentioned at least in two lists extracted from models. Further, we checked if those decisions solve the same problems according to the associated literature. If the decision appeared in at least two component models and solved the reoccurring problem, it was listed as a design pattern. Two authors came to the same results independently. The list of patterns was further evaluated and agreed upon by the rest of the authors. This method of synthesis may rely on the experience of the authors and can be considered a threat to validity (discussed in Section 7). To increase the reproducibility of our findings, we share all the extracted component models and the lists of architectural decisions derived from them in the Supplementary Artifact [350].

## 4.7 Interviews

To evaluate the identified patterns and their connections to quality attributes, we decided to involve experts. More specifically, semi-structured

---

<sup>1</sup><https://www.google.com/search/how-search-works/ranking-results/>

interviews were planned and conducted to elicit data that could be used to answer RQ2.

Semi-structured interviews give the right balance between freedom and limitations when it comes to the questions addressed, and let the interviewees think freely about what QAs stick out the most to them, as well as how large of an impact and the reasons why.

During the planning and conducting of the interviews, Hove and Anda's recommendations for semi-structured interviews in empirical software engineering research [353] were followed.

Interviewees were recruited via email from our existing networks. We focused on recruiting experts with knowledge of the engineering of AI-enabled systems, either due to a focus on AI, or on software engineering with a focus on AI systems. To that end, we invited faculty members from assistant professors to full professors of the Data Science and AI (DSAI) and Interaction Design and Software Engineering (IDSE) divisions of Chalmers University of Technology and the University of Gothenburg. The emails contained an overview of the research topic to explain the purpose of the interview.

To ensure a consistent procedure for the interviews, we created an interview guide up front, which was followed during the interviews. Below, we give an overview of the main questions, providing the full guide in the Supplementary Artifact [350].

- **Step 1.** To provide some general information, including an introduction to the study and why the input of interviewees is needed for the study.
- **Step 2** To ask interviewees to briefly share their experience with ML and ML-enabled systems.
- **Step 3.** To present the quality model (Table 2.1) and ask if interviewees have any questions regarding it.
- **Step 4.** To present a design pattern.
  - **Substep 4.1** To explain the definition of the pattern and an example of it in use.
  - **Substep 4.2** To ask them if interviewees have seen or used something like this before in practice.
  - **Substep 4.3** To ask which quality attributes they think the pattern affects and why.
  - **Substep 4.4** For each quality attribute connection, let interviewees rate on a scale from -3 to +3, how critical the positive or negative impact the pattern has on the quality attribute.
- **Step 5.** If time is available, go back to step 4. Otherwise, end the interview by thanking the interviewees for their participation.

Overall, 10 interviewees participated in the study. All of them had academic backgrounds and practical or theoretical experience in the design of ML-based software or integration of ML functionality. The assignment of the selected

Design Pattern \ Experts	Data Science Division						Interaction Design & Software Engineering Division			
	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
Ensemble learning		☑			☑		☑			
Parallel independent models	☑		☑				☑			
Sequentially dependant models		☑		☑					☑	
Situation-based model selection	☑				☑		☑			
Data transformation				☑		☑				
Feature extraction			☑			☑				☑
Post-processing			☑							☑
Prediction cache				☑		☑				
Multi-layer pattern							☑		☑	
Orchestrator							☑	☑		
Server-side ML model								☑	☑	
Expert validation	☑				☑					☑
Runtime eval. and improvement			☑							☑
System eval. and improvement								☑	☑	

Figure 4.3: The distribution of design patterns presented to each interviewee

patterns per interviewee is presented in Figure 4.3. The assignment was made according to the academic background and main area of expertise of each interviewee.

The interviews were recorded and transcribed. To extract the data from the interviews we went through the transcriptions of the meeting recordings to summarize the main points and their associated impact ratings for each pattern and quality attribute combination. To ensure the consistency of data extraction, this process was conducted independently by two authors for each conducted interview. Motivation was the most important aspect of the interview since the impact on the quality attribute is based on it. Therefore, ratings without motivation were removed from the interview data because the reader must be able to see how the pattern impacts the quality attribute. Motivations without a rating were not removed since removal would obscure important information from the readers.

# Chapter 5

## Design Patterns

We now present the results of RQ1, on the identification of design patterns. Figure 5.1 presents an overview of the identified design patterns along with the frequency of occurrence of each pattern in the 49 considered component models. Each pattern occurred between 8 and 2 times, reflecting the notion that patterns are *reoccurring* solutions to problems. In total, we derived 14 design patterns from 49 component models, grouped into four overarching categories depending on the level of abstraction on which they are implemented: data-level, which describes data flows used both in *training system* (that primarily characterizes ML pipeline for training an ML model), e.g., training data, and *deployed system* (that primarily characterizes the executed system that includes a trained ML model), e.g., real-time input data; model-level, which describes how several models can be organized in the training system; deployed system level, which represents high-level architectural design decisions of the deployed system and, as a cross-cutting concern, the validation level, which describes evaluation of processes in the training and deployed systems. Note that ML-enabled systems may embed the training system into the overall architecture to introduce continuous retraining and improvement based on new data [85]. However, in some cases, the training system can be independent.

We now present the patterns in detail, following the grouping as explained above.

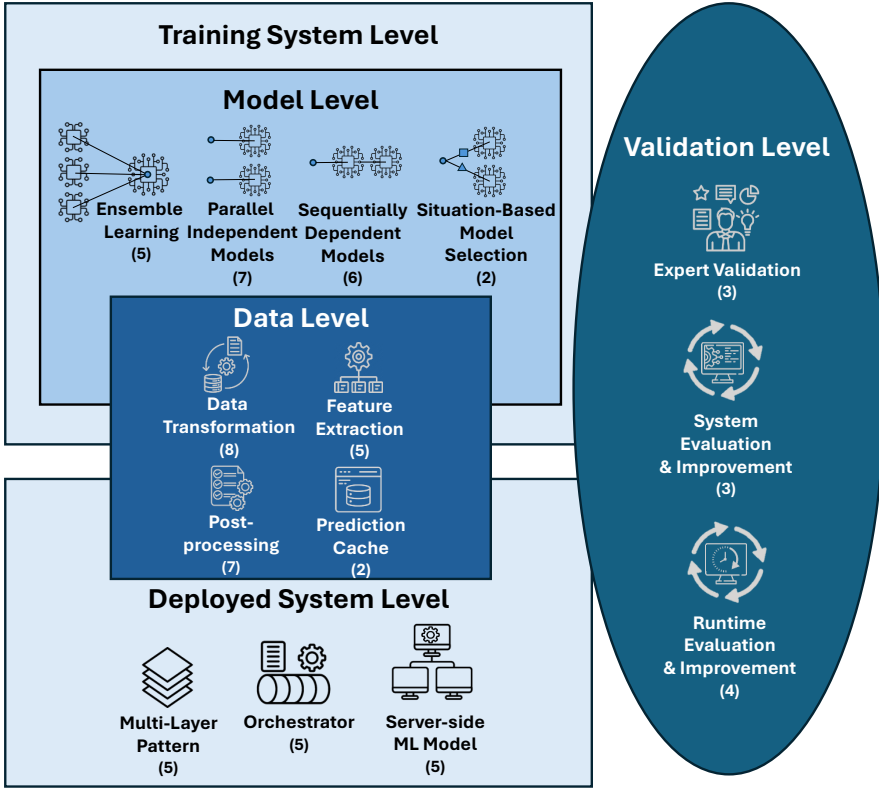
### 5.1 Model-Level Patterns

These patterns concern how a system can utilize multiple models in different ways to achieve different goals.

#### 5.1.1 Pattern 1: Ensemble Learning

*Definition:* A system running multiple models, combining them into one prediction. This can either be done by multiple models running on the same data or different parts of the same data.

*Problem to be solved:* Individual machine learning models may make errors, but the errors might not always overlap.



(x) – number of component models from which the design pattern was extracted

Figure 5.1: Overview of our 14 identified design patterns in 4 groups

*Background:* Ensemble learning is an established technique for combining the output of multiple models to increase model accuracy and resilience by studying the same problem from different angles [354].

*Example:* Figure 5.2 depicts a system trying to find *Botnet* domain names [355]. Botnet domains have names either of random characters or random words. The system looks for these features in parallel and then combines the answers from the two models to predict if it is a botnet domain.

### 5.1.2 Pattern 2: Parallel Independent Models

*Definition:* Running multiple independent models on the same input data to find different characteristics using different specialized models to get different outputs.

*Problem to be solved:* A need to differentiate between different types of information in the data within a frame of complex analysis.

*Background:* This pattern is related to *Microservice horizontal pattern* as described by Heiland et al. [345]. In our work, we separate parallel independent predictions from parallel combined predictions (where the integrated prediction

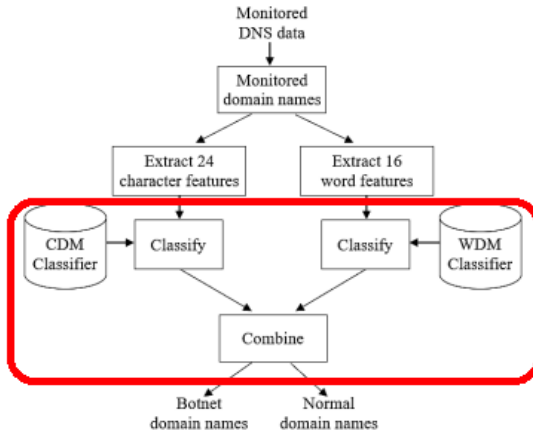


Figure 5.2: Example of the ensemble learning pattern taken from [355] with the pattern circled in red

depends on multiple models), which is instead classified as ensemble learning. Running multiple models specialized in finding their niche characteristics within the data can allow the system to more reliably detect these characteristics.

*Example:* In image processing, differentiating between different types of objects can be tricky for an ML model. Therefore, it could be easier and more reliable to have multiple models searching for different things. An example of this can be seen in Figure 5.3, where the image processing searches for traffic lights, lanes, and obstacles independently of the same input image.

### 5.1.3 Pattern 3: Sequentially Dependent Models

*Definition:* Running multiple ML models in sequence, each with a distinct responsibility of solving a smaller step in the process of solving a greater problem.

*Problem to be solved:* The need to break down a complex ML problem into a sequence of simpler sub-problems, where each model is responsible for its own sub-problem and uses the output of the previous model as input or corrects its errors.

*Background:* This pattern is related to the *Microservice vertical pattern* in the work by Heiland et al. [345], with a similar distinction to the case of parallel independent models.

*Example:* An example of this can be seen in Figure 5.4, where they

### 5.1.4 Pattern 4: Situation-Based Model Selection

*Definition:* When the system has multiple models to choose from, and picks one of them to use based on the situation.

*Problem to be solved:* A need to select the most efficient solution based on the specific characteristics of the problem, contextual environment, or certain conditions.

*Background:* This pattern can be seen as a specialized case of the strategy pattern [321]. The models are interchangeable and can be changed at runtime.

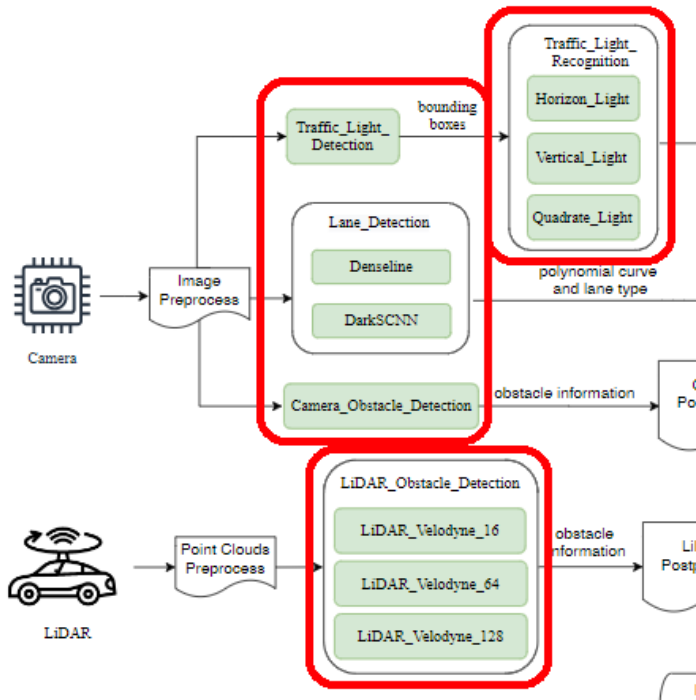


Figure 5.3: Example of the parallel independent models pattern taken from [356] with three separate instances of the pattern circled in red

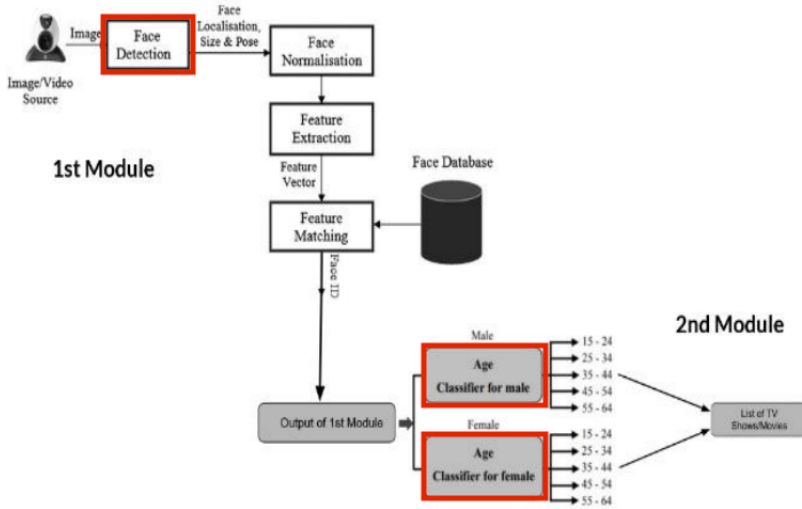


Figure 5.4: Example of the sequentially dependent models pattern taken from [357], with the machine learning models circled in red

*Example:* As seen in Figure 5.5, the “Scenarios Manager” has nine different models for different scenarios or states that a car can be in, with tailored models for each state to predict the optimal trajectory.

## 5.2 Data-Level Patterns

These design patterns concern how training, testing, input, and output data are handled.

### 5.2.1 Pattern 5: Data Transformation

*Definition:* Data is transformed through a series of procedures, which produce incremental results to transform the data into the desired type of input for the model.

*Problem to be solved:* The raw data is not in the correct format for the ML model and needs to be broken down or changed so it fits the model.

*Background:* This pattern is related to the traditional *pipes and filters* pattern [345], which involves breaking down a system into a series of data processing steps, expressed as filters. A strict implementation of pipes and filters leads to particularly strong modularity by explicitly creating pipes through which transformation steps are performed, which allows for freely exchanging the filters running between them.

*Example:* As seen in Figure 5.6, the input, in this case, pictures of blood vessels, are taken and filtered in different steps. The transformation is first to find the blood vessels, and then extract the parts of the blood vessel that might be an aneurysm, which is done via image processing and not ML.

### 5.2.2 Pattern 6: Feature Extraction

*Definition:* When the input for the model first goes through a component that

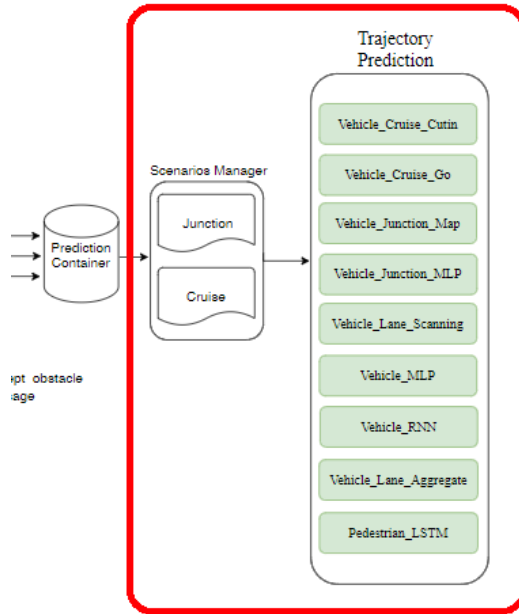


Figure 5.5: Example of the situation-based model selection pattern taken from [356], with the pattern circled in red

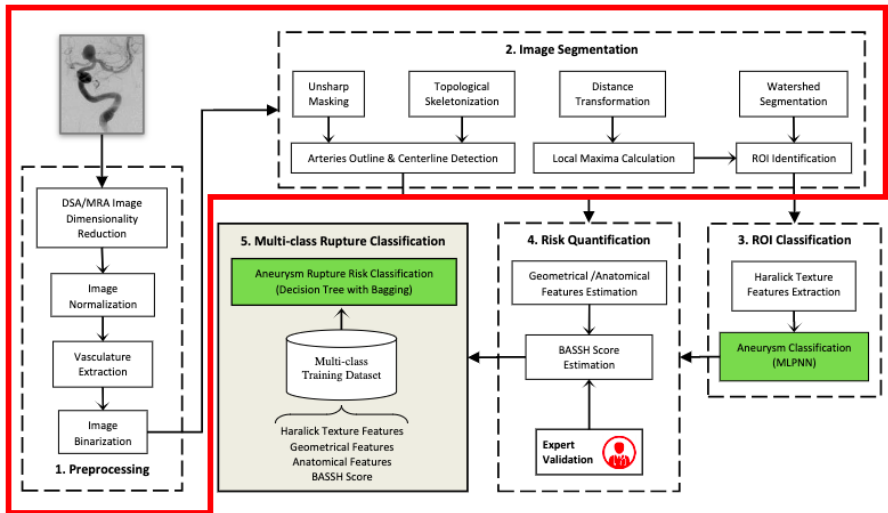


Figure 5.6: Example of the data transformation pattern taken from [358] with the pattern circled in red

extracts a set of features, instead of using the raw data as input for the model. *Problem to be solved:* A need to reduce the complexity of the model running on a dataset with large entries and improve its performance.

*Background:* Feature extraction is a well-established technique within machine learning [359] for decreasing the size of the feature space without losing information about the original feature space [360].

*Example:* In Figure 5.7, we can see a system that takes in domain names and extracts two sets of different features that run through the models. In this case, it is two different models, and their output is combined, but it could be any type of ML model, and all features can be input for the same model.

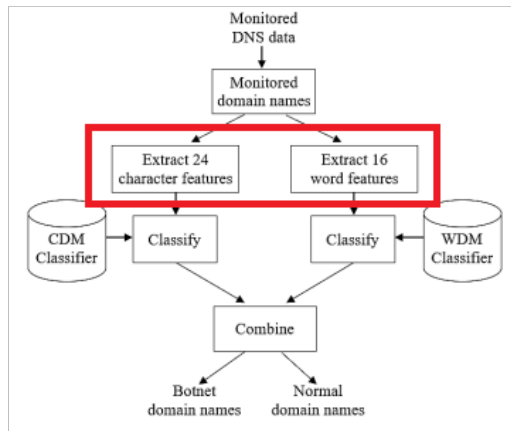


Figure 5.7: Example of the feature extraction pattern taken from [355] with the pattern circled in red

### 5.2.3 Pattern 7: Post-processing

*Definition:* If the output of the model is not in the format wanted by the system, it is fed into a component that processes the model output to the desired format.

*Problem to be solved:* The output from the model is in the wrong format, or some information or features need to be reconstructed since they can be lost in the model.

*Example:* Figure 5.8 shows an example of post-processing from a Bose Inc. patent. After the ML component has filtered out background noise from a microphone input, a component is used to reconstruct the sound of the person speaking into the microphone. Thanks to the reconstruction component, the background noise is removed, but the sound of the speaker can be preserved.

### 5.2.4 Pattern 8: Prediction Cache

*Definition:* The input and/or prediction is saved so that it can later be queried to see if it is unique.

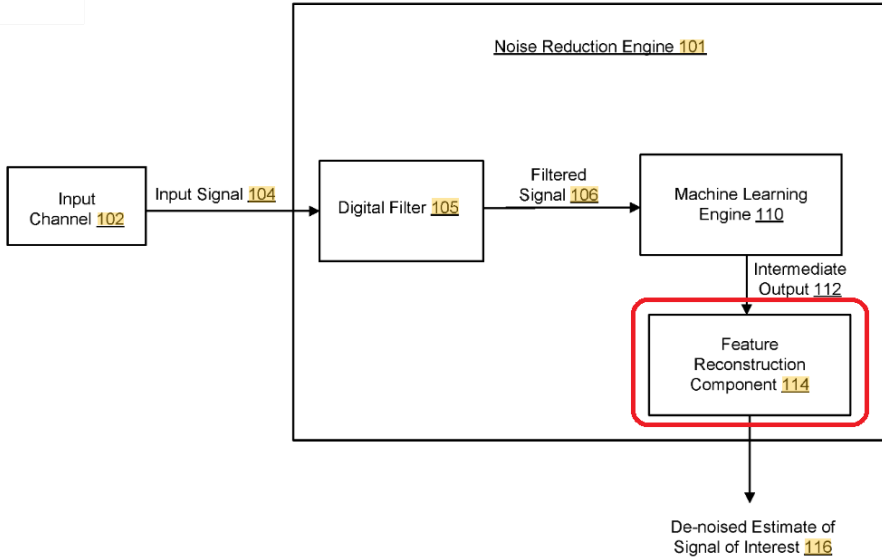


Figure 5.8: Example of the post-processing pattern taken from [361] with the pattern circled in red

*Problem to be solved:* A need to optimize resources when the input to the model is repeated multiple times and the same output is expected.

*Background:* This pattern exists in Heiland et al.’s work [345] with the same name. A similar pattern called *Data cache* is also presented there. While the prediction cache in Heilands et al.’s work is specific to predictions and the data cache is more towards preprocessing of data, in the context of the current study, they are merged since there is no significant architectural difference between the two.

*Example:* In Figure 5.9, we can see an example of the prediction cache pattern. This system takes real-time news and Twitter info and boils that info down to a simpler version. Since multiple sources can report the same event, there is a component at the end that saves each prediction, in this case, an event representation. These are then checked to see if any of the ones made before are close enough that they represent the same event to make sure the event feed is not flooded each time an event occurs.

### 5.3 Deployed System-Level Patterns

These patterns concern how components are structured and how they communicate with each other. These patterns do not concern the specifics of the models themselves. However, it should be highlighted that since these patterns were found in ML-enabled systems, they are suitable for this type of system even if they do not concern the model-specifics.

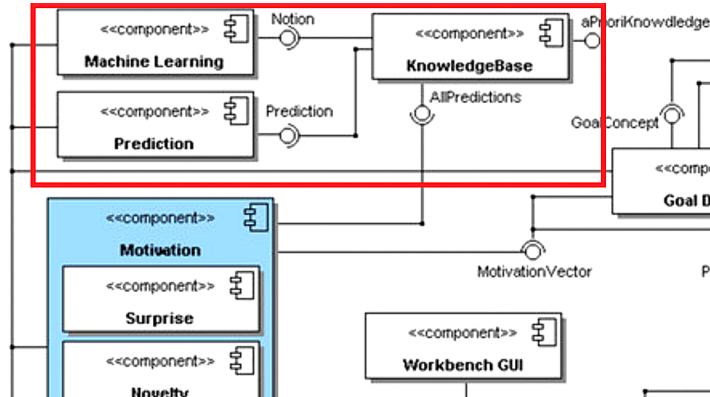


Figure 5.9: Example of the prediction cache pattern taken from [362] with the pattern circled in red

### 5.3.1 Pattern 9: Multi-Layer Pattern

*Definition:* The system is divided into different layers, which have clearly defined purposes, where each layer communicates only with its closest neighbors.

*Problem to be solved:* A need to introduce the *separation of concern* as well as *low coupling, high cohesion*.

*Background:* This pattern exists in Heiland et al.'s work [345], where it is described as a traditional pattern (i.e., a pattern that appears outside AI/ML contexts). This pattern is also known as *Separation of Concerns* or *Multi-Tiered Architecture* [345].

*Example:* In Figure 5.10, we see an example of a system employing the multi-layer pattern. This system has two separate paths, one for Twitter and one for news feeds, where it looks for new events. The information from these is passed through a series of layers that first detect events, secondly, filter the events (only for Twitter), then find the information about the event that the system wants, and finally look for uniqueness before publishing to the event feed.

### 5.3.2 Pattern 10: Orchestrator

*Definition:* A central component that handles and initiates communication between the machine learning-related components, including but not limited to data handling, training, evaluation, and prediction requests.

*Problem to be solved:* A need to efficiently manage interactions and data flows among diverse components within an ML-enabled system.

*Background:* This pattern is similar to the *mediator pattern* in the work by Heiland et al. [345]. But in contrast to the mediator pattern, the orchestrator is the initiator of the contact with the components, while in the mediator pattern, the components communicate with each other through a mediator. The problem they solve is therefore different, meaning they should be separate patterns.

*Example:* As seen in Figure 5.11, the orchestrator connects all the components. It manages the different models that a model builder creates with the data

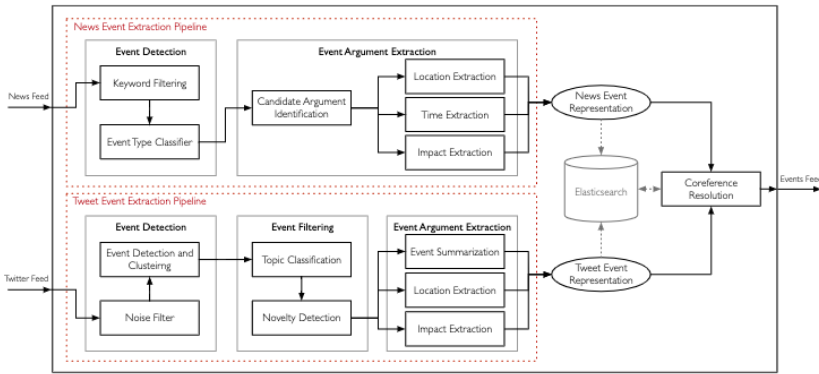


Figure 5.10: Example of the multi-layer pattern, taken from [363]

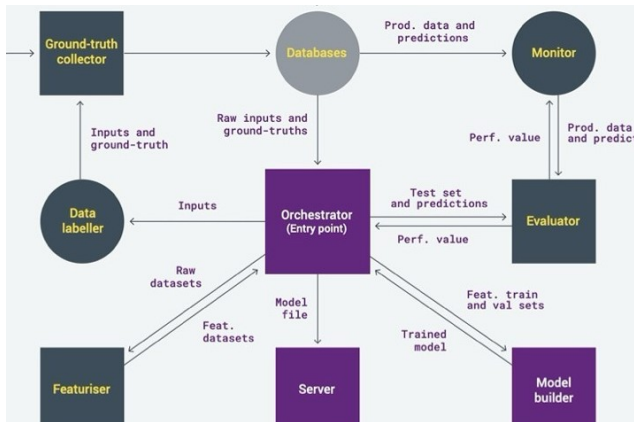


Figure 5.11: Example of the orchestrator pattern taken from [364]

supplied by the orchestrator, as well as evaluation, supplying models to the server, etc.

### 5.3.3 Pattern 11: Server-Side ML Model

*Definition:* One component has the role of server and allows components or sub-systems to take the role of client, initiating a connection with the server to obtain a prediction from its model.

*Problem to be solved:* A need to ensure efficient processing, scalability, and secure access to predictions for different clients.

*Background:* Related to the *Client-Server* pattern in Heiland et al.'s work [345], but here highlighting that the model is on a server, centralizing it for an arbitrary number of clients to use.

*Example:* In Figure 5.12, an example of the pattern can be seen where the sub-system at the bottom of the figure acts as a server and provides its machine

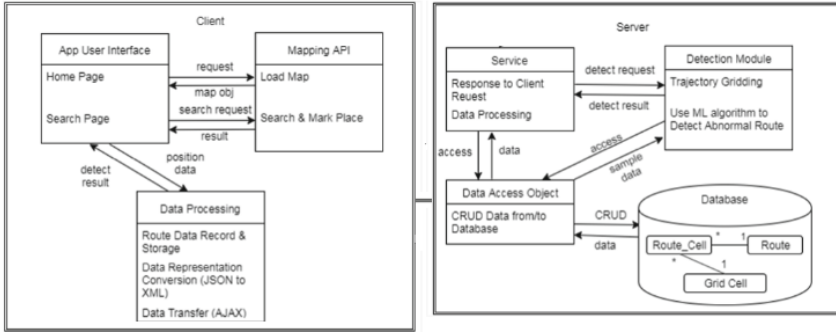


Figure 5.12: Example of the server-side ML model pattern from [365]

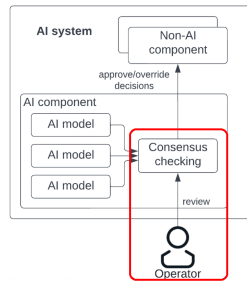


Figure 5.13: Example of the expert validation pattern taken from [366]

learning service to the client at the top.

## 5.4 Validation-Level Patterns

These patterns concern validation within the systems in some way, either by evaluating an ML model directly or an application within a larger system to ensure their correctness, performance, or reliability. By having some form of validation in these ways, an improvement to key aspects of the system is intended.

### 5.4.1 Pattern 12: Expert Validation

*Definition:* A system where a human domain expert validates a critical step to ensure it fulfills certain criteria.

*Problem to be solved:* A need to ensure the reliability of model predictions, especially in scenarios with low error margins.

*Example:* An example of the expert validation pattern can be seen in Figure 5.13, a simple component model depicting ensemble learning with expert validation. Here, the expert validates the consensus checking between the models.

### 5.4.2 Pattern 13: Runtime Evaluation and Improvement

*Definition:* A system that monitors and evaluates the deployed ML model, and makes improvements to it either by retraining, tweaking the model, or creating a new model with some changes.

*Problem to be solved:* A need to avoid model drift and ensure high reliability of the model over time.

*Example:* An e-commerce platform monitors model performance on live recommendation data and re-trains the model weekly using accumulated clickstream feedback to adapt to changing user behavior. It could use the architecture shown in Fig. 5.14, taken from the Amazon Web Services documentation. As seen, a model is trained and deployed, and then it is monitored and adjusted, either by tuning it or by going back to adjust the training data.

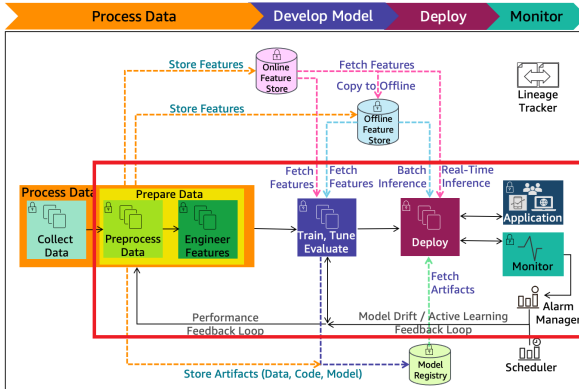


Figure 5.14: Example of the runtime evaluation and improvement pattern taken from [367]

### 5.4.3 Pattern 14: System Evaluation and Improvement

*Definition:* A system that is monitored and evaluated by an ML model and iteratively updated to the system based on the feedback from the model.

*Problem to be solved:* A need to continuously adapt a system in response to changes in its environment, as detected by an ML-based evaluation mechanism.

*Example:* A cloud monitoring service uses an ML model to detect degraded system configurations and suggests architectural adjustments, such as redistributing services across nodes to improve latency, as shown in Figure 5.15.

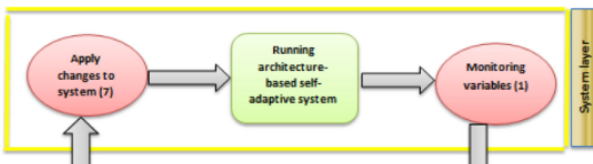


Figure 5.15: Example of the system evaluation and improvement, taken from [368]

# Chapter 6

## Quality Attribute Impacts

In this section, we present the results of our interviews conducted to answer RQ2. Following the methodology described in Section 4, we conducted these interviews to elicit qualitative and quantitative insights from experts on the impact of different patterns on relevant quality attributes. We now present these results for all patterns.

### 6.1 Model-Level Patterns

The impacts of the identified model-level patterns are summarized in Figure 6.1. The motivations for the identified impacts are presented in the following.

#### 6.1.1 Ensemble Learning

All interviewed experts agreed that ensemble learning is a design pattern, according to our chosen definitions of that term. Two experts stated that training each model on a subset of the data will contribute positively to resource efficiency, speeding up training. One of them noticed: *“However, if the task is specialized, one specific model predicts more efficiently”*. Three experts remarked that in terms of reliability, the prediction is more trustworthy when done by multiple models, since if one makes a low-quality prediction, the others can make sure the overall result is correct as well in terms of maintainability, the introduction of more models into the system requires more maintenance efforts. One expert observed: *“When it comes to portability, the system needs to operate with ensemble-learning-supported models compatible with the new hardware or software, which introduces some limitations on the model selection”*. All experts found that in terms of system accuracy, if the models are different but trained on the same data, it is possible to get different nuances of the same information, and thus get a more detailed prediction. Finally, one expert stated: *“Considering fairness, different models provide different views on the same problem, which reduces the way biases propagate throughout the system.”*. Thus, the statements of experts analyzing this pattern did not contradict each other.

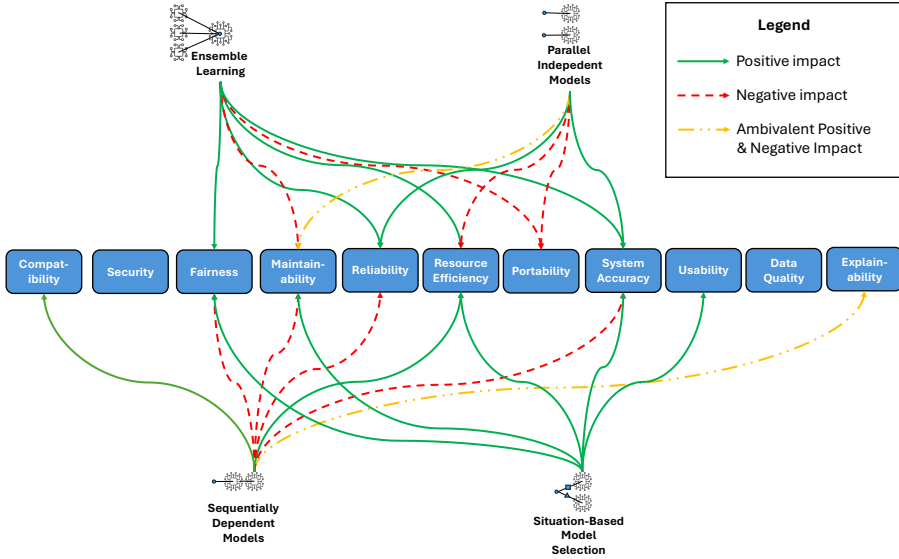


Figure 6.1: Summary of the identified impacts of model-level design patterns on quality attributes

### 6.1.2 Parallel Independent Models

All interviewed experts agreed that parallel independent models are a design pattern, according to our chosen definitions of that term. In terms of resource efficiency, one expert stated: *“Since we introduce more models that need to be run in parallel, there is an increase in the resources required to run the system”*. Two experts observed that in terms of reliability, since the models are independent, there is redundancy in the system, meaning that if one model becomes unavailable, the other models can continue operating as normal. One expert highlighted that, in terms of maintainability, locating faults is much easier with smaller, more specialized models rather than one big model. Further, this expert added *“However, there are more models to maintain, and if we get new data, all the models must be re-trained”*. One expert observed that in terms of portability, all the models would need to be compatible with the new hardware or software, which also introduces certain limitations on model selection. Two other experts stated that in terms of system accuracy, more channels of analysis allow more information to be extracted from the data. Smaller, more specialized models will allow each model to be more accurate within its limited scope than a larger, more general model. Thus, the statements of experts analyzing this pattern did not contradict each other.

### 6.1.3 Sequentially Dependent Models

All interviewed experts agreed that sequentially dependent models are a design pattern, according to our chosen definitions of that term. All experts independ-

ently came to the consensus that, in terms of maintainability, since the models are dependent on each other, there are multiple locations where a problem can stop the whole system and make it more difficult to maintain. One of them added: *“This fact also negatively affects the reliability of the overall system”*. One expert stated that in terms of explainability, if the outputs of each step are more interpretable, the system is easier to understand. However, another expert shared the opposite view: *“It becomes harder to mathematically explain where within the input data the prediction comes from since there are more models and intermediary steps that the input goes through”*. One expert observed that in terms of system accuracy, letting the model decide the steps itself usually results in more precise outputs, while in terms of compatibility, large models can be hard to get to interact with different software and, therefore, this design pattern makes it easier to change the models to interact with other software. Another expert highlighted that in terms of fairness, since the data is processed many times over, it becomes more homogeneous, meaning that it would not represent outliers as well. Thus, the statements of experts analyzing this pattern contradicted each other once, in the case of explainability.

#### 6.1.4 Situation-Based Model Selection

All interviewed experts agreed that situation-based model selection is a design pattern, according to our chosen definitions of that term. One expert summarized: *“The use of this pattern is rational only when the situation is complex and requires large amounts of data. In other cases, a model that handles all situations would probably be better overall”*. Another expert stated that in terms of resource efficiency, since each model only operates in a specific area, it can be smaller and require fewer resources to run. The third expert supported the positive impact of the pattern on resource efficiency and stated: *“We have models requiring only a subset of the data to train on. This means training will be faster and less computationally expensive”*. One expert observed: *“As we can choose different models for different scenarios, we can select the model that best suits the needs of the end-users as a response to usability issues”*. Another expert highlighted that in terms of maintainability, as the functionality for different scenarios is divided between different models, locating the faults in the system will be easier. The third expert supported the positive impact on maintainability by stating that, as the models are independent, they can be worked on independently. Two experts observed that in terms of system accuracy, as each model operates with a smaller input range, it can be more specialized, providing more accurate predictions. Finally, one of them stated that in terms of fairness, as each model operates within a known scenario, it does not have to be biased by factors that do not need to be considered. Thus, the statements of experts analyzing this pattern did not contradict each other; however, in cases of resource efficiency and maintainability, the motivations for the positive impacts varied.

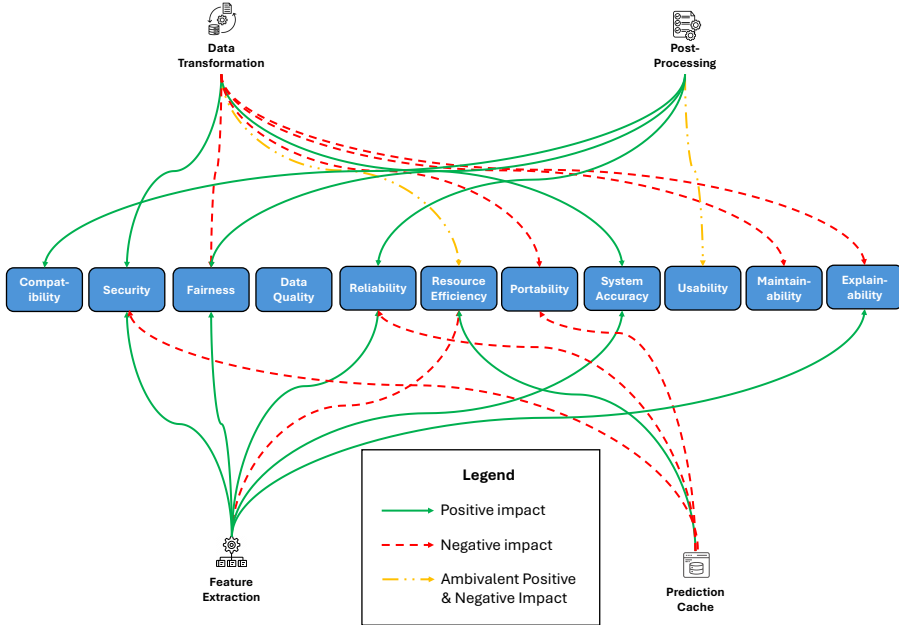


Figure 6.2: Summary of the identified impacts of data-level design patterns on quality attributes

## 6.2 Data-Level Patterns

The impacts of the identified data-level patterns are summarized in Figure 6.2. The motivations for the identified impacts are presented in the following.

### 6.2.1 Data Transformation

All interviewed experts agreed that data transformation is a design pattern, according to our chosen definitions of that term. Two experts remarked that in terms of resource efficiency, as many steps transform the data, there is a computational overhead. One of them stated that in terms of security, when the data is transformed, it becomes unrecognizable from its original state and thus more secure. This expert also observed that in terms of maintainability, as more steps in the processing of the data are added, there is more code to maintain in terms of portability, as this design pattern could need more computational power, porting the functionality requires the system to have higher performance. Another expert highlighted: *“When it comes to explainability, as the data is heavily processed, it can be hard to trace the origins of a prediction”*. This expert also stated that in terms of system accuracy, when data is heavily processed, it tends to be more homogeneous, meaning that the mean prediction may be better. Another expert supported the positive impact and emphasized: *“As data transformation is often a part of complex tasks, it is sometimes not even possible to get a functional model without this type of processing”*. Finally,

one expert stated that in terms of fairness, as we heavily process the data, it tends to go towards the mean, therefore, it may not represent outliers as well. Thus, the statements of experts analyzing this pattern did not contradict each other.

### 6.2.2 Feature Extraction

All interviewed experts agreed that feature extraction is a design pattern, according to our chosen definitions of that term. One expert remarked that in terms of resource efficiency, adding an extra step to the system is associated with some extra computational load. They added: “*Featurization allows the model to be more trustworthy outside of the range of the training data since we are only looking at the relevant features*”, which in turn contributes to the improvement in reliability. However, this expert also highlighted the importance of correct implementation of this design pattern: “*Improper featurisation will make the model less reliable since it will not take all relevant factors into account*”. Both experts remarked that in terms of security, featurizing the data allows to remove sensitive parts so that it becomes untraceable. In terms of explainability, one expert stated that since the score associated with a feature highlights its importance to the model, it will become easier to assign explanations for predictions. In terms of system accuracy, both experts remarked that the model is able to run on only the relevant features, which leads to more appropriate predictions. Finally, one expert stated: “*We can hide biases in the data, allowing the model to be more fair*”. Thus, the statements of experts analyzing this pattern did not contradict each other, however, one expert constantly highlighted the crucial importance of the correct implementation of this design pattern and the severe consequences when it is implemented improperly.

### 6.2.3 Post-processing

All interviewed experts agreed that post-processing is a design pattern, according to our chosen definitions of that term. One expert stated that in terms of usability, the step of post-processing is inherently done to make the prediction from the model usable for the rest of the system and, by extension, the end user. However, another expert provided the opposite view: “*In the process of doing post-processing, data that could be useful for users might inadvertently be hidden*”. One expert remarked that in terms of reliability, during post-processing, the output can be sanity-checked. They also observed that in terms of fairness, classification thresholds can be changed in the post-processing based on demographic attributes, and in terms of compatibility, data can be post-processed to be more usable by other software. Thus, the statements of experts analyzing this pattern contradicted each other once, in the case of explainability.

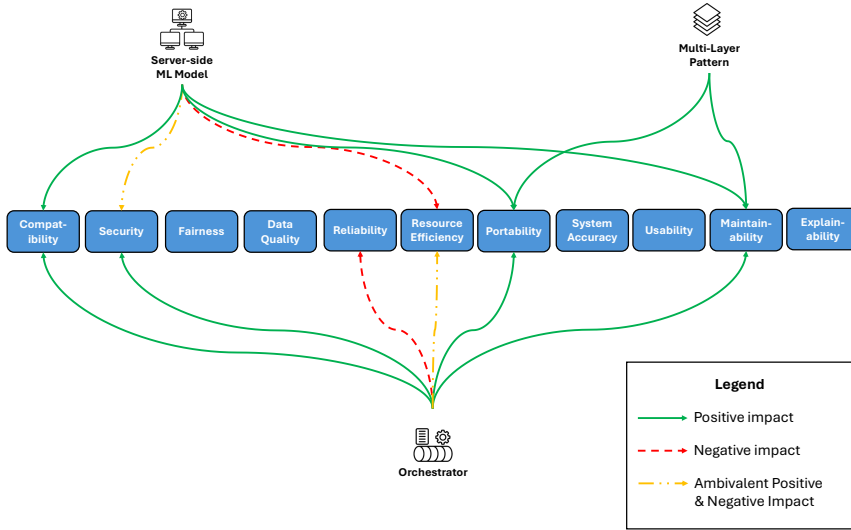


Figure 6.3: Summary of the identified impacts of deployed system-level design patterns on quality attributes

## 6.2.4 Prediction Cache

All interviewed experts agreed that prediction cache is a design pattern, according to our chosen definitions of that term. Both experts remarked that in terms of resource efficiency, when detecting duplicates, the system does not have to perform unnecessary calculations, which saves resources. One of them stated that in terms of reliability, for output uniqueness control, there is a risk for similar inputs to be classified as the same. The expert added: *“data needs to be stored, bringing in privacy and security issues such as physical intrusion and SQL injections”*. Another expert observed that when porting the system, the storage needs to be ported as well, which negatively affects the overall system portability. Thus, the statements of experts analyzing this pattern did not contradict each other.

## 6.3 Deployed System-Level Patterns

The impacts of the identified deployed system-level patterns are summarized in Figure 6.3. The motivations for the identified impacts are presented in the following.

### 6.3.1 Multi-Layer Pattern

All interviewed experts agreed that multi-layer architecture is a design pattern, according to our chosen definitions of that term. Both experts remarked that in terms of maintainability, as the layers are separated, they can be maintained

independently, simplifying the maintenance workflow. One of them added that in terms of portability, since the layers are easier to change, hardware-specific layers can be swapped according to the existing hardware. The statements of experts analyzing this pattern did not contradict each other.

### 6.3.2 Orchestrator

All interviewed experts agreed that an orchestrator is a design pattern, according to our chosen definitions of that term. One expert stated that in terms of resource efficiency, the orchestrator can prioritize resources and balance loads in the runtime, which will improve the performance of the system. However, another expert provided the opposite view: *“As each request has to go through the orchestrator and be routed, there will be a communication overhead”*. One expert remarked: *“As all things connect to the orchestrator, there is a single point of failure, meaning that if the orchestrator has problems, the whole system will”* and considered it as a severe threat to system reliability. Another expert observed that in terms of security, the orchestrator provides layering between the model and the system, allowing for precautionary action and protection. Both experts stated that, in terms of maintainability, the orchestrator separates concerns and allows the system to work with abstractions, meaning the underlying specifics can be changed without the system needing to change. One of them added: *“The orchestrator allows the models to be retrained more easily and on-demand”*. In terms of portability, one expert remarked that since the specifics of the model can be changed more easily, the model can be selected depending on the available hardware as well. Another expert stated: *“When it comes to compatibility, as we decouple the model, we allow other software to interact with abstractions, leading to easier communication”*. Thus, the statements of experts analyzing this pattern contradicted each other once, in the case of resource efficiency.

### 6.3.3 Server-Side ML Model

All interviewed experts agreed that the server-side ML model is a design pattern, according to our chosen definitions of that term. One expert remarked that in terms of resource efficiency, decoupling a system often brings communication overhead, slowing down the system slightly. Two experts stated that in terms of security, since the model is separated from the rest of the system, it is possible to place protections around the model. Further, one of them added: *“However, you should be aware that the communication with the server can be intercepted”*. One expert stated that in terms of maintainability, separating the system and model allows the model to be changed more easily. They also added: *“When we can change the model more easily, we can choose a model that can run on the sought hardware”*, contributing to the improvement of system portability. Finally, both experts remarked that in terms of compatibility, decoupling allows clients to interact with abstractions, leading to easier communication. Thus, the statements of experts analyzing this pattern did not contradict each other. While two experts commented on positive effects, one highlighted an additional

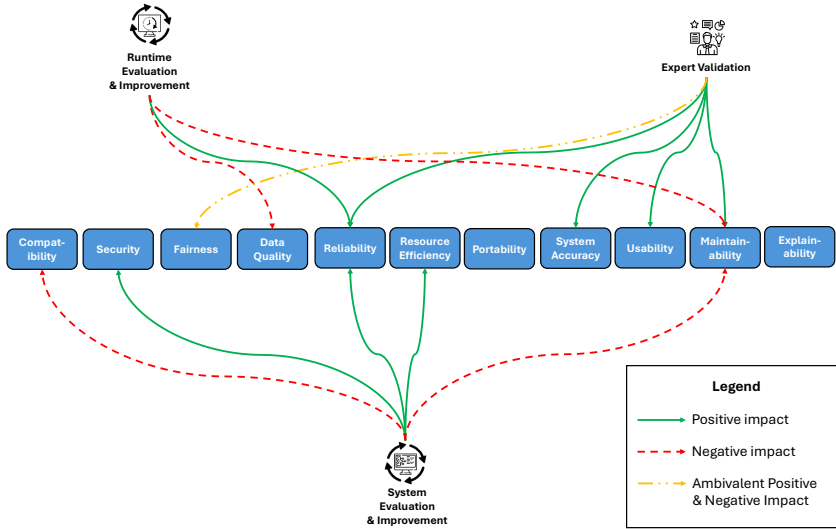


Figure 6.4: Summary of the identified impacts of validation-level design patterns on quality attributes

negative effect connected to security.

## 6.4 Validation-Level Patterns

The impacts of the identified validation-level patterns are summarized in Figure 6.4. The motivations for the identified impacts are presented in the following.

### 6.4.1 Expert Validation

All interviewed experts agreed that expert validation is a design pattern, according to our chosen definitions of that term. One interviewee remarked that since ML-enabled systems are often safety-critical, expert assurance is usable for an end user, contributing to the enhancement of system usability. They also noted that in the context of maintainability, the expert is efficient in detecting if the system does not work as intended. Both interviewees stated that, in terms of reliability and system accuracy, expert validation can ensure that the output is correct, minimizing incorrect predictions. One interviewee stated that in terms of fairness, experts can find if the system does not work as intended and sanity-check the outputs for bias. Further, this interviewee also added that experts could also introduce bias based on their previous experiences. Thus, the statements of experts analyzing this pattern did not contradict each other; however, in the case of fairness, one interviewee provided opposite views on the impact of this design pattern.

## 6.4.2 Runtime Evaluation and Improvement

All interviewed experts agreed that runtime evaluation and improvement is a design pattern, according to our chosen definitions of that term. Two experts remarked that the model gives the best results when the input data is close to the training data, and in terms of reliability, constant monitoring and improvement of the model ensures it is performing optimally, even if the environment changes and the input data starts to drift from the original training data. One expert stated: *“This type of system demands a good architecture since it needs to be able to change models autonomously”*, which poses certain threats to maintainability. In the context of data quality, another expert stated: *“The newest data will always be the most valuable, but there will never be much of it. Therefore, this type of system might need to operate with that data, which could lead to using a small amount of data in the system”*. Thus, the statements of experts analyzing this pattern did not contradict each other.

## 6.4.3 System Evaluation and Improvement

All interviewed experts agreed that system evaluation and improvement is a design pattern, according to our chosen definitions of that term. In terms of resource efficiency, one expert stated: *“As the system is monitored, decisions can be based on resource availability”*, contributing to the improvement of resource efficiency. They also noted that in terms of reliability, the system can adapt if the metrics chosen for evaluation change, allowing the system to adapt to environmental changes, and in terms of security, the system can detect intrusions, allowing for active protection. Another expert observed: *“When we consider maintainability, we need to keep in mind that as the system changes, the evaluation technique might need to do so as well”*, making maintenance more complicated. Finally, another expert stated that in terms of compatibility, when the system can autonomously change, it becomes harder for other software to rely on it. Thus, the statements of experts analyzing this pattern did not contradict each other.

# Chapter 7

## Discussion

In this chapter, we discuss the practical value of results and compare them with related work. We also highlight some important aspects that might be of relevance to the results. Finally, we discuss the various threats to the validity of the study and how this research can be continued in the future.

### 7.1 Practical Usage

The results provided by this work can be used by software architects, who strive to design trustworthy ML-enabled systems, but have no landmarks on how to build an appropriate architecture. With the help of the obtained results, it is possible to select design patterns in accordance with project priorities and key quality characteristics. The identified impacts can serve as architectural smells as well, and warn an architect about the possible consequences of each decision made. For example, if architects operate with limited hardware capacities, they can consider the implementation of an orchestrator to balance loads. At the same time, they will be aware of the potential trade-off with reliability.

Our results can also serve as a basis for further studies by researchers. Based on the results of the interviews, we detected a high interest among experts in investigating ML-enabled software architectures. We believe that the set of design patterns can be further evaluated and potentially extended by using other methodologies and sources.

### 7.2 Pattern Comparison

We now compare our results to the two previous studies on design patterns for AI- and ML-based systems. We find that the overlap with the present study is moderate, with only half of our considered patterns being reported in previous research. This difference can be attributed to the different applied methodologies, with ours being based on an extraction of patterns from available systems.

This study	Heiland et al.	Washizaki et al.
Data transformation	Pipes and filters	
Ensemble learning		
Expert validation		
Feature extraction		
Multi-layer pattern	Multi-layer pattern	
Orchestrator	Mediator pattern	
Parallel independent models	Microservice horizontal pattern	
Post-processing		
Prediction cache	Data cache / Prediction cache	
Runtime evaluation and improvement		
Sequentially dependent models	Microservice vertical pattern	
Server-side ML model	Client-server	
Situation-based model selection		
System evaluation and improvement		
	AI pipelines	
	Asynchronous pattern	
	Daisy architecture	
	Data lake	Data lake
	Data warehouse for ML	
	Distinguish business logic from ML model	Distinguish business logic from ML models
	Fluid architecture	
	Functional-style architecture	
	Handshake	
	Kappa architecture	Kappa architecture
	Lambda architecture	Lambda architecture for ML
	Model-View-Controller (MVC)	
	Parameter-server abstraction	
	Proxy pattern	
	Synchronous pattern	
	Tar pit	
	Workflow pipeline	
		Data flows up, model flows down

Table 7.1: Comparison of patterns found in this work with the ones found by Heiland et al. [345] and the ones by Washizaki et al. [344].

Table 7.1 lists the 14 identified patterns in this work, the 25 architectural design patterns found by Heiland et al. [345], and 5 patterns identified by Washizaki et al. [344], which intersect with the list of Heiland et al. Equivalent or similar patterns are placed on the same row. 7 patterns were identified by both our study and Heiland et al. [345]. There is no overlap between the patterns identified in this paper and the ones described in Washizaki et al.’s work [344]. Many of the patterns in their work, while still being general solutions to commonly occurring problems, can be seen as principles rather than patterns and are therefore not easy to recognize or visualize in component models, for example, *Discard proof of concept code* or *Data flows up, model flows down*.

## 7.3 Threats to Validity

### 7.3.1 Internal Validity

Internal validity threats arise from inappropriate conclusions about cause-and-effect relationships in the data, e.g., because of confounding factors that were not correctly considered [369], [370]. In the case of our study, such threats

could have different natures depending on the multivocal literature review and interviews.

A threat pertaining to the extraction process is the potential impact of subjectivity: the authors' interpretation of the considered component models might have impacted the identification of patterns. To mitigate this threat, each component model was considered independently by two different authors, and extracted patterns were checked for compliance with the definition of design patterns used in this study among the authors. The final list of patterns was agreed on by all authors. The process of negotiations on what is considered a pattern in the current context was done thoroughly and systematically, however, subjectivity or biases can never be completely avoided.

A further possible threat is the lack of appropriate expertise for the specific area in which a particular pattern is relevant. To address this thread, the assignment of patterns to experts was done manually, with the intention of involving experts in the particular area that the pattern affects. For instance, the deployed system-level patterns were only shown to the experts who have a background in software engineering and are more familiar with software architectures.

### **7.3.2 Construct Validity**

Construct validity is concerned with to what extent the measurements are suitable for the study and represent what is investigated [369], [370].

Evaluating design pattern impact on quality attributes is already established to be a complicated task [315], [338], [345] associated with several risks. The chosen methodology for the evaluation was expert assessment in the form of interviews, which induces threats to construct validity. All interviewees have deep academic experience in the investigated domain, and we share responsibility for any judgments about the impacts of patterns on quality attributes with them and rely solely on the assessment of experts. On the one hand, this reduces the bias that could potentially appear from the perception of authors; on the other hand, it over-relies on the experience of the interviewees. Since the interviewees predominantly had an academic background, the results obtained may largely reflect the "academic" perspective on patterns and qualities, making the classification of impacts limitedly reliable for industries. A potential mitigation for this threat could be the involvement of expert practitioners.

### **7.3.3 External Validity**

External validity concerns the generalizability of the study's findings [369], [370]. Given that a component model can be created for each ML-enabled system, exhaustiveness in our literature review is unattainable. The same applies to pattern extraction, as some patterns may have been missed, recorded differently, or appeared only once in our collection. However, since our study's goal is to identify common patterns that improve specific quality attributes, this lack of exhaustiveness does not diminish its value. The study emphasizes patterns with significant impacts on quality attributes, rather than aiming for complete coverage. On the positive side, via our methodology, we only extracted and shared patterns that occur in multiple systems, which ensures

that our patterns are, to some extent, inherently generalizable.

## 7.4 Future Work

There are several directions for future work.

First, our expert-based evaluation is only a first step towards understanding the impact of our identified patterns on system quality. A complementary study could be artifact-driven: researchers can investigate the impact of deploying selected patterns in specific ML-enabled systems, quantitatively measuring how they affect the relevant quality attributes [338]. A more detailed picture of these impacts can also be obtained by distinguishing potential impacts in different types of ML-based systems (e.g., different types of deep learning networks, or application domains), as can be answered either with interviews or with artifact studies.

Second, while our work shows that the extraction of design patterns from ML-based systems is possible, it focuses on high-level patterns at the level of component models. A complementary study based on mined class diagrams could serve to identify design patterns that are closer to the implementation. The same applies to models in particular domain-specific languages, such as the cloud specification languages, which could be an interesting resource in this context: ML-based systems are often deployed on cloud services.

Third, for some of the identified design patterns, we found that the impact on relevant quality attributes is *ambiguous*: the overall impact could be either positive or negative, depending on the circumstances at hand. Giving guidance to developers on when and under which circumstances a particular pattern should be deployed is an important direction for future work. Such guidance could either be made in the form of general guidelines, or by automating it, in the form of a recommender system.

# Chapter 8

## Conclusion

We conducted a multivocal literature review of 754 sources and extracted 49 component models of ML-enabled systems. From these, we derived 14 recurring architectural design patterns, categorized into *model-*, *data-*, *deployed system-*, and *validation-*level patterns. To assess their quality impacts, we conducted expert interviews in which 10 domain experts rated the patterns on 13 quality attributes. This resulted in 71 positive, negative, or ambivalent connections between patterns and QAs. Our findings support practitioners in designing ML-enabled systems more systematically. Documenting recurring patterns, previously underrepresented in ML system design, can improve maintainability and enhance overall system quality.

**MLTradeOps: Embedding Trade-Off  
Management into the MLOps Workflow**

**V. Indykov, D. Strüber, R. Wohlrab**

Proceedings of the 51st Euromicro Conference Series on Software Engineering  
and Advanced Applications (SEAA), 2025, pp. 97–112

# Abstract

The unique nature of machine learning (ML) software systems, characterized by a high level of uncertainty and a crucial dependency on data, has led to challenges for traditional DevOps practices. As a result, a new domain entitled MLOps emerged, which considered these specifics. The evolution of MLOps is aligned with its specialization on certain quality attributes, such as security (*SecMLOps*) or reliability (*SafeMLOps*). However, due to their focus on one exclusive quality characteristic, such frameworks have limited applicability for production aimed at achieving multiple quality objectives at once (e.g., high reliability with the least resources consumed). Explicitly managing the trade-offs between different, potentially competing, quality objectives can help organizations by enhancing the flexibility and predictability of the MLOps workflow.

This vision paper presents a vision around the novel notion of *MLTradeOps*, focused on explicitly managing trade-offs during the MLOps workflow. It brings together the expertise of existing and emerging *DevOps* branches focused on specific quality attributes, and also ongoing monitoring and addressing of other general quality characteristics of in-production software systems. We envision a framework that makes trade-off management a core part of the decision-making process and contains a high-level cycle to make conscious trade-offs for the ML-enabled system, which are then reflected in lower-level decisions during the MLOps lifecycle. We supplement our vision with a roadmap for the potential formation of this framework.

# Chapter 1

## Introduction

Over the last decades, the role of machine learning in industrial software production has become prominent and significant, since ML technologies allow organizations to reach results that are difficult to achieve through traditional solutions [371]. This fact caused the need for significant changes in common DevOps practices, since introducing machine learning into production comes with unique challenges, such as fundamental dependency on data, other acceptable levels of uncertainty for the final results, and specific procedures of model training and testing [5], that could not be optimally addressed by existing DevOps vision. In response to this phenomenon, a new paradigm called *Machine Learning Operations (MLOps)* emerged in 2018 and attracted significant interest from the industry and researchers.

In this vision paper we operate with the broadest definition of *MLOps* given by Kreuzberger et al., who considered it as “a paradigm, including aspects like best practices, sets of concepts, as well as a development culture when it comes to the end-to-end conceptualization, implementation, monitoring, deployment, and scalability of machine learning products” [5]. This definition covers not only the production of ML pipelines but also the production of ML-enabled systems in general. The selection of MLOps interpretation is important for defining its scope and the possible challenges it may address. In other words, MLOps explores not only the behavior and characteristics of the model and data flows but also monitors the decisions and their consequences within the overall system architecture and production workflow.

The decisions can vary on different levels of abstraction and perspectives on the production, starting from the overall MLOps team setting, and ending with a small function added to a certain software component. What they have in common is the fact that every decision (even the most minor one) has the potential to significantly impact the quality of the final product. Such impacts regularly lead to a large set of potential *trade-offs*, since the process of decision-making is usually aligned with sacrificing resources (time, computational power, cost, etc.) to achieve a certain goal [372]. While several trade-offs relate to resource efficiency, some emerge between other quality attributes (e.g., a software architect may choose a less efficient ML algorithm in terms of accuracy

if their primary goal is to introduce a highly explainable model [373]). These *trade-offs* require balancing or finding compromises between several quality attributes.

While trade-offs may appear in ML pipelines (e.g., between resource efficiency and model accuracy [374]), a systematic view of the overall architecture and production workflow of ML-enabled systems opens the horizon of complex trade-off mapping, which is far beyond the ones that appear in ML pipelines in isolation. We noticed that the existing MLOps paradigm includes trade-off management in certain phases [375]; however, it does not make their consideration fundamental for decision-making. It leads to the issue that, in certain cases, the final product is not optimal or even unsatisfactory from the perspective of certain quality characteristics. For example, if the MLOps team never balanced their decisions with ethical considerations, the final product may fail to meet the expectations of stakeholders and society [376].

Our vision makes trade-off management a core part of the decision-making process within the MLOps workflow. We believe that such an approach has the potential to make decisions more informed and conscious by estimating their impacts on quality attributes, and, at the same time, provide extra flexibility in the adaptation of MLOps practices specific to certain quality characteristics by embracing techniques from different quality-oriented MLOps frameworks (such as SecMLOps, SafeMLOps).

The remainder of this vision paper is structured as follows: Section 2 provides an analysis of the DevOps evolution and identifies trends in its development; Section 3 includes our vision as a way to address the issues of existing MLOps paradigm; Section 4 describes a roadmap for the potential formation of the proposed framework; Section 5 includes the discussion of future development and potential limitations of our vision; Section 6 concludes the paper.

Table 1.1: Evolution of Operations in chronological order

<b>Operations</b>	<b>Year of Emergence</b>	<b>Class</b>	<b>Description</b>
DevOps	2007	Fundamental	Optimizing the production of software
DevSecOps	2012	Quality-oriented	Optimizing the security of software in production
DataOps	2014	Quality-oriented	Optimizing data transparency and maintainability within the software in production
AIOps	2016	Methodological	Optimizing processes within DevOps using AI
FinOps	2018	Quality-oriented	Optimizing computing, development, and maintenance costs of software in production (in SE)
MLOps	2018	Fundamental	Optimizing production of ML-enabled systems
ModelOps	2018	Type-oriented	Optimizing governance and life-cycle management of AI models
FLOps	2022	Type-oriented	Optimizing production of Federated Learning models
SecMLOps	2022	Quality-oriented	Optimizing the security of ML-enabled systems in production
LLMOps	2023	Type-oriented	Optimizing production of Large Language Models
SafeMLOps	2024	Quality-oriented	Optimizing the reliability of ML-enabled systems in production
IoTOps	2024	Type-oriented	Optimizing production of IoT systems (incl. devices, deep learning models, etc.)

# Chapter 2

## Background

From the end of the 20th century, more and more companies started to produce software products, following waterfall or semi-agile approaches. The development process had become clearly not optimal as systems became more complex and, as a response to this, a completely new paradigm of *DevOps* was introduced [377] in 2007. Its wide adoption made the delivery of software faster, improved internal and external collaborations, and increased the reliability of the final products.

Over time, this paradigm has evolved and transformed. We collected the most significant milestones of DevOps evolution in Table 1.1. These milestones were divided into 4 classes. The first class is “*fundamental*”, which means that the transformation of DevOps was so significant that it emerged in a completely new paradigm with a set of unique approaches and priorities. The second one is “*quality-oriented*”. This class is characterized by the focus of the existing fundamental paradigm on certain quality attributes of the produced software (e.g., on security or reliability) and its possible extension with quality-specific techniques. The third class is “*type-oriented*”, which is characterized by the focus of the existing fundamental paradigm on a certain type of the produced software (e.g., IoT systems) or type of model in its core (e.g., Large Language Model) and its possible extension with type-specific techniques. Finally, the fourth class is “*methodological*”. It describes the changes in the existing paradigm due to newly emerged unique tools or methods that can be used to optimize the workflow. While there are several interpretations of the scope of each type of operation, we selected the most high-level ones. We observe that DevOps extensions emerged in different contexts: DevOps for security-critical systems (SecDevOps) [378] and the financial side of production, including cloud-based software production (FinOps) [379]. The growth of data volume caused the emergence of DataOps [380], requiring more maintainable and transparent data flows within the systems in production. AIOps came as an extension to DevOps and as a response to emerging AI tools and their potential usage to optimize the production processes [381].

While the year of MLOps’ emergence is debatable, the most widespread version is that it was introduced in 2018 by several Big Tech companies [382].

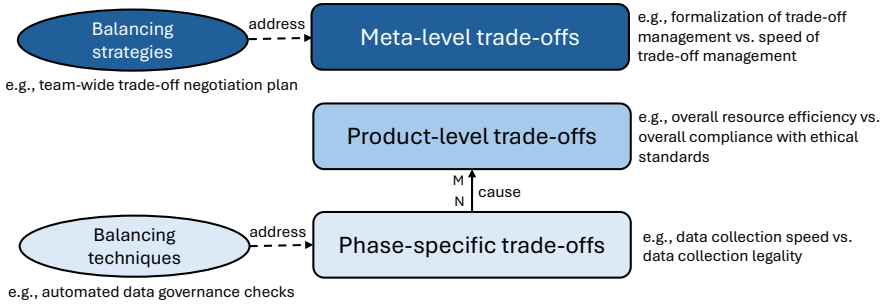


Figure 2.1: Taxonomy of trade-offs in MLOps according to our vision

The first implementations of MLOps dealt primarily with operationalization of model training, data versioning issues, and enhanced communications between data scientists and software engineers. The MLOps paradigm itself continued to evolve and be refined.

While there are several views on the taxonomy of MLOps and ModelOps, the most widely used one is that MLOps was extended by ModelOps, which covers a broader spectrum of operations connected to the AI models and ensures their reliability [383]. Further, different MLOps extensions arose, such as LLMOps [384] and IoTops [385]. We are also witnessing the emergence of SecMLOps [386], which is the combination of SecDevOps and MLOps principles, SafeMLOps [387] prioritizing the reliability over other system qualities.

Summing up this investigation, we can state that DevOps and MLOps were the “game-changers” in the area of software production. Other types of *DevOps* are serving as their extensions to specify or expand the use of these 2 frameworks in different contexts:

1. Based on the specific issues that arise from prioritized quality attributes (SecDevOps, SecMLOps, SafeMLOps, FinOps)
2. Based on the specific issues that arise from different types of algorithms or models lying in the core of the systems in production (LLMOps, IoTops)
3. Based on the potential to optimize the existing paradigm with new, unique methods and tools (AIOps).

While quality-oriented extensions arise dynamically, providing useful insights on how to achieve specific quality objectives, none of them is optimized for achieving multiple quality objectives at the same time. Given the increasing complexity of software being produced and the dynamic growth of competition in the market, pursuing multiple quality goals is becoming an important vector of industry development, while prioritizing only one quality attribute in isolation from others may lead to non-optimal system performance, user dissatisfaction, or even critical vulnerabilities [388].

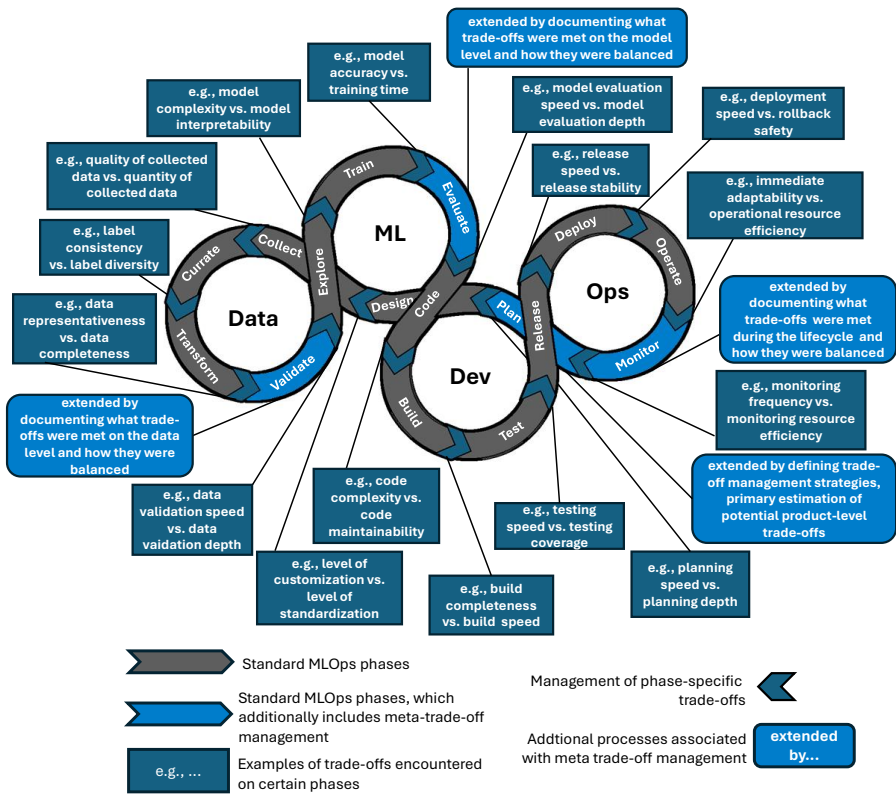


Figure 2.2: Overview of MLTradeOps

# Chapter 3

## Vision

The core vision of *MLTradeOps* is to consider quality attributes in a holistic way, raising awareness of trade-offs and supporting MLOps teams in making principled, quality-oriented decisions. We propose to aggregate best practices from existing quality-oriented MLOps extensions, e.g., security-focused practices from SecMLOps, reliability-centered ones from SafeMLOps, methods of resource analysis from FinOps, as well as from emerging ones that are not fully formed yet but provide first promising solutions, e.g., based on environmental considerations in GreenMLOps. Quality attributes may generally conflict with each other. Therefore, to support decision-making, we suggest the development of a coherent framework that makes multi-level trade-off management a first-class citizen.

The proposed framework would enable industries to utilize suitable techniques from various quality-oriented types of MLOps, thereby achieving multiple quality goals simultaneously by striking a balance among them. For example, if a company is required to deliver security-critical products, but at the same time has to ensure high levels of resource efficiency, monitoring and managing this trade-off becomes essential at each phase of the MLOps life cycle. Moreover, other quality characteristics - such as fairness, reliability, functional suitability - may need to be satisfied at least at a minimum acceptable level before the product is delivered. In existing MLOps workflows, mapping these qualities and trade-offs can become overly complicated.

We now explain our vision in detail, first introducing a taxonomy of trade-offs, and then suggesting activities for addressing them. Together, this leads to a framework of *MLTradeOps* building on the MLOps paradigm as a basis. According to the selected definition [5], MLOps covers best practices, sets of concepts, and developing culture in four main areas: data management, machine learning model production, system-level production, and operations associated with the deployed product.

**Taxonomy of trade-offs.** Our vision distinguishes three types of trade-offs, shown in the overview in Fig. 2.1: (i) *Phase-specific trade-offs*, emerging during a specific MLOps phase and involving low-level quality attributes associated with it (e.g., for the phase of data validation: data validation speed vs. data

validation depth). These trade-offs can be balanced with certain balancing techniques (e.g., probabilistic data validation); (ii.) *Product-level trade-offs*, emerging at the level of the overall product and being the consequences of one or several phase-specific quality trade-offs (e.g., product resource efficiency vs. product security). Such trade-offs could not be solved in isolation from the phase-specific ones that caused them. Thus, product-level trade-offs serve primarily as signals for MLOps teams to take action. According to our vision, the only way to balance product-level trade-offs is to balance phase-specific one(s) that caused it; (iii.) *Meta-level trade-offs*, emerging on the meta-level of trade-off management. They involve teams' strategic goals in organizing trade-off management workflow (e.g., level of formalization of trade-off management vs. speed of trade-off management).

Addressing these trade-offs leads to a comprehensive set of activities that augment the regular MLOps process, shown graphically in Fig. 2.2. It adds new proposed steps to the standard MLOps schema and updates certain phases with extra logic. In our vision, the end of each phase of the MLOps workflow should be supplemented by a mandatory step of phase-specific trade-off management, while the phases of "*planning*", "*data validation*", "*model evaluation*" and "*monitoring*", must be updated with the additional strategies for meta trade-off management.

**Balancing meta-level trade-offs.** According to our vision, meta trade-off management should be conducted in addition to standard MLOps procedures at the phase of "*planning*", and include mapping of the main quality priorities for the delivered product(s), trade-offs that may potentially be encountered further, and high-level approaches for balancing them. Quality priorities should be selected from a predefined catalog of quality attributes supported by the framework. The selection of priorities for the specific product should be conducted according to stakeholders' needs, governmental policies, and the developers' mission. Together with such priorities, the team has to define a minimal threshold for other quality attributes commonly used in the domain, which are proposed by *MLTradeOps* as well. At this phase, the team analyzes meta-level trade-offs and selects the overall workflow of trade-off management.

**Balancing phase-specific trade-offs.** Further, the team should estimate what main trade-offs it may encounter at different phases of the MLOps lifecycle. While the most widespread trade-offs will be described by *MLTradeOps*, teams will never be able to estimate all the possible cases on each step of the MLOps workflow in advance [375]. Therefore, the need for regular adjustments at the end of each phase arises, which is addressed by the phase-specific trade-off management.

The phase-specific trade-off management deals with the trade-offs arising from the decisions made on a certain phase and level of abstraction. For example, decisions made on the phase of data curation may cause trade-offs between data quality vs. resource efficiency (e.g., if the datasets are big but noisy, they require significant resources to be cleansed [392]), data relevance vs. data stability (the most up-to-date data can improve model relevance, but at the same time can lead to data drift [393]).

These trade-offs can be managed and balanced by applying certain tech-

Table 3.1: Examples of potential product-level trade-offs caused by phase-specific ones

<b>Phase</b>	<b>Phase-specific trade-off</b>	<b>Product-level trade-off</b>	<b>Description</b>
Model Training	Model Accuracy vs. Training Time	Product Reliability vs. Product Resource Efficiency	Model training requires significant resources. If the model lies at the core of the product, the lack of resources invested in its training may cause issues with overall product reliability [389].
Coding	Code Complexity vs. Code Maintainability	Product Resource Efficiency vs. Product Maintainability	The proper and clear structuring of code requires efforts, however, when those efforts are insufficient, they lead to increased code complexity, reduced readability, and higher technical debt, which in turn reduces overall product maintainability [390].
Data Transformation	Data Representativeness vs. Data Completeness	Product Accuracy vs. Product Fairness	When the data is transformed to reflect the target distribution for the model (data representativeness), the performance of the algorithms that use it can be higher than when it operates with rawer data (data completeness). At the same time, the algorithms may further ignore rare cases or anomalies that affect minority users, decreasing the fairness of the overall product [391].

niques [33], [394]. Such techniques are different for each trade-off as well as for each phase and level of abstraction. The *MLTradeOps* framework includes the most common ones.

**Balancing product-level trade-offs.** Decisions made at a given phase typically result in trade-offs between phase-level quality attributes; however, they may later lead to product-level trade-offs. For example, for the phase of model evaluation, the trade-off may occur between evaluation speed and evaluation depth (basic evaluation makes the phase quick; however, deeper evaluation provides more reliable performance estimates [395]). This trade-off appears exclusively on the model level at the evaluation phase, however, it may later cause a system-level trade-off between overall resource efficiency (how many resources were invested in the product in total) and overall reliability (how reliable the outputs of the product are). In Table 3.1, we take examples of phase-specific trade-offs from Fig. 2.2 and suggest which product-level trade-offs they may lead to.

According to our vision, the only way to balance product-level trade-offs is to balance phase-specific ones that caused them. On the phases of “data validation”, “model evaluation” and “operational monitoring”, in addition to standard MLOps procedures, the connection of certain phase-specific trade-offs, encountered in the associated layers (model layer, data layer, operational layer, respectively), to product-level ones should be investigated. We propose to add extra meta trade-off management activities associated with documenting the applied balancing techniques and their outcomes retrospectively. In parallel, management at these phases allows teams to re-balance meta-level trade-offs and adjust their strategies accordingly (e.g., to increase participation of stakeholders).

In this section, we provided the vision of how our framework may be used by MLOps teams. In the following section, we provide our vision on the possible formation of such a framework and the main artifacts it should contain.

# Chapter 4

## Road map

We present a roadmap for the formation of MLTradeOps, showing an overview in Table 4.1. This roadmap covers its entire development from initial conceptualization, systematic collection of necessary content, validation of the framework through production testing, and logic for regular updates. This structured process ensures that the framework is both practical and adaptable to the changing needs of ML-enabled systems in production environments.

*MLTradeOps*, like other quality-oriented MLOps frameworks, will be implemented as a set of best practices oriented on enhancing the quality of the delivered software products and associated workflow. While the application of *MLTradeOps* in the processes within companies is supposed to be fully agile-based, the formation of the framework follows more sequential logics and implies different interconnected steps of data collection. One of the main sources of data for its formation should be the most recent experience from the successful mature companies. The main attention should be paid to companies that successfully implemented certain quality-oriented MLOps extensions.

Table 4.1: Roadmap for the implementation of MLTradeOps.

Formal Step	Description
<b>Step 1:</b> Planning the embedding of the framework into the existing MLOps paradigm.	This step is associated with the conceptual design of the <i>MLTradeOps</i> framework and includes a deep investigation of existing MLOps implementations and theoretical integration of the framework into the MLOps paradigm with analysis of potential inconsistencies and conflicts.  If the designed <i>MLTradeOps</i> concepts contradict existing MLOps principles, the designers should revise the framework to avoid this conflict or provide a solid analysis of the possible consequences of such a contradiction on the overall MLOps workflow.

Continued on next page

**Table 4.1 – continued from previous page**

First column	Second column
<p><b>Step 2:</b> Collecting the most common product-level quality attributes of ML-enabled solutions</p>	<p>Product-level quality attributes describe the overall quality of the software in production. Additionally, they can consider the quality of the overall workflow, excluding the process associated with trade-off management.</p> <p><i>Examples of the product-level quality attributes:</i> resource efficiency, reliability, security, maintainability.</p>
<p><b>Step 3:</b> Collecting the most common trade-offs encountered on the product level in the production of ML-enabled solutions.</p>	<p>Product-level trade-offs are caused by one or several phase-specific trade-offs and affect the overall quality of delivered software.</p> <p><i>Examples of the product-level trade-offs:</i> resource efficiency vs. reliability; system accuracy vs. system explainability; maintainability vs. security.</p>
<p><b>Step 4:</b> Collecting the most common meta-level trade-offs.</p>	<p>Meta-level trade-offs describe possible dilemmas that arise when organizing the processes of trade-off management.</p> <p><i>Examples of the meta-level trade-offs:</i> level of formalization of trade-off management vs. speed of trade-off management; degree of stakeholder involvement vs. speed of trade-off management.</p>
<p><b>Step 5:</b> Collecting the most common meta trade-off balancing strategies.</p>	<p>Meta trade-off management strategies imply high-level decisions associated with the workflow and serve to address meta-level trade-offs.</p> <p><i>Examples of the balancing strategies:</i> team-wide trade-off negotiation plan, methods of multi-objective optimization, stakeholder-driven prioritization.</p>
<p><b>Step 6:</b> Collecting the most common quality attributes specific to all standard MLOps phases.</p>	<p>Phase-specific quality attributes describe the quality of processes within different MLOps phases. While the collection of all possible phase-specific quality attributes is unfeasible due to the unique architecture of each product, the list of attributes provided by the framework must operate with the most widely applicable ones.</p> <p><i>Examples of the phase-specific quality attributes:</i> code complexity for coding phase; environment consistency for deploying phase; training speed for model training phase.</p>
<p><b>Step 7:</b> Collecting the most common trade-offs that emerge between phase-specific quality attributes.</p>	<p>Phase-specific trade-offs come as consequences of the decisions made on certain phases and levels of abstraction. However, they may later evolve into dilemmas between product-level quality attributes. The framework must explicitly describe these connections.</p> <p><i>Examples of the phase-specific trade-offs:</i> for the data collection phase: data collection speed vs. data collection legality (e.g., ensuring that data is collected strictly according to GDPR, HIPAA principles may sufficiently reduce the speed of data collection); for the design phase, the trade-offs may arise between phase-specific attributes: design scalability vs. design simplicity, which may later lead to a trade-off between product-level quality attributes: product maintainability vs. product reliability.</p>
<p><b>Step 8:</b> Collecting the most common techniques to balance phase-specific trade-offs.</p>	<p>These techniques can be applied on different levels of abstraction according to the phase with which they are associated.</p> <p><i>Examples of the balancing techniques:</i> data anonymization or automated data governance checks to deal with data collection speed vs. data collection legality; modularization or use of lightweight abstractions to balance design scalability vs. design simplicity.</p>

Continued on next page

Table 4.1 – continued from previous page

First column	Second column
<b>Step 9:</b> Assembling and formalizing knowledge into a single coherent framework.	Once all the best practices have been collected, they need to be combined into one coherent structure. At this stage, the framework is documented and formalized. An important task at this step is to ensure that all of the identified product-level trade-offs are connected to certain phase-specific ones.
<b>Step 10:</b> Testing the framework in industrial settings and conducting necessary updates.	Before the framework is presented to the public, it must go through wide testing in industrial settings. To obtain fair testing, collaboration with industrial partners of different sizes, domains, and maturity levels is required. If the results of testing require any changes in the designed framework, the updates are conducted accordingly at the appropriate step.
<b>Step 11:</b> Promoting the framework for industrial use.	On this step, the marketing strategies for framework distribution are designed and followed.
<b>Step 12:</b> Initiation of framework updates and return to <i>Step 1</i> .	The updates of the framework should be triggered by a certain event, which can be, for example, the end of a certain period (e.g., the framework remains up-to-date only for 1 year) or the emergence of new ML implementations. Since the development of new implementations is currently quite rapid, it is necessary to constantly monitor their emergence. However, before initiating updates and including the experience of such implementations into MLTradeOps, it is necessary to conduct certain checks, including checking the scope of implementation, the methodologies by which it was formed, the level of assessment of practices in industrial conditions to evaluate if the content of certain implementation corresponds to the requirements of MLTradeOps content. As soon as the time for updates comes, the process comes back to Step 1.

The formation of the framework starts with the planning step (*Step 1*). While this paper describes our high-level vision, Step 1 must include a deep investigation of potential conflicts and inconsistencies with the existing MLOps paradigm. The resource planning must also be included. The primary strategy involves a close collaboration between universities and industries. The academic environment promotes a systematic approach to the framework formation and consideration of existing scientific knowledge, while the industry is a provider of the experience that forms the main part of the framework content.

All the content included in *MLTradeOps* (including collected quality attributes, trade-offs, balancing strategies, and techniques) must satisfy the following criteria and be:

1. *Common*, i.e., combining the most widespread knowledge among different industries.
2. *Unified*, i.e., relevant for all types of delivered ML-enabled products (e.g., LLM-based software, deep learning systems).
3. *Up-to-Date*, i.e., regularly updated according to the recent developments of the industry.

The road map is cyclical. This was made to ensure that “up-to-date” criteria are satisfied and that the framework includes the logic for its constant updates. While the formal trigger for such updates will be defined on *Step 1*, since such a decision requires deep research into the dynamics of change in overall MLOps, we see a need for conducting such updates regularly. In parallel, we acknowledge the fact that balancing techniques may evolve much faster than balancing strategies, and possible phase-specific trade-offs may require specification and review more frequently than at the product or meta level. As a result, the framework updating process must be flexible and include only the necessary steps.

**Product-level trade-offs.** Once the planning step is complete, the collection of common quality attributes at the product level begins. The studies conducted previously by the authors of this vision proposed a common quality model for ML-enabled systems based on the results of a systematic literature review and the evaluation of findings in industrial settings [371]. To be included in the framework, this quality model requires a broader evaluation within larger and more mature companies. The consideration of existing studies [27], standards, such as ISO-25059 [73], and, most importantly, documented experience of a wide range of industries is also required at this step.

Once the list of product-level quality attributes is formed, the collection of the most common trade-offs encountered on the product level begins. Previously, we conducted a study that collected trade-offs met by different industries on a product-level [316]. We believe that the methodology proposed by that study may be scaled and embrace the experience of a wider range of companies to be included in *MLTradeOps*.

**Meta-level trade-offs.** Step 4 involves identifying the most common meta-level trade-offs associated exclusively with the workflow of trade-off management, including the identification of the most widespread strategic goals and priorities.

Further, the most common strategies used by organizations to manage trade-offs on the meta level are identified. These high-level strategies serve as guiding principles for navigating quality concerns across the MLOps life cycle and are selected accordingly to balance meta-level trade-offs. To ensure wide applicability, only the strategies that have proven efficiency across multiple domains and organizational maturity levels must be considered for inclusion in the framework. The strategies must be augmented with potential use cases, constraints, and maturity indicators to guide their adoption.

**Phase-specific trade-offs.** Once the high-level perspective is captured, the framework proceeds to collect quality attributes specific to individual MLOps phases. These phase-specific quality attributes provide a more granular view of quality and reflect the unique demands of each phase. Each attribute is defined in the context of its operational environment. While the exhaustive collection of all possible attributes is infeasible due to architectural variation across products, the framework focuses on identifying the most widely applicable ones through literature synthesis and industrial feedback.

After identifying the relevant phase-specific quality attributes, the investigation of the trade-offs that commonly emerge between them is conducted. These

trade-offs typically emerge when optimizing one phase-specific attribute leads to the degradation of another. Initially localized in specific MLOps phases, such trade-offs can escalate to product-level ones. Therefore, understanding the cascading nature of these trade-offs is essential. The framework explicitly traces the evolution of phase-specific decisions into product-wide implications, thus promoting more holistic trade-off management.

To navigate these localized conflicts, the framework includes a catalog of balancing techniques tailored to specific phases. Phase-specific techniques are more granular and actionable than the balancing strategies identified previously, but like them, are annotated with use cases, constraints, and maturity indicators.

**Validation and Dissemination.** Once the fundamental elements (quality attributes, trade-offs, strategies, and techniques) are collected, assembling and formalizing this knowledge into a single coherent framework begins. At this stage, the framework from a conceptual collection of insights evolves into a structured and navigable model that can be applied in real-world settings. The formalization process includes defining relationships between the different elements, establishing a consistent taxonomy, and ensuring terminological clarity across components.

Following framework formalization, Step 10 involves rigorous testing in industrial settings. This validation phase is critical to ensure that the framework performs effectively across varying levels of organizational maturity, domain specificity, and scale of ML integration.

Once validated, Step 11 is dedicated to the promotion and dissemination of the framework for industrial use. This includes designing targeted marketing strategies, planning the presentations at diverse seminars and conferences, and developing scenarios of direct communications with the companies.

The roadmap represents our high-level vision of a possible framework formation. It can be updated and specified in the process, however, it is supposed to be used as the basis for further work. In the following section, we discuss potential limitations associated with our vision and with the proposed roadmap in particular.

# Chapter 5

## Discussion

**Embedding into evolution of DevOps and MLOps.** Based on the analysis of past DevOps development, we see several potential scenarios for its further evolution. Our vision could contribute to each of them:

*Scenario 1.* MLOps may evolve into new directions, such as the following three: First, extensions specialized in newly emerged types of ML models (e.g., self-optimizing models with a framework that could be called SOMLOps). Regular updates to our framework allow us to consider the specifics of such models and analyze the unique trade-offs they may lead to. Second, extensions specialized in quality attributes of emerging importance (e.g., GreenMLOps focused on sustainability). Experience from such extensions can be augmented by the consideration of other quality attributes, proposed by *MLTradeOps*.

*Scenario 2.* New DevOps extensions, not necessarily connected to ML, may appear (e.g., DevOps focused on ethical considerations - EthicOps). Fundamental concepts of *MLTradeOps* are flexible and may cover trade-off management of non-ML-enabled products by analyzing trade-offs specific to them.

*Scenario 3.* There may be another “game changer” that will fundamentally shift the paradigm of DevOps, like it was with the emergence of MLOps: e.g., Artificial General Intelligence with GIOps. We believe that new paradigms inevitably come with trade-offs. While the trade-offs and associated balancing strategies may differ, the need for their management remain relevant.

The vision at the core of our framework allows different teams to operate in the dimensions of several quality priorities, mitigating the need for new separate quality-oriented extensions of MLOps, but provoking the need to constantly update the proposed framework with the most recent knowledge in the industry. As a result, by applying *MLTradeOps*, the overall quality of delivered products increases, and the workflow becomes more predictable.

**Justification of assumptions.** Our overall vision is based on a foundational assumption that product-level trade-offs arise as consequences of certain MLOps phase-specific ones. According to the Architecture Tradeoff Analysis Method (ATAM) [396], the analysis of trade-offs within the product is directly connected to the investigation of the trade-off points (parameters in the architecture to which several measurable quality attribute responses are highly correlated).

Such points emerge as the result of decisions made on different levels of abstraction and different phases of the MLOps lifecycle. ATAM is compatible with our vision and can be leveraged to analyze cause-effect relationships and balance phase-specific trade-offs at the end of each phase.

**Challenges.** The execution of our vision may lead to a number of challenges. Considering the proposed road map, some steps take inspiration from the studies that rely on literature and limited-scale industrial evaluation. This fact may omit factors that only emerge at broader operational scales. Therefore, future iterations should include multiple independent methodological validations of each step, ideally performed by external researchers or practitioners not involved in the original design of this road map.

Our framework is supposed to operate with the most common experience, which means widely used practices in companies of different domains, sizes, and maturity levels. This introduces a threat that the content of the framework relies excessively on the chosen definition of such "commonality". To mitigate this, the analysis of how existing MLOps frameworks were formed methodologically should be done, and the formation of *MLTradeOps* must be aligned with that.

Although the framework is designed to be unified and common across various industries, differences in organizational maturity, team structures, regulatory environments, and ML deployment pipelines may be too significant. As a result, the applicability of the framework in certain contexts may be limited. For example, the most widespread quality trade-offs in heavily regulated domains (e.g., healthcare, autonomous vehicles) may differ from more flexible ones (e.g., marketing, retail sectors). Additionally, organizations at the early stages of MLOps adoption may find the full framework overwhelming or not directly applicable. Therefore, we see the need to customize our framework based on organizational maturity and domain-specific extensions where appropriate, in the future.

# Chapter 6

## Conclusions

This vision paper provides our view on how the existing MLOps paradigm may be enhanced to obtain high-quality production in terms of several quality characteristics, while maintaining satisfactory levels of quality for the rest common quality attributes. The main difference of our framework from the existing MLOps paradigm is fundamental embedding of trade-off management on two levels: phase-specific ones, where certain balancing techniques are applied, and meta-level, where overall balancing strategies are defined, and applied decisions are documented and analyzed.