KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Engineering Software Systems with Self-Adaptation and Machine Learning

Federico Quin

Supervisors:
Prof. dr. Daniel Weyns
Prof. dr. Daniel Hughes

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

October 2023

Engineering Software Systems with Self-Adaptation and Machine Learning

Federico QUIN

Examination committee:
Prof. dr. ir. Paul Sas, chair
Prof. dr. Daniel Weyns, supervisor
Prof. dr. Daniel Hughes, supervisor
Prof. dr. ir. Hendrik Blockeel
Prof. dr. Patrick Decausmaecker
Dr. Sam Michiels
Prof. dr. ir. Jan Goedgebeur
Prof. dr. Ada Diaconescu
(Institut Polytechnique de Paris)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

October 2023

Uitgegeven in eigen beheer, Federico Quin, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Preface

The completion of this dissertation has been a rewarding journey, made possible through the support and guidance of numerous individuals who have played important roles in my academic pursuits.

First and foremost, I would like to express my deepest appreciation to Danny Weyns, my mentor, for his continuous encouragement, unbounded passion, profound expertise, and patient guidance throughout this research endeavor. Your belief in my capabilities has been a driving force behind my academic achievements.

I extend my heartfelt gratitude to my dissertation committee, Sam Michiels, Danny Hughes, Patrick De Causmaecker, Hendrik Blokeel, Jan Goedgebeur, and Ada Diaconescu, for their dedication, invaluable insights, and mentorship. Their collective wisdom has been instrumental in shaping my academic growth and the successful completion of this dissertation. I would also like to thank Paul Sas for chairing the examination committee.

I would also like to acknowledge my colleagues and friends, who have provided unwavering support, encouragement, and understanding during this challenging yet rewarding pursuit. I explicitly want to thank the following people: Kristof Jannes for always providing me with an (involuntary) ear to discuss any topic at any time; Pieter-Jan Vrielynck for his warm and caring presence in the office; Emad Heydari Beni for always creating an uplifting and funny environment; Michiel Provoost and Omid Gheibi for both fruitful collaborations and pleasant conversations; Bert Lagaisse for the frequent coffee breaks with some much-needed distractions; and Nima Rahimi Foroushani, Justus Fasse, Tobias Reinhard, Niels Mommen, Dorde Markovic, Denis Carnier for their open and wonderful friend group.

On top of this, I would also like to thank the following people, in no particular order: Laurens Sion, Martijn Sauwens, Koen Jacobs (MoV), Majid Salehi, Toon Dehaene, Dimitri Van Landuyt, Koen Yskout, Amin Timany, Sepideh Pouyanrad, Giuseppe Garofalo, Jan Vermaelen, Brendan Mackenzie, Victor Le Pochat, Jan Vanhoof, Weihong Wang, Yana Dimova, Gertjan Franken, and many others. To all those whose names may not appear here but whose encouragement, advice, and friendship have played a significant role in my academic journey, I offer my heartfelt thanks.

I would also like to thank my family for their unconditional love and support. I am especially grateful to my parents, who have always encouraged me to pursue my dreams and have done everything in their power to provide me with all possible opportunities to find happiness in my life.

Lastly, I would like to express my deepest gratitude to my girlfriend Jing Xu, for being there for me during the most challenging times of my academic journey. Your love, support, and encouragement have been a constant source of strength and inspiration.

Thank you!

– Federico Quin

Abstract

Modern software systems are often deployed in dynamic and uncertain environments, where the operating conditions of the system are difficult to predict before the system is in operation. Not attending to these uncertainties may jeopardize the system's goals. To mitigate such uncertainties, self-adaptation presents a consolidated approach that adapts software systems to changing operating conditions. With the increasing complexity of modern software systems and their environments, it becomes essential to engineer self-adaptive systems that can deal with this complexity in an efficient manner. A promising technique that has gained a significant amount of attention in supporting self-adaptation that can address complexity and scalability concerns is machine learning. It has seen widespread use, from supporting analysis of adaptation options in self-adaptive systems to fully encompassing adaptation logic.

In this thesis, we investigate how self-adaptation supported by machine learning can be used to engineer systems from two complementary perspectives. In the first perspective, we contribute a systematic literature review on the use of machine learning in self-adaptive systems. From this literature review we identified a gap in the literature on systematically dealing with large adaptation spaces that complements model checking techniques. To address this gap, we contribute an architecture-based solution that leverages machine learning in conjunction with statistical model checking to efficiently reduce large adaptation spaces. In the second perspective, we contribute a systematic literature review on A/B testing with a focus on engineering aspects. From this literature review we identified a need for the automation of A/B testing, alongside the need for efficient execution of A/B tests. To address these needs, we contribute an approach that leverages machine learning and self-adaptation to automate the efficient execution of pipelines of A/B tests.

Beknopte samenvatting

Moderne softwaresystemen worden vaak ingezet in dynamische en onzekere omgevingen, waarbij de operationele omstandigheden van het systeem moeilijk te voorspellen zijn voordat het systeem in werking treedt. Wanneer geen rekening gehouden wordt met deze onzekerheden kunnen de doelstellingen van het systeem in gevaar gebracht worden. Om dergelijke onzekerheden aan te pakken, biedt zelf-adaptiviteit een geconsolideerde aanpak die software systemen aanpast aan veranderende operationele omstandigheden. Met de toenemende complexiteit van moderne softwaresystemen en hun omgevingen wordt het essentieel om zelf-adaptieve systemen te ontwikkelen die op een efficiënte manier met deze complexiteit kunnen omgaan. Een veelbelovende techniek die subtantiële aandacht heeft gekregen in het ondersteunen van zelf-adaptiviteit, en die complexiteits- en schaalbaarheidsproblemen kan aanpakken, is machine learning. Het wordt veelvuldig gebruikt, van het ondersteunen van de analyse van adaptatie-opties in zelf-adaptieve systemen tot het volledig omvatten van adaptiviteitsslogica.

In dit proefschrift onderzoeken we hoe zelf-adaptiveit ondersteund door machine learning kan worden gebruikt om systemen te ontwikkelen vanuit twee complementaire perspectieven. In het eerste perspectief dragen we bij met een systematische literatuurstudie over het gebruik van machine learning in zelf-adaptieve systemen. Uit deze literatuurstudie hebben we een lacune in de literatuur geïdentificeerd met betrekking tot het systematisch omgaan met grote adaptatie ruimtes die model checking technieken aanvullen. Om deze lacune aan te pakken, dragen we een architectuur-gebaseerde oplossing bij die machine learning combineert met statistische model checking om grote adaptatie ruimtes efficiënt te verminderen. In het tweede perspectief dragen we bij met een systematische literatuurstudie over A/B-testen met een focus op ingenieursaspecten. Uit deze literatuurstudie identificeerden we de behoefte aan automatisering van A/B-testen, samen met de behoefte aan efficiënte uitvoering van A/B-tests. Om aan deze behoeften tegemoet te komen, dragen we een aanpak bij die machine learning en zelf-adaptiveit benut om de efficiënte uitvoering van reeksen A/B-testen te automatiseren.

Contents

Chapter 1

Introduction

In the last few decades, technology has pushed society forward in ways that were initially hard to imagine. Technology is advancing rapidly, and the use of technology in people's everyday life has become increasingly more common. Alongside the growing advancement of technology, software systems are becoming increasingly complex. An example of this is the rapid growth in the domain of the Internet-of-Things [190, 157]. A popular example can be found in modern home appliances such as refrigerators, washing machines, and thermostats. These devices[1] are increasingly connected to the Internet, providing remote-control functionality to monitor the devices and change the settings of the devices. The capabilities of such devices are constantly growing, e.g., Samsung providing among-device artificial intelligence capabilities within households [166].

Alongside the increasing complexity and capabilities of modern software systems, the environments in which these systems operate are also becoming increasingly more complex. Software systems are often deployed in dynamic and uncertain environments, where the operating conditions of the system are difficult to predict before the system is in operation. Not attending to these uncertainties may jeopardize the system's goals. For example, network interference may affect the availability of the system if not properly dealt with. To mitigate such uncertainties, self-adaptation presents a consolidated approach that adapts modern software systems to changing operating conditions [64, 301, 361].

A foundational work for the field of self-adaptation was presented by Kephart and Chess. They introduced their vision of autonomic computing [193], motivated by the growing complexity of engineering software systems. The authors argued that systems should

[1] Popularly dubbed as smart devices

be able to autonomously, i.e., without intervention, manage themselves given a set of high-level objectives, denoted as *self-management*. The authors further argued that self-management should be realized through a feedback loop that relieves the system's operators from the burden of managing the system. MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) is a commonly used pattern to realize such a feedback loop [64, 223, 376]. The MAPE-K pattern consists of four phases: (1) *Monitor* the system and its environment, (2) *Analyze* the monitored data and options to adapt the system, (3) *Plan* the adaptation, and (4) *Execute* the adaptation plan. The *Knowledge* is a data structure that is shared and used by the various MAPE components.

Self-adaptation can be realized through two types of mechanisms: following an internal mechanism, or following an external mechanism [361]. In the case of the internal mechanism, self-adaptation is realized by embedding the necessary logic and mechanisms inside the software system itself. An example of how such an internal mechanism is established is through the use of software exceptions provided by a range of popular programming languages. As their name suggests, software exceptions serve the purpose of handling exceptional circumstances a software program needs to tackle. For the external mechanism, self-adaptation is realized through an external entity that is responsible for handling the adaptation logic and mechanisms [376]. The adaptation logic and handling of uncertainties is thus separated from the main software system. A common example of how such an external mechanism is established is through the use of a feedback loop that is responsible for monitoring the environment and adapting the system to changing operating conditions [353, 94]. The system responsible for handling the adaptation logic and mechanisms is commonly referred to as the *Managing System*, and the system that undergoes adaptation is commonly referred to as the *Managed System* [14, 134].

Following the seminal vision paper from Kephart and Chess, the field of self-adaptation gained popularity in the software engineering research community. Through the years, research in self-adaptation can broadly be summarized in 7 topics, denoted as *waves* in [361], as presented in Figure 1.1. For each wave, we elaborate on the core research topic that has been investigated. Afterward, we position the research presented in this thesis in the context of the waves.

The first wave marks the starting point of the field of self-adaptation. The need for self-adaptation stems from the need to build software systems that are able to deal with uncertainties in operation. Whereas operators of software systems were responsible for handling uncertainties, self-adaptation enables the automation of tasks that were previously handled by operators [263, 193].

The second wave follows shortly after the first wave, inspired by the need for a more comprehensive and systematic undertaking of realizing self-adaptation. The second wave introduces the concept of architecture-based self-adaptation [210, 142], where the adaptation logic and mechanisms are separated from the main software

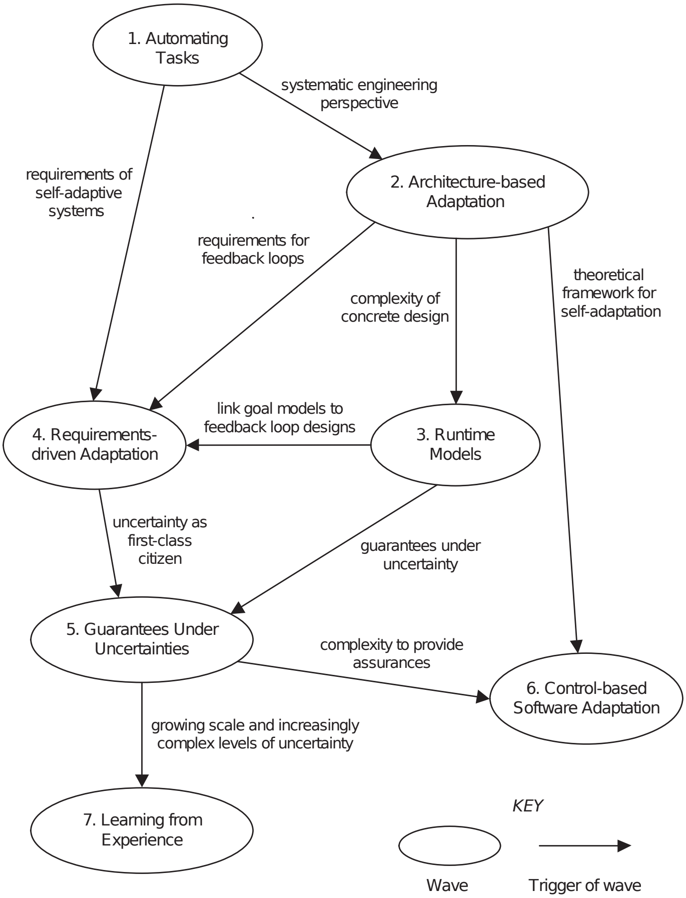

1. Automating Tasks
systematic engineering perspective
2. Architecture-based Adaptation
requirements of self-adaptive systems
requirements for feedback loops
complexity of concrete design
theoretical framework for self-adaptation
link goal models to feedback loop designs
4. Requirements-driven Adaptation
3. Runtime Models
uncertainty as first-class citizen
guarantees under uncertainty
5. Guarantees Under Uncertainties
complexity to provide assurances
6. Control-based Software Adaptation
growing scale and increasingly complex levels of uncertainty
7. Learning from Experience
KEY
Wave
Trigger of wave

Figure 1.1: The seven waves in the field of self-adaptation (borrowed from [361]).

system. An architectural perspective (1) enables a clear separation between goal management and changing operating conditions, and (2) enables effective decision-making through architectural models of the system. The separation of the adaptation logic and mechanisms from the main software system (i.e., *Managed System*) enables reusability across different software systems [375].

The third wave tackles concerns related to detailed modeling of the adaptation logic and mechanisms. The wave introduces the concept of models@runtime [29], putting forward the use of models during the operation of the system. The models are kept up-to-date at runtime and are used to reason about the viability and correctness of potential adaptations to the *Managed System*. The models@runtime approach is commonly used in the context of architecture-based self-adaptation [176].

The fourth wave focuses on the requirements of self-adaptive systems. In this wave, the field of self-adaptation explores languages and formalisms to realize requirements concerning adaptations to the *Managed System*, and requirements related to the realization of the self-adaptive system [377, 325].

The fifth wave puts a lens on uncertainties in self-adaptive systems, and their effects on the guarantees the system can provide. The wave puts forward engineering considerations for designing self-adaptive systems that are able to systematically deal with and provide guarantees in the presence of uncertainties [49]. To accomplish this, the wave explores the use of formal methods to reason about the guarantees the system provides, and the effects of uncertainties on these guarantees [45, 258].

The sixth wave investigates principles from control engineering to (partially) alleviate the complexity of providing guarantees under uncertainties in self-adaptive systems [131, 315]. The wave explores the use of control theory to realize self-adaptive systems, and the use of control theory to reason about the guarantees the system provides.

The seventh and final wave explores the use of machine learning in self-adaptive systems. Motivated by the growing scale of software systems, and the growing complexity of the environments they operate in, the wave aims to equip self-adaptive systems with capabilities to learn from experiences to handle and manage uncertainties more effectively [107, 108, 182].

1.1 Research Background

The work laid out in this thesis is situated in wave 7. We focus on the application of architectural principles to enable the engineering of complex software systems that can efficiently deal with such complex environments. Specifically, we apply architecture-based self-adaptation [142, 374, 263], realized using a feedback loop that follows the Monitor-Analyze-Plan-Execute (MAPE in short) pattern [193, 107]. To realize self-adaptation, we follow the external approach.

In this dissertation, we focus on two main topics: machine learning for self-adaptation, and A/B testing supported by self-adaptation and machine learning.

Machine learning for self-adaptation. Dealing with uncertainties in self-adaptive systems is a complex task [112, 301]. The complexity stems from the fact that the self-adaptive system needs to reason about the effects of potential adaptations in the presence of uncertainties. Additional techniques such as formal verification have been proposed to alleviate this complexity [336]. However, formal verification is a computationally expensive approach, especially when the number of available adaptations is large [365]. Moreover, since self-adaptive systems have to adapt to uncertainties at runtime, the time available to adapt a system may be limited. To alleviate this scalability issue, machine learning presents a promising technique.

A/B testing supported by self-adaptation and machine learning The automation of tasks that were once undertaken by human operators at runtime is at the core of self-adaptation [263, 193]. A field that can benefit from the automation of tasks is A/B testing. Administering A/B tests manually can be time consuming [202, 164, 297, 141], motivating the potential for self-adaptation. Moreover, running A/B tests is cost-intensive[2] itself, typically running for multiple weeks [218, 381]. To alleviate this cost, we explore the use of machine learning to target A/B tests to segments of the population, reducing the time it takes to draw conclusions from A/B tests.

Our choice to focus on machine learning in the context of self-adaptation is twofold. Firstly, an essential aspect of self-adaptation is the system's ability to learn from previous experiences to better handle uncertainties. Machine learning aligns perfectly with this objective, as evidenced by a substantial body of research across various domains [101, 182, 25]. Secondly, machine learning serves as a powerful tool for addressing complex problems, such as the analysis of very large adaptation spaces [251, 324]. In addition, machine learning encompasses a wide array of techniques, including supervised, unsupervised, and interactive learning, which can be applied to address diverse challenges in self-adaptive systems. Each machine learning category also offers a multitude of unique algorithms, each with its own strengths and weaknesses. This diversity is particularly beneficial for self-adaptation since the nature of complex problems can vary significantly from one adaptation problem to another. On the other hand, our choice to focus on machine learning does not mean that alternative approaches, such as search-based techniques, would not be viable alternatives.

In the following sections, we first provide some background on self-adaptation. Then we delve deeper into the two main topics that are central to this thesis: machine learning for self-adaptation, and A/B testing. Afterward, we highlight the research challenges we have identified for each topic. Subsequently, we present the research focus in this thesis and the research questions that we ascertained from the identified research challenges.

[2] Cost-intensive in this context refers to the time it takes to be able to draw conclusions from A/B tests, consequently limiting the number of A/B tests that can be conducted.

1.1.1 Self-Adaptation

Self-adaptation is a software principle that equips software systems with mechanisms to deal with changing operating conditions and circumstances at runtime. Figure 1.2 provides a conceptual view of a self-adaptive system. The figure depicts the following key elements that are central to self-adaptation: a *Managed System*, a *Managing System*, and an *Environment*.

Managed System and Managing System. A self-adaptive system is composed of two elements: the emphManaged System and the *Managing System*. The *Managed System* refers to the software system that is responsible for delivering the core functionalities of the software system, i.e., that handles the domain goals of the system. The *Managing System* is a software system that is responsible for handling the adaptation logic and adaptation mechanisms to adapt the *Managed System*, i.e., that handles the adaptation goals of the system. To accomplish this, the *Managing System* is responsible for monitoring both the environment and *Managed System*, triggering adaptations to the *Managed System* whenever the specified adaptation goals are not achieved. In addition, human operators might still be involved during the adaptation process [364], e.g., to adjust adaptation goals, or to provide additional information to the *Managing System*.

Environment. A self-adaptive system is situated in an environment. The environment is not directly controlled by a self-adaptive system; rather a self-adaptive system can only sense and indirectly affect the environment. Since the environment is not directly controlled, uncertainties may arise that the self-adaptive system has to mitigate [170]. Examples of uncertainties include fluctuations in the network, changes in the workload of the system, changes in the requirements of the system, and changes in the resources available to the system.

Adaptation problem. To meet the adaptation goals of the system, a self-adaptive system undertakes the task of analyzing and selecting suitable adaptation actions that satisfy the specified adaptation goals. Examples include: adding a new replica to the system to deal with increased workload demands [67], changing a system's configuration to optimize the system's performance [179], and recovering components that failed due to a bug or fault [275]. We refer to the problem of analyzing and selecting adaptation actions as the *Adaptation Problem*. To solve the adaptation problem, the self-adaptive system may utilize additional techniques such as for example formal modeling [258], and machine learning [107].

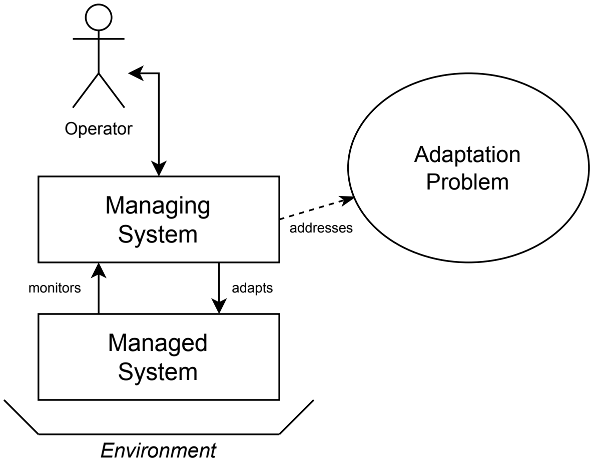


Figure 1.2: Conceptual view of self-adaptation.

1.1.2 Machine Learning for Self-Adaptation

To illustrate the role of machine learning in self-adaptation, we expand on the conceptual view of self-adaptation in Figure 1.3. In addition to the *adaptation problem* that the self-adaptive system has to solve, the figure depicts a *learning problem* that is solved by machine learning. The *learning problem* is a sub-problem of the *adaptation problem*.

Addressing the *adaptation problem* is context-dependent, and undertaking the problem typically involves the use of complementary methods[3]. One example of a complementary method is the use of models@runtime [29] that aids the process of self-adaptation by keeping an up-to-date model of the *Managing System* to facilitate reasoning about the viability and correctness of potential adaptations to the system. Another example of a complementary method is the use of machine learning that solves a sub-problem of the *adaptation problem*, i.e., the *learning problem*. The use of machine learning in self-adaptive systems has gained growing popularity and covers a broad range of applications [151].

The *learning problem* that machine learning can address can cover problems across all MAPE functions of self-adaptation. Most commonly, machine learning is applied to tackle *learning problems* in the *Analyzer*, *Planner*, and *Monitor* functions of the MAPE loop [151]. In the case of the *Analyzer* function, examples of the use of

[3] A complementary method refers to an additional mechanism or tool that is used by the self-adaptive system to enhance the adaptation process.

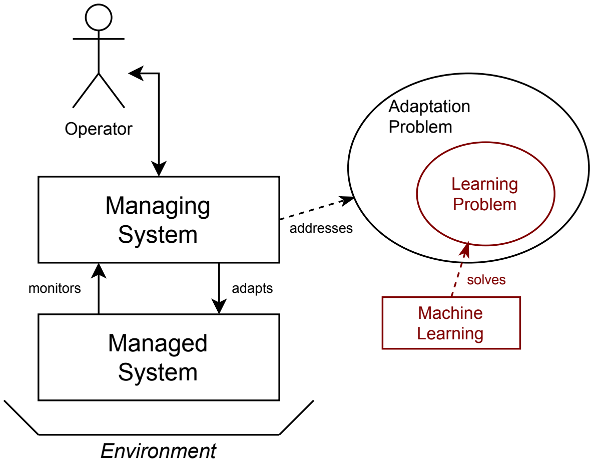


Figure 1.3: Conceptual view of self-adaptation supported by machine learning.

machine learning include performing anomaly detection in the monitored data [128], and reducing large adaptation spaces to a subset of relevant adaptation options [179]. For the *Planner* function, an example of the use of machine learning is the use of reinforcement learning to learn a model of the environment and the system [265]. The model is then used to make predictions of the system's behavior in the environment, and to select the most suitable adaptation option using these predictions [251]. Besides the *Analyzer* and *Planner*, machine learning is also frequently used in the *Monitor* function. An example of the use of machine learning in the *Monitor* function is the processing of monitored data to update knowledge models [209].

1.1.3 A/B testing

Building and maintaining complex software systems is in itself a multi-step and complex process. To address these challenges, software engineering has developed a set of common practices to help engineers in building and maintaining complex software systems. These practices include, but are not limited to, requirements engineering, software architecture, and software testing. In the past, these practices were typically applied in a sequential manner, where the output of one practice is used as input for the next practice. For example, requirements engineering aids in defining the requirements of a software system, which are then used to define the architecture of the software system, which is then used to define the test cases for the software system. In more recent years, software engineers have started to adopt more iterative and incremental approaches to software engineering. One such approach is the Development and Operations (DevOps in short) methodology.

In 2009, the first DevOpsDays conference was held in Ghent, Belgium. The DevOps movement is a reaction to the traditional sequential approach to software engineering, and aims to bring together the development and operations teams to work together throughout the entire software development lifecycle. DevOps comprises a set of tools, practices and guidelines for software engineers to build, test, and release software faster and more reliably. The DevOps movement has been growing in popularity since its inception, and has been adopted by many companies. The DevOps movement has also been adopted by the software engineering community, and has been the subject of many research studies.

Key to DevOps is the concept of Continuous Integration and Continuous Deployment (CICD in short). CICD is the practice of continuously developing and engineering software, and rapidly deploying software to production [172, 252, 296]. CICD is supported by a set of tools and practices that enable software engineers to continuously develop and deploy software. One such key practice is the use of A/B testing [203, 202, 116]. A/B testing enables software engineers to test new features or software updates in live production environments, providing a way to assess the viability of new features or software updates and make data-driven decisions [117, 338]. A/B testing is a practice where two versions of a feature, typically denoted as A and B, are deployed to production simultaneously. A feature can refer to any element of a software system, e.g., a variation in the user interface of an application, a variation in an algorithm, and a newly introduced software functionality in the application. Figure 1.4 depicts an overview of the A/B testing process.

During deployment of the variants, relevant data is tracked that is used to assess the merit of the variants. The data that is tracked is typically related to the goals of the software system, e.g., the number of clicks on a button in the user interface, the number of users that use a particular functionality in the application, and the time it takes users to complete a task in the application. The data is then analyzed to assess the viability of the variants. The version that performs better is then typically selected for further development or deployment in the software system. A/B testing is a popular practice in the industry and has been the subject of many research studies.

To illustrate the role of A/B testing in relation to self-adaptation and machine learning in this dissertation, we expand on the conceptual view of self-adaptation supported by machine learning in Figure 1.5. Similarly to Figure 1.3, machine learning solves a *learning problem* that is a sub-problem of the *adaptation problem*. The *Managed System* is the system under test, i.e., the system in which A/B testing is conducted. The *Managing System* is responsible for conducting the A/B testing, including setting up, monitoring, and analyzing the results of the experiment.

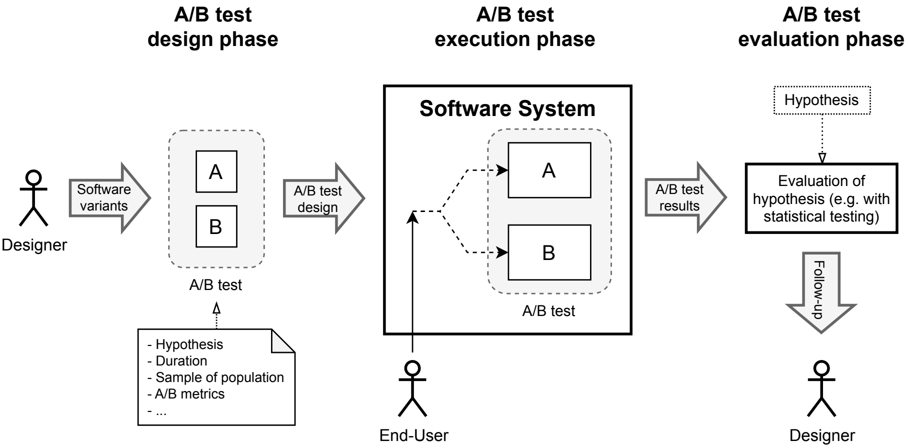


Figure 1.4: Overview of the A/B testing process.

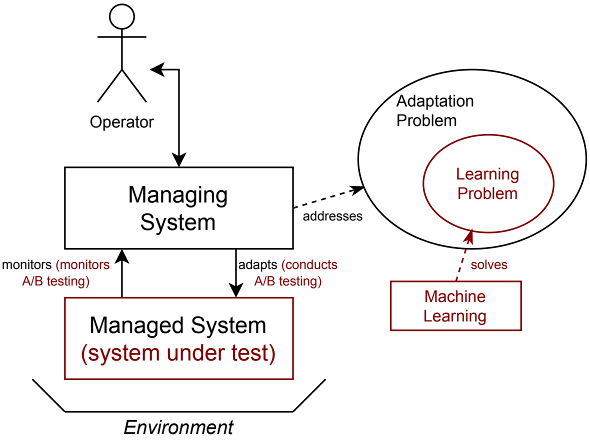


Figure 1.5: Conceptual view of self-adaptation supported by machine learning to enhance A/B testing.

1.2 Research Challenges

We highlight now the research challenges we have identified in the context of the two research topics. We zoom in on concrete challenges in the context of machine learning for self-adaptation and research challenges in the context of A/B testing.

1.2.1 Machine Learning for Self-Adaptation

We zoom in on one particular learning problem that machine learning can effectively address in the analysis function of the MAPE loop: reducing large adaptation spaces. Analyzing the viability of a large number of adaptation options can be time-consuming. Moreover, the analysis of large adaptation spaces may prove to be infeasible due to the time penalty it incurs to adapt the system that undergoes the adaptation [366]. One approach to deal with this issue is by reducing large adaptation spaces to a subset of relevant adaptation options.

Various works have investigated machine learning techniques to reduce adaptation spaces in self-adaptive systems. Reinforcement learning is one of the popular choices to realize this [53, 234, 273]. Reinforcement learning is a machine learning technique that is concerned with learning a policy that maximizes the reward of an agent in an environment. In the context of adaptation space reduction, reinforcement learning has been used to learn a policy that selects the most suitable adaptation option for a given adaptation goal. Other techniques that have been used to reduce adaptation spaces consist of the use of neural networks [88], and decision trees [107].

However, in these works machine learning is either used to replace the analysis and planning functions entirely, or the machine learning techniques are entangled in the analysis process in the MAPE-K loop. This entanglement makes it difficult to utilize additional techniques in the analysis of the viability of adaptation options, such as formal verification methods. Additionally, the entanglement in specific processes makes it difficult to reuse the machine learning techniques across different self-adaptive systems.

1.2.2 A/B Testing

A/B testing is a popular industry practice and has been the subject of many research studies. However, the practice of A/B testing is still a manual process, and requires a significant amount of manual effort to conduct A/B testing [164, 202, 141, 300]. Furthermore, some researchers have suggested automating the A/B testing process [247, 141, 59]. To address this, some preliminary work has been conducted on automating facets of the A/B testing process [77, 245, 335, 291]. However, the automation of the

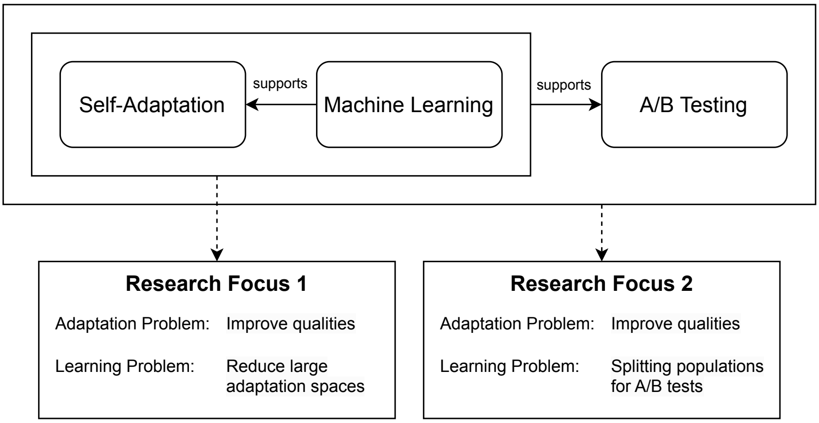

Self-Adaptation
supports
Machine Learning
supports
A/B Testing
Research Focus 1
Adaptation Problem: Improve qualities
Learning Problem: Reduce large adaptation spaces
Research Focus 2
Adaptation Problem: Improve qualities
Learning Problem: Splitting populations for A/B tests

Figure 1.6: Research focus of this thesis.

A/B testing process is still in its infancy, and requires further research to be adopted in practice.

Another challenge particular to A/B testing is the need for efficiency. Efficiency in this context refers to discovering the true effect of a new feature or software update as quickly as possible. The need for efficiency is particularly important in the context of A/B testing, as the longer it takes to discover the true effect of a new feature or software update, the longer it takes to make data-driven decisions. Moreover, as the duration of an A/B test increases, the number of A/B tests that can be conducted is further restricted. Numerous techniques have been proposed to improve the efficiency of A/B testing. One way of improving the efficiency is by improving the sensitivity of A/B testing results, e.g., exploiting the time it takes users to get used to novel features in the analysis of the results [98], and using machine learning techniques [161, 332, 227]. Another technique is the use of additional data to analyze the results of the A/B tests, such as using periodicity patterns in the user engagement to judge the results [97], and using regression techniques to reduce the variance in A/B testing results [274]. However, the use of machine learning has not been explored yet in the context of targeting A/B testing to a subset of the population that is directly relevant to the experiment.

1.3 Research Focus

The research in this thesis is divided into two lines, corresponding to the two research topics presented in the previous section. Figure 1.6 depicts the relation of the topics in this dissertation to the lines of research. For each, we provide a brief outline, and we elaborate on the concrete adaptation and learning problems that we address.

Self-Adaptation supported by Machine Learning In the first line of research, we address the challenges identified in the context of machine learning that supports self-adaptive systems. We concretely focus on the following high-level adaptation problem: *Improve qualities*. In other words, self-adaptation is used to improve the qualities in the *Managed System*, such as e.g. response time or energy consumption in an Internet-of-Things application. The learning problem supporting self-adaptation that we address is the following: *Reduce large adaptation spaces*. Solving this learning problem supports self-adaptation by focusing more time on the analysis of relevant adaptation options, and less time on the analysis of irrelevant adaptation options. In turn, the *Managing System* can adapt the *Managed System* more effectively.

A/B testing supported by Machine Learning and Self-adaptation In the second line of research, we address the challenges identified in the context of A/B testing. We focus on the same high-level adaptation problem as in the first line of research: *Improve qualities*. The learning problem supporting self-adaptation we address in this line of research is the following: *Splitting populations for A/B tests*. Solving this learning problem supports self-adaptation in its task of conducting efficient A/B testing by focusing the A/B tests on a subset of the population that is directly relevant to the A/B test.

1.3.1 Research Questions

Based on the identified challenges and the research focus in this dissertation, we put forward two research questions. Each research question corresponds to the line of research presented before. The research questions are as follows:

RQ1: How can machine learning be used to reduce large adaptation spaces of self-adaptive systems with different types of adaptation goals to perform more efficient analysis without compromising the goals?

RQ2: How can we automate A/B testing pipelines, and how can machine learning be used to run A/B testing pipelines more efficiently?

1.3.2 Research Context

Assumptions We make the following assumptions in our research that hold for both lines of research, unless stated differently

- We assume that we have knowledge about the boundaries of the adaptation space beforehand, i.e., theoretically, we can enumerate the possible adaptation options.
- We assume that boundaries of the adaptation space do not change at runtime.
- We assume that the learning problems we consider are linear, i.e., the learning problems can be solved using linear machine learning models.
- We assume that the uncertainties we cover are of the type known unknowns [170, 111].

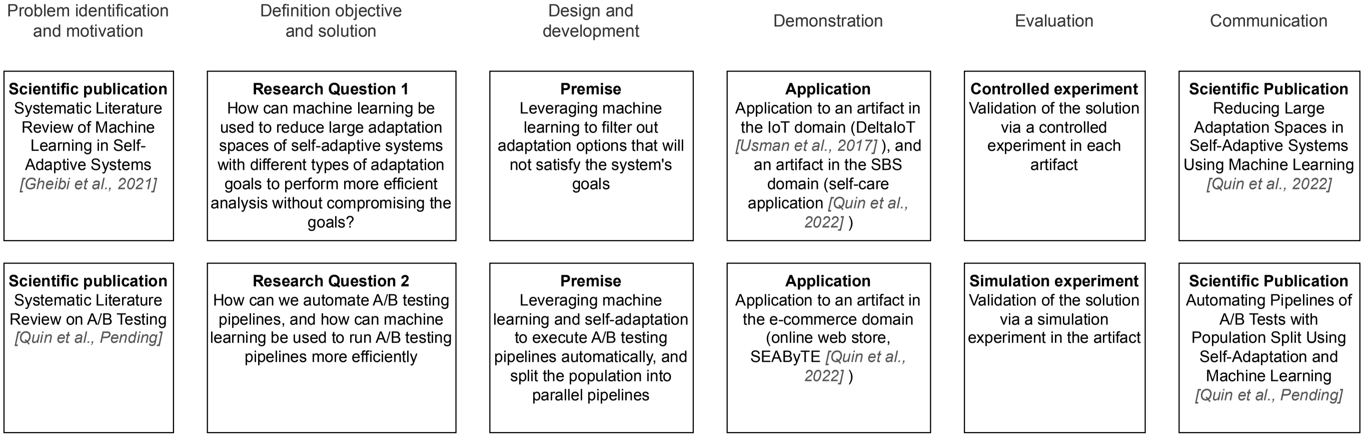
1.4 Research Methodology

In this section, we present the research methodology we followed to answer the research questions. The research methodology is based on the Design Science Research (DSR in short) methodology [379, 378]. DSR is a research methodology that is commonly used in the field of software engineering. DSR is a problem-solving paradigm that aims to develop and evaluate artifacts that address identified problems. The DSR methodology consists of six steps, namely (1) Problem identification and motivation, (2) Objective and solution definition, (3) Design and development, (4) Demonstration, (5) Evaluation, and (6) Communication. Figure 1.7 presents an overview of the DSR methodology applied in this dissertation.

The methodology lays out two main tracks, corresponding to the proposed research questions. The first track is concerned with the use of machine learning to reduce large adaptation spaces. The second track is concerned with the automation of A/B testing pipelines, and the efficient operation of those pipelines by using machine learning.

1.4.1 Machine learning to reduce large adaptation spaces

In the first research track, we investigate the use of machine learning to reduce large adaptation spaces. For problem identification and motivation, we conduct a systematic literature review to identify the state of research on machine learning in the field of self-adaptation. A systematic literature review (SLR in short) is a scientific approach [192, 198] that ensures that the review identifies, evaluates, and interprets all relevant literature. Systematic literature reviews are conducted using a research protocol that is developed prior to the execution of the review. The protocol consists of a rigorous description of the three main phases in a literature review: planning, execution,



Problem identification and motivation	Definition objective and solution	Design and development	Demonstration	Evaluation	Communication
Scientific publication Systematic Literature Review of Machine Learning in Self-Adaptive Systems *[Gheibi et al., 2021]*	**Research Question 1** How can machine learning be used to reduce large adaptation spaces of self-adaptive systems with different types of adaptation goals to perform more efficient analysis without compromising the goals?	**Premise** Leveraging machine learning to filter out adaptation options that will not satisfy the system's goals	**Application** Application to an artifact in the IoT domain (DeltaIoT *[Usman et al., 2017]*), and an artifact in the SBS domain (self-care application *[Quin et al., 2022]*)	**Controlled experiment** Validation of the solution via a controlled experiment in each artifact	**Scientific Publication** Reducing Large Adaptation Spaces in Self-Adaptive Systems Using Machine Learning *[Quin et al., 2022]*
Scientific publication Systematic Literature Review on A/B Testing *[Quin et al., Pending]*	**Research Question 2** How can we automate A/B testing pipelines, and how can machine learning be used to run A/B testing pipelines more efficiently	**Premise** Leveraging machine learning and self-adaptation to execute A/B testing pipelines automatically, and split the population into parallel pipelines	**Application** Application to an artifact in the e-commerce domain (online web store, SEAByTE *[Quin et al., 2022]*)	**Simulation experiment** Validation of the solution via a simulation experiment in the artifact	**Scientific Publication** Automating Pipelines of A/B Tests with Population Split Using Self-Adaptation and Machine Learning *[Quin et al., Pending]*

Figure 1.7: Overview of the DSR methodology applied in this dissertation.

and synthesis. The planning phase consists of defining the research questions, the search strategy, the inclusion and exclusion criteria, and the data items to be extracted. The execution phase consists of executing the search strategy, applying the inclusion- and exclusion criteria, and data extraction. The synthesis phase consists of analyzing the extracted data and reporting the results to the research community. The protocol ensures the repeatability of the study and reproducibility of the results.

Based on the findings of the literature review, we define the objective of *Research Question 1*, as outlined in Section 1.3.1. We then put forward the premise of the envisioned solution to the proposed research question: leveraging machine learning to filter out adaptation options that will not satisfy the system's goals.

The premise is then evaluated in the context of two application domains: Internet-of-Things and Service-Based Systems. The evaluation and validation of the approach happen in the form of a controlled experiment [380]. Controlled experiments presented a suitable scientific approach to evaluate the approach in the two application domains: DeltaIoT, an internet-of-things exemplar [175] that simulates a real-world IoT network; and a simulator of a service-based system in the context of patient self-care [289], inspired by a popular artifact in the self-adaptation community called Tele-Assistance System [367]. Finally, the results of the evaluation are then communicated in the form of a scientific publication titled "Reducing Large Adaptation Spaces in Self-Adaptive Systems Using Machine Learning" [289].

1.4.2 Automatic and efficient execution A/B testing pipelines

In the second research track, we investigate the automation of A/B testing pipelines, and the efficient execution of those pipelines by using machine learning. For problem identification and motivation, we conduct a systematic literature review to identify the state of research on A/B testing. We refer back to Section 1.4.1 for a description of the systematic literature review process.

Based on the findings of the literature review, we define the objective of *Research Question 2*, as outlined in Section 1.3.1. We then put forward the premise of the envisioned solution to the proposed research question: leveraging machine learning and self-adaptation principles to split the population into parallel A/B testing pipelines, and execute the pipelines automatically.

The premise is then evaluated in the context of a web-store application. The evaluation and validation of the approach happen in the form of a simulation experiment. We refer to a simulation experiment as an experiment that is conducted in a simulated environment or setting where a baseline approach is compared to the proposed approach. In our case, we compare a baseline approach of A/B testing with our proposed approach via simulated A/B tests. Simulation experiments presented a suitable scientific approach to evaluate the approach in the context of A/B testing due to the lack of publicly available industrial datasets. The simulation experiment is conducted using a web-store application artifact, called SEAByTE, which is specifically designed to conduct A/B testing. We developed and published this artifact for the research community to validate self-adaptation approaches in the context of A/B testing [283]. Finally, the results of the evaluation are then communicated in the form of a scientific publication submitted for review [281].

1.4.3 Contributions

We now summarize the contributions presented in this thesis, and how each contribution relates to the presented research question in this thesis. The contributions are laid out in separate chapters. At the start of each chapter, we additionally give a brief overview of the presented work alongside key information about the presented work.

Contribution 1. The first contribution is a summarized version of a systematic literature study that investigates the use of machine learning within the field of self-adaptation. The summarized version contains snippets of the original publication that highlight relevant insights and findings leading to the formulation of *Research Question 1*.

Contribution 2. The second contribution explores an approach to handle large adaptation spaces named Machine Learning to Adaptation Space Reduction Plus (ML2ASR+ in short). The chapter provides an answer to *Research Question 1* by laying out an architecture-based self-adaptive solution, supported by a semi-formal notation, that is able to adapt software systems in the presence of large adaptation problems using classic machine learning techniques (classification and regression). The approach is then evaluated in the application domains of Internet-of-Things (IoT) and Service-Based Systems (SBS).

Contribution 3. The third contribution presents a systematic literature review on the use of A/B testing. The literature study explores the subject of A/B testing, how A/B tests are designed and executed, and what open problems remain in the field of A/B testing. During this literature review, one of the gaps we identified was related to the automation of aspects of the A/B testing process. This gap leads us to devise and explore *Research Question 2*.

Contribution 4. The fourth contribution presents an approach to automate A/B testing pipelines, leveraging self-adaptation principles and machine learning to automate and improve the efficiency of the A/B testing process. This chapter addresses *Research Question 2* by providing an architecture to enable the automatic execution of A/B testing pipelines and enabling parallel running of A/B tests when appropriate. We evaluated the approach in the context of a web-store application, comparing it to a baseline approach that conducts the A/B testing in a classic sequential way.

Contribution 5. The fifth contribution of this dissertation is an artifact of a web-store application, with explicit support to conduct A/B testing in the application. We utilized the artifact to validate the approach presented in Chapter 5.

1.4.4 Positioning contributions

Research track 1. The first research track is situated entirely in the research domain of self-adaptive systems. The research track consists of two concrete contributions: a systematic literature review on the use of machine learning to support self-adaptive systems, positioned in the self-adaptive systems research domain; and ML2ASR+ in the self-adaptive systems research domain to deal with large adaptation spaces.

Research track 2. The second research track is situated in the research domains of self-adaptive systems and A/B testing. The research track consists of three concrete contributions: a systematic literature review on A/B testing positioned entirely in the A/B testing research domain; AutoPABS for automated and segmented A/B testing on soft facts within the A/B testing research domain, leveraging concepts from the self-adaptive systems research domains; and SEAByTE, an artifact for the self-adaptive systems research domain that enables the validation of self-adaptation approaches in the context of A/B testing.

1.5 Thesis outline

The remainder of this thesis is structured as follows. Chapter 2 explores the use of machine learning in the context of self-adaptive systems. This chapter is devoted to *Contribution 1*, and is based entirely on an authored prior publication [151].

Chapter 3 introduces an approach to deal with large adaptation spaces in self-adaptation by leveraging machine learning to reduce the adaptation space to a subset of relevant adaptation options. This chapter is devoted to *Contribution 2*, and is based entirely on an authored prior publication [289].

Chapter 4 investigates A/B testing through the lens of software engineering. This chapter is devoted to *Contribution 3*, and is based entirely on a paper under review [286].

Chapter 5 introduces an approach to realize the efficient automated execution of A/B testing pipelines in the underlying system. This chapter is devoted to *Contribution 4*, and is based entirely on a paper under review [281].

Chapter 6 presents an artifact of a web-store application that enables testing and validating self-adaptation approaches in the context of A/B testing of the web-store application. This chapter is devoted to *Contribution 5*, and is based entirely on an authored prior publication [283].

Lastly, we conclude the dissertation in Chapter 7, where we summarize the contributions of this thesis, and discuss opportunities for future work.

Chapter 2

Systematic Literature Review on Machine Learning in Self-Adaptation

Publication details. This chapter is based entirely on a journal publication in the journal Transactions on Autonomous and Adaptive Systems (TAAS) [151]. We present a shortened version by focusing on aspects relevant for this dissertation. Shortened sections are marked with an ellipsis.

Personal contributions. Conceptualization (20%), Methodology (20%), Data collection (20%), Validation (30%), Formal analysis and interpretation results (25%), Writing (25%), Visualization (20%).

Positioning. Recently, we witness a rapid increase in the use of machine learning techniques in self-adaptive systems. While a body of work on the use of machine learning in self-adaptive systems exists, there is currently no systematic overview of this area. This chapter reports the results of a systematic literature review that aims at providing such an overview. We focus on self-adaptive systems that are based on a MAPE-based feedback loop. Results show that machine learning is mostly used for updating adaptation rules and policies to improve system qualities, and managing resources to better balance qualities and resources. These problems are primarily solved using supervised and interactive learning with classification, regression, and reinforcement learning as the dominant methods. Surprisingly, unsupervised learning that naturally fits automation is only applied in a small number of studies. Key open challenges in this area include the performance of learning, managing the effects of learning, and dealing with more complex types of goals.

2.1 Introduction

The ever-growing complexity of software systems that need to maintain their goals 24/7 while operating under uncertainty motivates the need to equip systems with mechanisms to handle change during operation. An example is a cloud service that needs to satisfy user performance and minimize operational costs while operating under dynamically changing workloads. This service may be enhanced with an elasticity module that adjusts resources with changing workloads.

A common approach to handle change is the use of "internal mechanisms", such as exceptions (as a feature of a programming language) and fault-tolerant protocols. The application of such mechanisms is often domain-specific and tightly bound to the code. This makes it costly to build, modify, and reuse solutions [142]. In contrast, change can be handled using "external mechanisms" that are based on the concept of a feedback loop. An important paradigm in this context is self-organization, where relatively simple elements apply local rules to adapt their interactions with other elements in response to changing conditions in order to cooperatively realize the system goals [169, 268]. Another important paradigm is control-based adaptation that relies on the mathematical basis of control theory for designing feedback loop systems and analyzing and guaranteeing key properties [395, 314].

In this chapter, we focus on architecture-based adaptation, which is an extensively studied and applied approach to handle change [193, 142, 375, 29, 64, 362]. Architecture-based adaptation relies on a feedback loop that monitors the system and its context and adapts the system to ensure its goals, or degrade gracefully if necessary. Pivotal in tackling this task is the use of runtime models [142, 29] that enable the system to reason about (system-wide) change and make adaptation decisions. The feedback loop localizes the adaptation concerns in separable system elements that can be analyzed, modified, and reused across systems. Examples of practical applications of architecture-based adaptation are management of renewable energy production plants [50], information systems for public administration [320], and automation of the management of Internet of Things applications [372].

Realizing feedback loops for architecture-based adaptation is in general not a trivial task. Over the past years, several techniques have been investigated to support the design and operation of self-adaptation. One of these techniques is search-based software engineering. For instance, [65] argues for the use of evolutionary computation to generate and analyze models of dynamically adaptive systems in order to deal with uncertainties both at development time and runtime. Our focus in this chapter is on another prominent line of research that applies machine learning techniques in the design and operation of self-adaptation. We highlight a few of the incentives to apply machine learning techniques to architecture-based self-adaptive systems. Online verification of rigorously specified runtime models enables providing guarantees about the adaptation decisions. However, formal verification of runtime models for all possible adaptation options during operation can be time-consuming. We may then use a learning mechanism to filter the configurations before starting the analysis. Another challenging aspect is the design of runtime models of complex software systems. These models may become particularly complicated up to the level that it may be infeasible to design the models if the structure of the system or its context is not known beforehand. Hence, the models need to be derived during operation, for which we may use learning techniques.

At the current point in time, we have a sizeable body of work in this area. Yet, even though the amount of research that has been conducted on this topic is quite substantial, there is no clear view of the state-of-the-art. Such an overview is important for researchers in this area as it will document the current body of knowledge and clarify open challenges. To tackle this problem, we performed a systematic literature review [192]. The goal of this study is to provide a systematic overview of the state of the art on the application of machine learning methods in self-adaptation. The survey is centered on (i) the problems that motivate the use of machine learning in self-adaptive systems, (ii) key engineering aspects of learning applied in self-adaptation, and (iii) open challenges.

The remainder of this chapter is structured as follows. Section 2.2 starts with explaining the background and outlining the focus of the literature review. In Section 2.3, we position this review in the landscape of related review work. Section 2.4 then summarizes the protocol of the study that includes the research questions, the search string, inclusion and exclusion criteria, the data items that we extracted from the papers, and the methods we used for the analysis. Section 2.5 reports the results from the analysis of the collected data, providing answers to the research questions. In Section 2.6, from the main findings of the literature review, we present an initial design process for applying machine learning in self-adaptive systems, we present opportunities for further research, and we discuss threats to validity. Finally, we wrap up and conclude the chapter in Section 2.7.

2.2 Background and Review Focus

In this section, we briefly introduce self-adaptive software with a MAPE-based feedback loop, and we summarize the essential dimensions of machine learning methods. Then we clarify the distinction between an adaptation problem and a learning problem in the context of this study, and we highlight the importance of the use of machine learning in self-adaptive systems.

2.2.1 MAPE-based Self-adaptation

This study focuses on self-adaptive systems based on architecture-based adaptation [142, 210, 375]. Such a self-adaptive system comprises a *managed system* that is controllable and subject to adaptation, and the *managing system* that performs the adaptations of the managed system. The managed system operates in an *environment* that is non-controllable. The managing system realizes a feedback loop that comprises four essential functions: Monitor-Analyze-Plan-Execute that share Knowledge; often referred to as MAPE-K or MAPE in short [193]. The monitor tracks the managed system and the environment in which the system operates and updates the knowledge. The analyzer uses the up-to-date knowledge to evaluate the need for adaptation, possibly exploiting rigorous analysis techniques [45, 176, 51] or simulations of runtime models [369]. Such analysis may apply rigorous methods to provide guarantees for the If adaptation is required, it analyzes alternative configurations of the managed system. We refer to these alternative configurations as the adaptation options. The planner then selects the best option based on the adaptation goals and generates a plan to adapt the system from its current configuration to the new configuration. Finally, the executor executes the adaptation actions of the plan. It is important to highlight that MAPE provides a reference model that describes a managing system's essential functions and the interactions between them. A concrete architecture maps the functions to corresponding components, which can be a one-to-one mapping or any other mapping, such as a mapping of the analysis and planning functions to one integrated decision-making component.

In this literature review, we consider studies that are based on the MAPE reference model that maps the MAPE functions (or some of them) to a specific component-based architecture.

2.2.2 Machine Learning

T. Mitchell defines machine learning as follows: "A computer program is said to learn from experience E concerning some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [255]. For example, consider a self-adaptive sensor network that needs to keep packet loss and latency under given thresholds. The training experience (E) from which we learn could be the results of the analysis of adaptation options. The task (T) could be a classification of the adaptation options in two classes: those that are predicted to comply with the goals (and should be analyzed) and those that are predicted not to comply (and should not be analyzed). The performance measure (P) to perform this task could be the comparison of predicted values of packet loss and latency with the threshold values of the respective adaptation goals. In this example, learning (classification) supports the analysis stage of the feedback loop by reducing a large number of adaptation options, aiming to improve the efficiency of analysis.

In the field of machine learning, a distinction can be made in four dimensions: [310, 28]:

- *Unsupervised vs Supervised vs Interactive*: An unsupervised learner aims at finding previously unknown patterns in data sets without preexisting labels. One of the main methods used in unsupervised learning is cluster analysis. Cluster analysis identifies commonalities in the data and reacts based on the presence or absence of such commonalities in new data. A supervised learner learns a function that maps an input to an output based on example input-output pairs. The function is inferred from labeled data and can then be used to map new data. An interactive learner collects the input-output pairs by interaction with the environment. The learner uses a learning model to make predictions that are then used to perform actions in the environment. The environment then provides feedback on the actions that the learner incorporates in its learning model improving the learning process; a classic example is reinforcement learning. We refer to the basic dimension that distinguishes unsupervised, supervised, and interactive learning as *learning type*.
- *Active vs Passive*: In active learning, a learning algorithm interactively queries some information source in the environment (e.g., a user or teacher) to obtain the desired outputs at new data points. These outputs in turn are used to affect the environment. A passive learner only perceives the information from the environment without affecting it.
- *Adversarial vs Non-Adversarial*: Adversarial learning attempts to fool models through malicious input. This technique can be applied to attack standard learning models. An example is an attack in spam filtering, where spam messages are obfuscated through the manipulation of the text. Non-adversarial learning has no concept of malicious input.
- *Online vs Batch Protocol*: In online learning, data becomes available in a sequential order and is used to update the learning model for future data at each step. Batch learning on the other hand generates the learning model by learning on the entire training data set at once.

In the literature review, we consider the four dimensions of machine learning methods.

2.2.3 Adaptation Problem versus Learning Problem

As explained above, in this survey we target software systems that comprise two parts: a managed system that is adapted by a managing system, as illustrated in Figure 2.1.

The managing system is based on a MAPE-K feedback loop that solves an *adaptation problem*. An adaptation problem relates to one or more concerns of a managed system that typically pertain to quality properties. The MAPE-K feedback loop is supported by a *machine learner* that solves a particular *learning problem*. Hence, in this study, we look at learning problems that are part of adaptation problems.

Consider the example we used in the introduction of a cloud service as a managed system. The adaptation problem is to ensure user performance while minimizing operational costs for the owner under changing workloads. To that end, the cloud service is extended with an elasticity module. This module realizes a feedback loop that dynamically adjusts the resources of the cloud service based on changing workloads. A learning problem in this context may be the prediction of the workload of the cloud service. Such a learner would support the monitor and analyzer of the feedback loop of the elasticity module. Hence, the elasticity of the cloud service is realized by the collaboration of the MAPE-K feedback loop and the machine learner that together form the managing system.

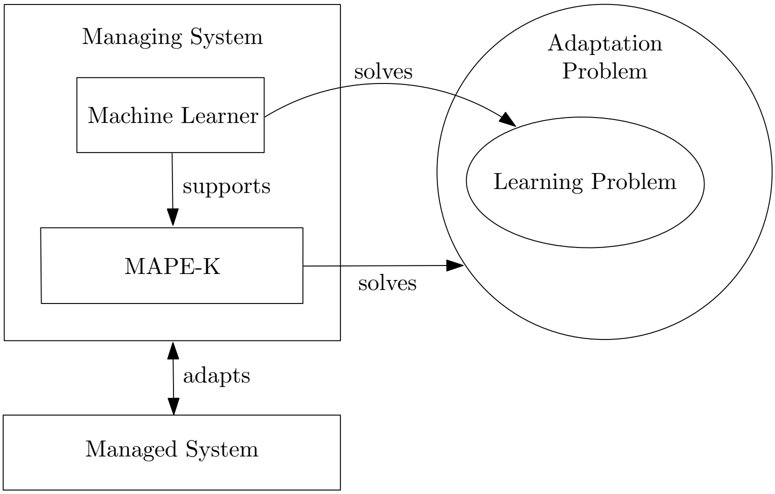


Figure 2.1: Relation of learning problem and adaptation problem with the components of the managing system.

2.2.4 Importance of Machine Learning in Self-Adaptive Systems

A recent book structures the field of self-adaptation in seven "waves" that highlight the important research areas of the past two decades that have contributed to the current body of work [361]. The seventh wave focuses on machine learning techniques as a means to enhance the realization of a self-adaptive software system. The book argues that machine learning can be used to support different activities of the MAPE workflow of self-adaptive systems and highlights three characteristic use cases. The first use case enhances the monitor function of a self-adaptive system with a Bayesian estimator that keeps a runtime model up to date. A concrete example is described in [108]. This simple use case underpins the power of machine learning when dealing with parametric uncertainties represented in runtime models. The second use case enhances the analyzer function with a classifier that enables large sets of adaptation options to be reduced at runtime, improving the efficiency of the analysis phase of self-adaptation. A concrete example is described in [284]. This use case shows how learning can help to deal with the complexity that comes with the increasing scale of self-adaptive systems. Finally, the third use case enhances various feedback loop functions with a learning strategy that combines fuzzy control and fuzzy Q-learning to adjust and improve auto-scaling rules of a cloud infrastructure at runtime. An example is described in [182]. This use case shows how machine learning can help to support decision-making in self-adaptive systems that are subject to complex types of uncertainties.

These examples underpin the importance of machine learning techniques in self-adaptive systems. This chapter aims to study the use of machine learning in self-adaptation in a systematic way in order to document the current body of knowledge in this area and identify open challenges.

2.3 Related Reviews

...

Several reviews related to this study have been published, but they differ in the methodology used, the domain studied, or the types of feedback loops considered. Table 2.1 summarizes the related work.

2.4 Summary Protocol

...

Table 2.1: Summary of related reviews (SLR refers to Systematic Literature Review).

Related review	**Method**	**Domain**	**Type feedback loop**
Klaine et al. [347]	Literature review	Self-organizing networks	Control-theoretic
D'Angelo et al. [75]	SLR	Collective self-adaptive systems	Decentralized
Gambi et al. [140]	Not specified	Cloud	Not constrained
Lorido-Botran et al. [237]	Not specified	Cloud	Not constrained
Masdari et al. [243]	Not specified	Cloud	Systems in general
Cui et al. [74]	Literature review	IoT	Systems in general
Saputri and Lee [304]	SLR	Self-adaptive systems	Systems in general

2.5 Results

We now report the results. We start with some general information and then give results grouped per research question. All data and analysis results of the study are available on the study website[1].

2.5.1 Demographics

. . .

2.5.2 RQ1: What problems have been tackled by machine learning in self-adaptive systems?

To answer this research question, we analyze the data of the following data items: Adaptation problem (F7), Learning problem (F8), Application domain (F12).[2]

Adaptation problem. In a concrete setting, self-adaptation is applied to solve a particular adaptation problem. This adaptation problem refers to the concerns that the managing system is dealing with. Previous research has shown that these concerns relate to quality properties of, and resources used by the system, see e.g., [352, 363]. Table 2.2 lists the adaptation problems we have identified from the papers, illustrated with examples. The last column shows the number of papers for each type.

Learning problem. A learning problem refers to a concrete problem that needs to be solved by machine learning in support to realize self-adaptation. Table 2.3 shows the six types of learning problems that we have identified from the papers. Each type is illustrated with examples. The last column shows the number of papers for each type of learning problem.

Adaptation problems versus Learning problems. We can now map adaptation problems to learning problems. This mapping allows us to identify whether particular types of adaptation problems delegate particular sub-problems to a machine learner. Figure 2.2 shows the mapping.

A few observations jump out. First, self-adaptive systems that aim at improving the qualities of the managed system primarily use learning to solve the problem of updating

[1] https://people.cs.kuleuven.be/danny.weyns/material/ML4SAS/SLR/

[2] During the coding process, we used the following terminology. We used the term *quality* to refer to non-functional requirements that describe how a system should perform its functions, such as its performance and reliability. We used the term *resource* to refer to the means or supplies a software system uses to realize its functions, such as memory and bandwidth. We used the term *cost* to refer to monetary aspects, i.e., the price one has to pay to use or operate a system.

Table 2.2: Adaptation problems illustrated with examples, with the frequencies of papers (right column).

Adaptation Problem	**Brief Description with Example**	**#**
Improve qualities	This adaptation problem is about improving (i.e., optimizing, maintaining, etc.) quality properties of the system. An example is keeping the response time low and the reliability high under changing workloads and the occurrence of unexpected events [107].	44
Balance qualities with resources	This adaptation problem is about keeping a balance between improving quality properties of the system and resources (i.e., CPU, energy, etc.) required to achieve that improvement. For example, [241] aims at keeping the latency within a specified range while minimizing the computational resource required to achieve that.	37
Balance qualities with cost	This adaptation problem is about keeping a balance between improving the system's quality properties and the cost (i.e., operational, financial, etc.) required to achieve that. For example, [45] aims at keeping the failure rate of a service-based workflow below a specified threshold while minimizing the cost of using the services.	15
Improve resource allocation	This adaptation problem is about improving (i.e., optimizing, managing, etc.) the system's resource consumption. An example is managing the system's CPU and memory usage under uncertain workloads of the system [180].	10
Protect against cyber threats	This adaptation problem is about automating the cyber defense of a system by detecting and managing threats (i.e., intrusion, anomalies, etc.). For instance, [128] considers cyber defense in fifth-generation mobile systems by detecting and dealing with system intrusions.	3

Table 2.3: Learning problems illustrated with examples, with the frequencies of papers (right column).

Learning Problem	Brief Description with Example	#
Update or change adaptation rules or policies	This learning problem is about updating or changing adaptation rules or policies to support a managing system when dealing with changing operating conditions. For example, in [159], the managing system is supported by a reinforcement learner that dynamically updates adaptation policies to deal with changing workloads.	36
Predict or analyze resource usage	This learning problem is about predicting or analyzing resources that are used by the managed system that affect the decision-making of the managing system. Examples are learners that predict the energy consumption of batteries [234], storage [191], and CPU usage [191].	23
Keep runtime models up-to-date	This learning problem is about supporting a managing system by keeping runtime models up-to-date. Examples are a model of the environment [331], a performance model [183], and a reliability model [48].	18
Reduce large adaptation space	This learning problem is about supporting a managing system by reducing a large number of adaptation options (large adaptation space) such that the system can make more efficient decisions. For instance, [324] and [284] use learners to predict quality properties of adaptation options to select options, speeding up analysis.	16
Detect or predict anomalies	This learning problem is about detecting or predicting anomalies in the behavior of the system or its environment that are relevant for adaptation. For example, in [211] a learner detects abnormal flow of traffic in a traffic management system, and in [267] a learner identifies cyber threats in a communication network.	12
Collect unavailable prior knowledge	This learning problem is about collecting initially unknown runtime knowledge to support adaptation. For instance, in [148] a learner builds a performance model to support the managing system with computing the utility of different configurations; in [340] a learner identifies management policies without any prior knowledge.	4

and changing adaptation rules and policies. Second, self-adaptive systems that aim at protecting against cyber threats exploit learning only to detect or predict anomalies. Third, self-adaptive systems that aim at balancing qualities with resources used by the system exploit learners to solve all types of learning problems.

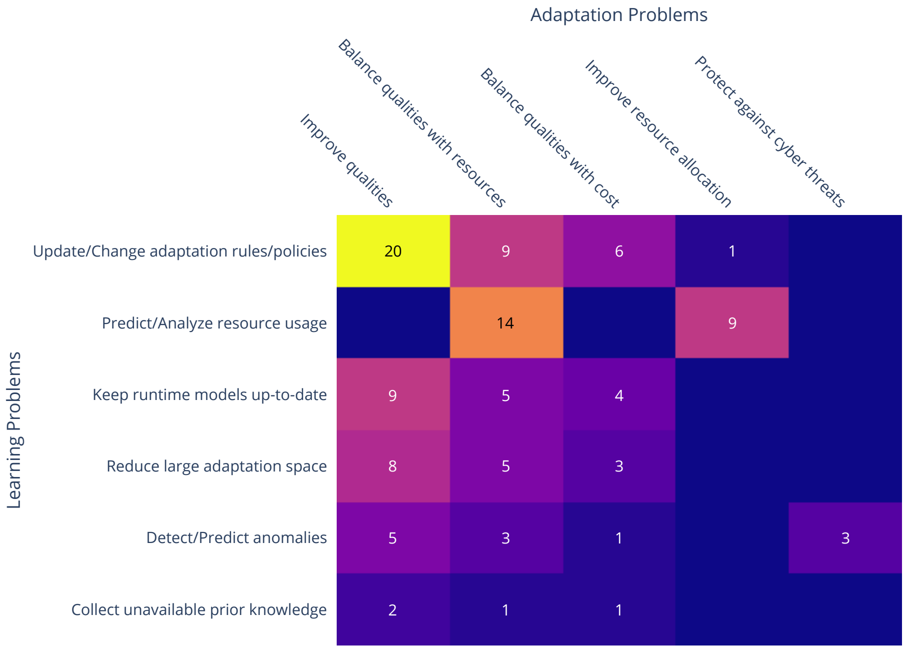


Figure 2.2: Learning problems versus adaptation problems.

Application domain. Table 2.4 shows the application domains where machine learning has been applied and evaluated in support of self-adaptation. The results show that learning has been applied in a wide variety of application domains, yet over 60% of the papers have studied and validated their work in three domains: cloud, client-server systems, and cyber-physical systems.

Application domains versus Adaptation problems. Figure 2.3 shows the adaptation problems solved in different application domains. Improving qualities is the main adaptation problem considered in all domains, except cloud. This can be expected as managing resources is vital in cloud applications, so balancing qualities with resources is the dominant adaptation problem in the cloud domain. Surprisingly, only a small fraction of the studies in the domains of cyber-physical systems and the internet-of-things consider resource management as part of the adaptation problem. Cost as an explicit factor in self-adaptation is mainly considered in client-server systems and service-based systems (in terms of the price to pay for using services).

Table 2.4: Application domains of the collected papers.

Application domain	Number
Cloud	33
Client-server system	18
Cyber-physical system	16
Internet-of-things	9
Service-based system	8
Robotics	7
Network management	6
Business process management	4
Remote data mirroring	3
Traffic management	3
Stream processing	2
Grid computing	1
Medical simulation	1
No specific domain	3

Application domains versus Learning problems. It is also interesting to take a look at the mapping of solved learning problems in different application domains, as shown in Figure 2.4. One key observation is that learning for updating and changing adaptation rules and policies is used in all domains, with cloud and client-server systems as the main domains. The latter resonates with the broad use of rule-based and policy-based techniques in the two domains. Learning to predict and analyze resource usage on the other hand is primarily used in the cloud domain only. Learning to keep runtime models up to date as well as reducing large adaptation spaces are also broadly used across domains (except in network management). Finally, detecting and predicting anomalies is primarily applied in network management, but sporadically also in a variety of other domains.

Answer to RQ1 - What problems have been tackled by machine learning in self-adaptive systems? We identified five types of adaptation problems where learning is applied: *improve qualities*, *balance qualities with resources*, *balance qualities with cost*, *improve resource allocation*, and *protect against cyber threats*. We identified six types of learning problems: *update/change adaptation rules/policies*, *predict/analyze resource usage*, *keep runtime models up-to-date*, *reduce large adaptation space*, *detect/predict anomalies*, and *collect unavailable prior knowledge*. The dominant case where learning is used to solve adaptation problems is updating and changing adaptation rules and policies to support improving qualities of the system. Learning to support

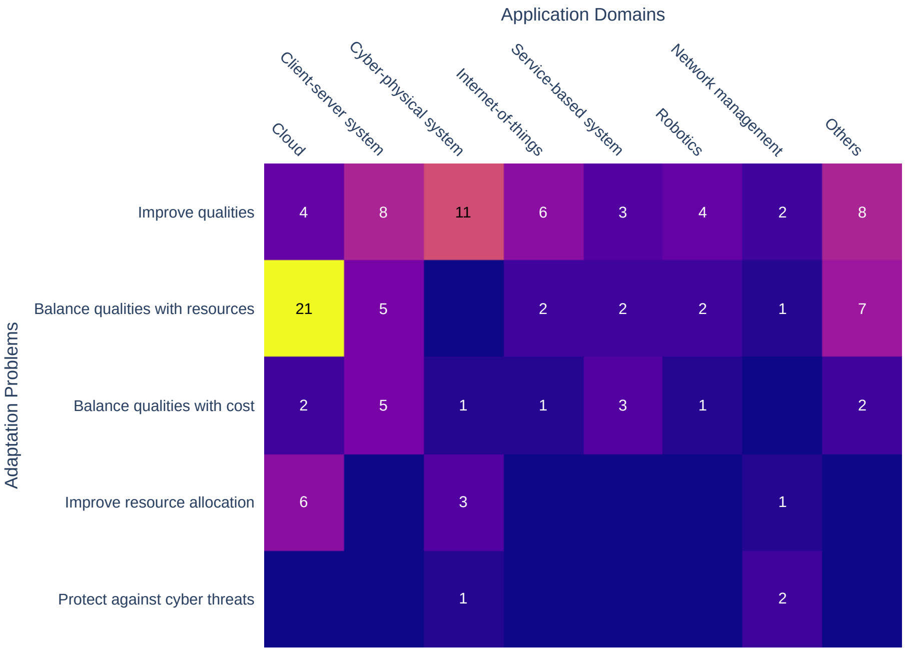


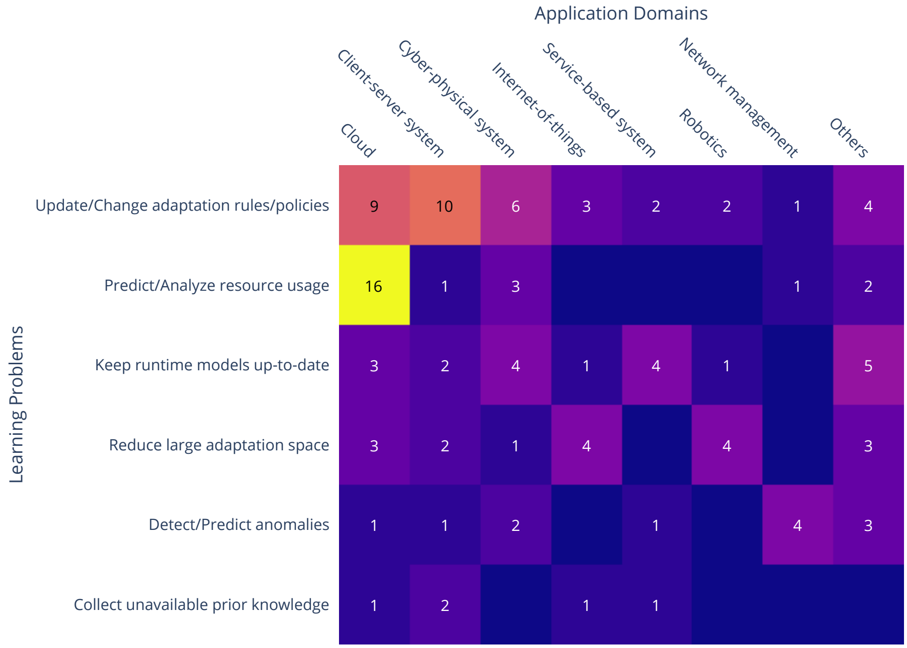
Figure 2.3: Adaptation problems versus Application domains.

> self-adaptation has been applied in a variety of applications, with cloud, client-server systems, and cyber-physical systems as the main domains.

2.5.3 RQ2: What are the key engineering aspects considered when applying learning in self-adaptation?

To answer this research question, we use the data items: MAPE stage(s) supported by learning (F9), Dimensions of learning methods (F10), and Learning methods used to support self-adaptation (F11).

MAPE functions supported by learning. Figure 2.5 shows the distribution of learning methods used to support MAPE functions. The diagram shows that learning is dominantly used to support the analysis part of the decision-making process in the feedback loop (in 83 studies). Thirty-six of these studies apply learning in support of analysis only. A typical example is [284], where machine learning is used to reduce large adaptation spaces such that only the relevant options need to be analyzed. Besides



Learning Problems	Cloud	Client-server system	Cyber-physical system	Internet-of-things	Service-based system	Robotics	Network management	Others
Update/Change adaptation rules/policies	9	10	6	3	2	2	1	4
Predict/Analyze resource usage	16	1	3				1	2
Keep runtime models up-to-date	3	2	4	1	4	1		5
Reduce large adaptation space	3	2	1	4		4		3
Detect/Predict anomalies	1	1	2		1		4	3
Collect unavailable prior knowledge	1	2		1	1			

Figure 2.4: Learning problems versus Application domains.

analysis, learning is often used to support planning of the feedback loop (57 studies). Fifteen of these studies apply learning in support of planning only. For example, [266] adopted an instance-based learning method to implement a hybrid planning approach that selects an optimal planning strategy among possible strategies. Learning has also been used to support monitoring (23 studies), in particular, to update knowledge models. Ten of these studies apply learning in support of monitoring only. For example, in [209], monitoring data is pre-processed using a lightweight classifier in order to learn optimization rules. We only identified one paper [267] where machine learning was used to support the execution function of the feedback loop (in combination with planning). In this paper, the authors used a classifier to choose the actuator that the system should use to adapt. The diagram shows that in a substantial number of papers (34 in total), learning supports both analysis and planning. An example is described in [136], where machine learning is applied to generate Event-Condition-Action rules that are evaluated and subsequently used to make adaptation decisions. A smaller fraction of the papers (six in total) combine learning to support monitoring and analysis. Finally, a small number of papers (seven in total) apply learning that spans monitoring, analysis, and planning. As an example, [27] proposed a proactive learner that supports monitoring, analysis, and planning by collecting context data, extracting required data for updating the learning model, and preparing a reasoning module for the decision-making process.

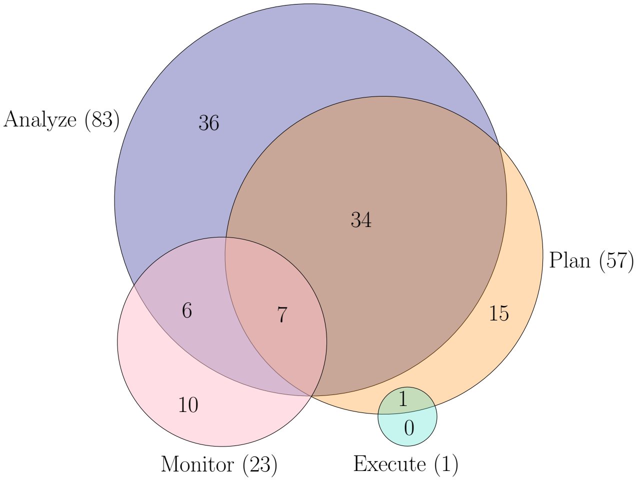


Figure 2.5: Distribution of learning methods supporting the MAPE functions.

Learning problems versus MAPE functions. Figure 2.6 maps the learning problems to MAPE functions. The heat map shows that all types of learning problems are tackled in support of monitoring, analysis, and planning. As can be expected, the dominant case is learning used for updating and changing adaptation rules and policies to support analysis and planning, followed by predicting and analyzing resource usage. Another observation is the importance of learning used to keep runtime models up-to-date in support of monitoring and analysis.

Learning dimensions. Figure 2.7 gives an overview of the learning dimensions that have been applied in the papers. Each layer of the sunburst diagram shows the options for one of the learning dimensions, where the same value of a dimension is represented by the same color. The numbers in the diagram are based on the number of learning tasks that are solved to support self-adaptation. For instance, six learning tasks have been solved using one or more learning methods that apply online, non-adversarial, passive, and unsupervised learning. In total 165 learning tasks have been solved in the 109 papers using a variety of learning methods. We zoom in on the concrete learning tasks solved by concrete learning methods below. The results show that a wide variety of combinations of dimensions are applied. The most popular learning methods used in self-adaptation apply supervised, passive, non-adversarial, and online learning. In terms of learning type, we observe that supervised learning dominates (71% of the learning tasks). On the other hand, only a small number of papers apply unsupervised learning (7% of the learning tasks), which is surprising for self-adaptive

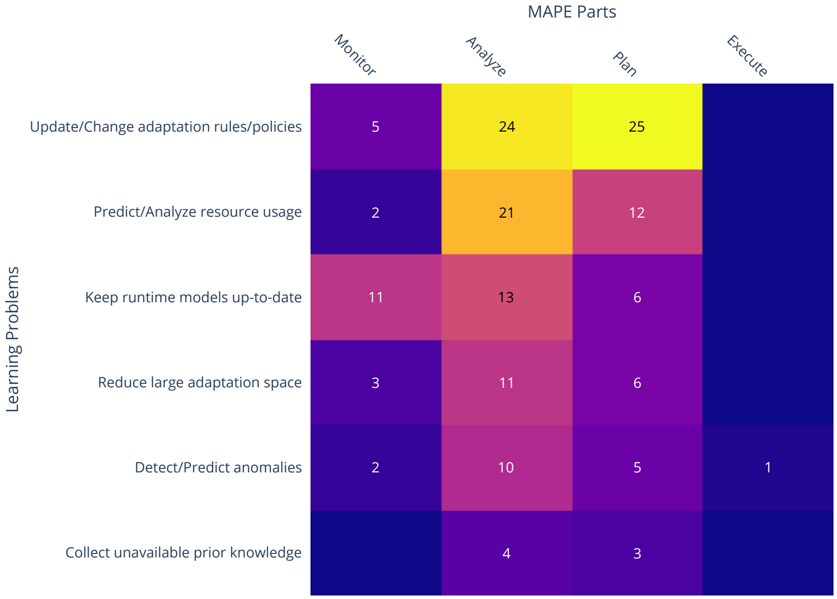


Figure 2.6: Learning problems vs. MAPE functions.

systems that aim at automation and dealing with uncertainties that may not have been anticipated [47]. Example papers that applied unsupervised learning are [101] and [399], in particular clustering-based learning techniques. Another observation is that we only encountered two papers that use adversarial learning, namely [194] and [220] that applied a game-theoretical learning approach.

Learning problems versus learning types. Figure 2.8 shows the mapping of learning problems to learning types. The results demonstrate that supervised learning is frequently used for all types of learning problems, but mostly to predict and analyze resource usage. Interactive learning is also used for all types of learning problems, yet updating and changing adaptation rules and policies together with keeping runtime models up-to-date make up 80% of these problems. Unsupervised learning is most frequently used for detecting and predicting anomalies, but nevertheless supervised is still used three times more to tackle this learning problem.

Learning methods used to support self-adaptation. Figure 2.9 shows an overview of concrete learning methods used in self-adaptation. Note that multiple methods may be used in a single study. Each layer of the sunburst diagram shows a different level of abstraction of the learning methods. The inner layer groups methods based on learning type, i.e., the dimension “unsupervised vs. supervised vs. interactive” (we further elaborate on dimensions of the learning methods below). The layer in the

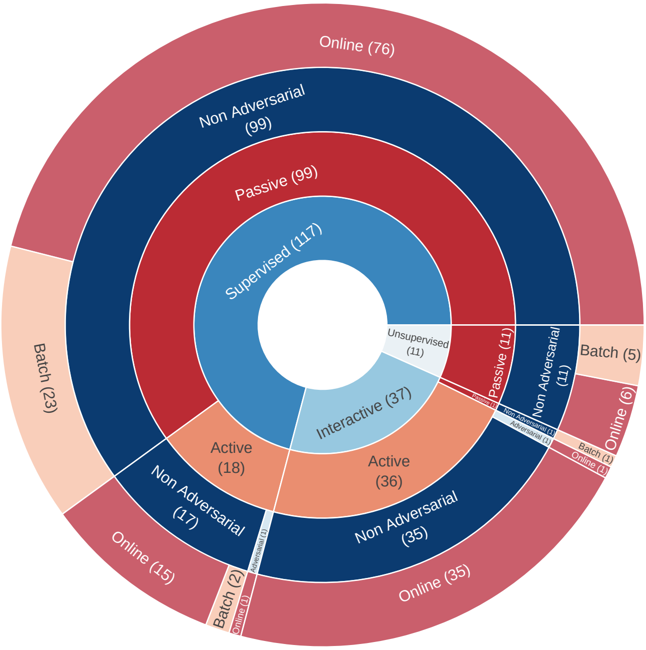


Figure 2.7: Distribution of learning dimensions used in machine learning for self-adaptive systems

middle groups learning methods based on common tasks of machine learning methods, i.e., classification, regression, reinforcement learning, clustering, and feature learning. This grouping is based on [310, 28]. Note that one method can be used for multiple tasks, for instance, support vector machines and deep learning have been used for both regression and classification. Finally, the outer layer shows concrete learning methods that were used in support of self-adaptation together with their frequencies (numbers between brackets). Areas marked with "Other tasks/methods" group other options. For instance, out of the 37 papers with interactive learning methods, 31 use reinforcement learning to solve a learning problem; the other six studies use other methods, such as hidden semi-Markov models and partially observable Markov decision processes. The diagram shows that the dominating learning method used in support of self-adaptation is model-free reinforcement learning (29 papers). An example is [13] which utilized fuzzy Q-learning to reason about new rules from the data collected at runtime. Other popular learning methods used in self-adaptation are support vector machines (15 times used; eight for regression and seven for classification), and traditional artificial neural networks and linear regression (both 14 times used). For instance, [105] exploited a support vector machine to detect network attacks in cyber-physical systems, [61] used

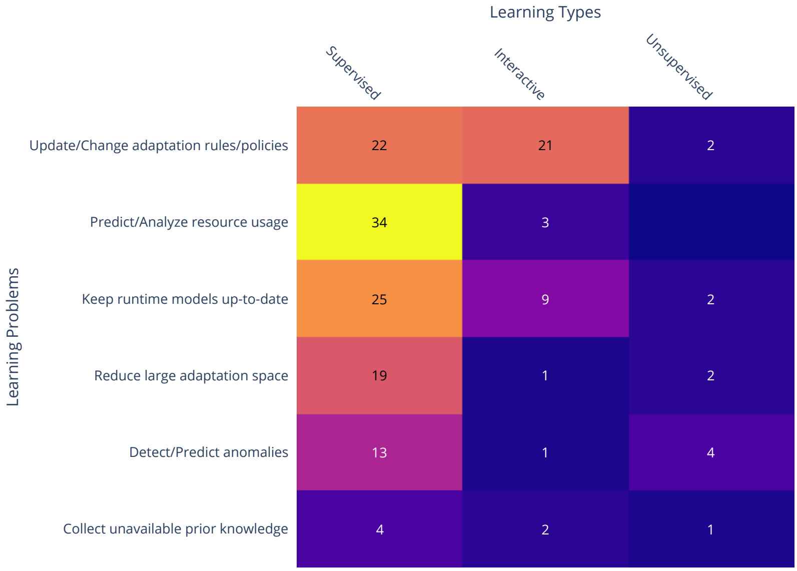


Figure 2.8: Learning problems vs. Learning types.

an artificial neural network to predict qualities of services such as response time and throughput of the system, and [80] applied linear regression to predict performance and power consumption.

Learning problems versus learning tasks. We also looked at the mapping between the learning problems in support of self-adaptation and the different types of learning tasks that need to be solved by the learners. Figure 2.10 shows an overview of this mapping. Regression is frequently used to solve all types of learning problems, but mostly to predict and analyze resource usage (36% of the problems solved with regression). Classification is also broadly used, with keeping runtime models up-to-date as the main learning problem (28% of the problems solved with a classifier). Updating and changing adaptation rules and policies is the primary learning problem solved by reinforcement learning (65% of the problems solved by a reinforcement learner).

Distribution of learning tasks over time and number of citations. To conclude, we look at the distribution of learning types over the years and the impact of the papers based on the number of citations they generated. For the latter, we used the citations from Google Scholar on February 2021.[3] Figure 2.11 plots the results. We observe that only seven papers have generated more than 100 citations; four that used supervised learning [128, 111, 107, 404], two that used interactive learning [340, 45], and one

[3] We used Google Scholar as it is widely used, but we acknowledge its limitations, such as the inclusion of self-citations.

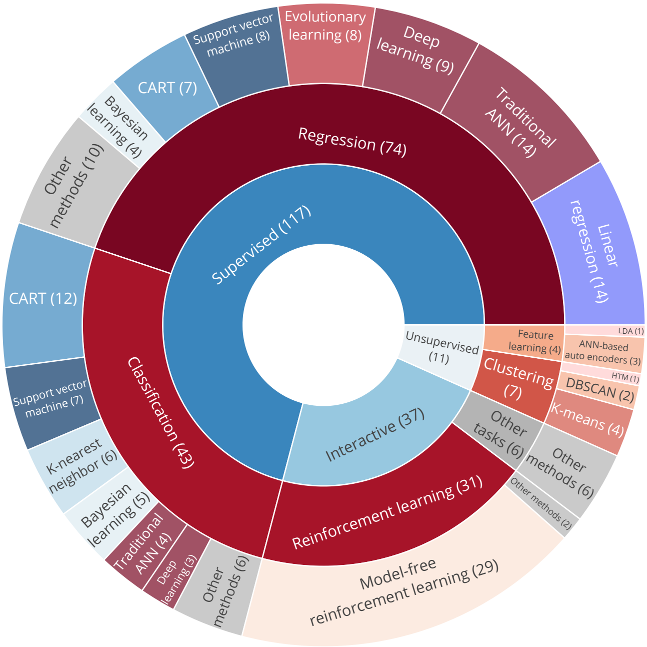


Figure 2.9: Distribution of learning types, tasks, and methods used in self-adaptive systems. ANN refers to Artificial Neural Network, HTM to Hierarchical Temporal Memory, and LDA to Latent Dirichlet Allocation.

paper that used unsupervised learning [128]. Overall, none of the three learning types seem to have generated clearly more impact (normalized number of citations[4] 2007-2019 for supervised learning avg 4.8, std 7.0; interactive learning avg 4.4, std 7.4; and unsupervised learning avg 6.5, std 11.4.). The plot shows that supervised and interactive learning have been used frequently over the full period from 2007 to 2019. Yet, since 2016, we note an increase in the use of supervised learning and a decrease in interactive learning. Remarkably, after some small attention to the use of unsupervised learning in the period 2010 to 2013 (three studies), we observe an increase of this type of learning in the last three years, from 2017 to 2019 (eight studies).

Answer to RQ2 - What are the key engineering aspects considered when applying learning in self-adaptation? Machine learning is primarily used to

[4] The number of citations for each paper has been normalized by past years since it was published, i.e., normalized number of citations of paper = Google Scholar citation of the paper in 2021/(2021 - publication year of the paper).

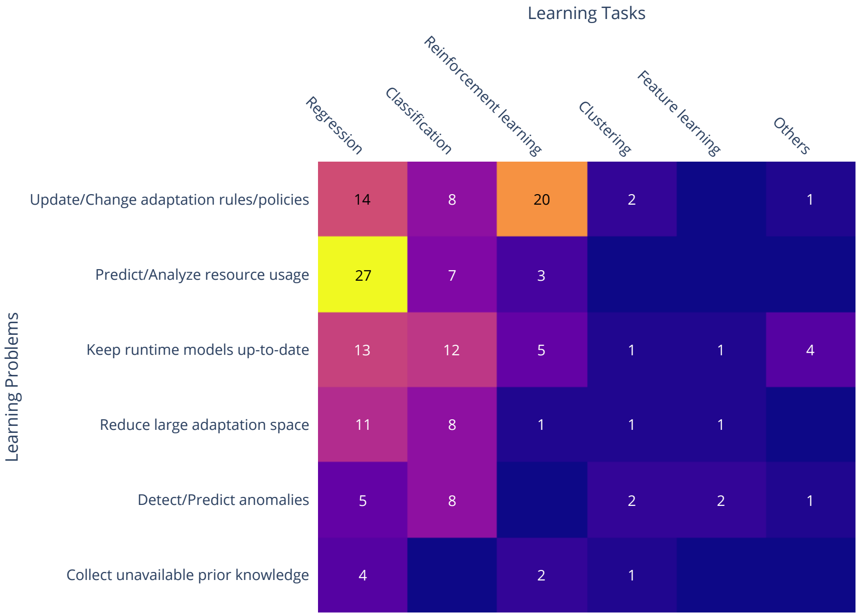


Figure 2.10: Learning problems vs. Learning tasks.

support analysis and planning in self-adaptive systems. The majority of the papers use supervised or interactive learning; these learners typically exploit the results of runtime analysis and observed effects of applied adaptations to learn. The most frequent problem of self-adaptation delegated to learning is updating and changing adaptation rules and policies (primarily solved using regression and reinforcement learning). Other important learning problems are predicting and analyzing resource usage (primarily solved using regression) and keeping runtime models up-to-date (primarily solved using regression and classification). The most popular learning method that is applied in self-adaptive systems is model-free reinforcement learning used for updating and changing adaptation rules and policies. Adversarial learning on the other hand is understudied, while this approach has a huge potential to deal with security concerns in self-adaptation. We observe that supervised and interactive learning have been used frequently over the years. Unsupervised learning on the other hand has only been used in a limited number of papers. Yet, this approach supports detecting novelty in data without any labeling, which can play a key role in managing complex types of uncertainty. None of the three types of learning has clearly generated more impact over the years.

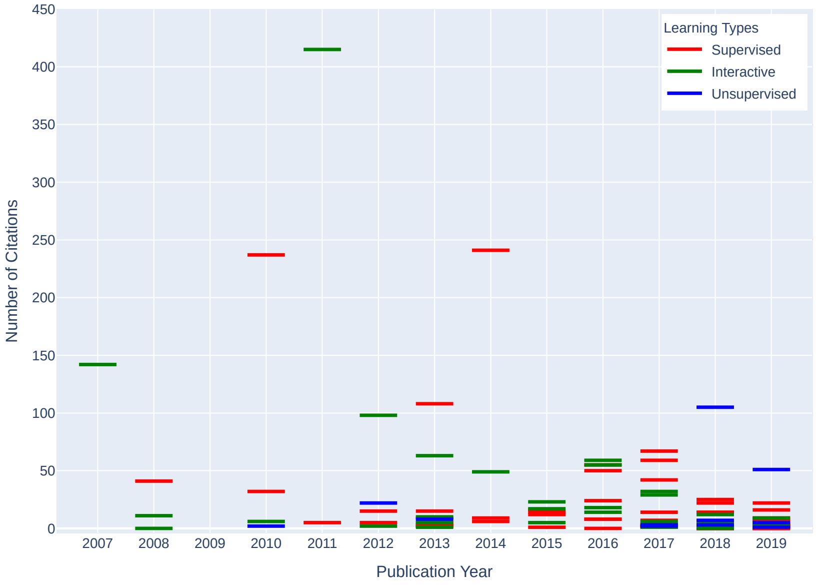


Figure 2.11: Distribution of learning types through years and the number of citations (Google Scholar 2/2022). Each study is represented by a tiny horizontal bar as indicated in the key. Studies of the same learning type with citation counts close to each other form thicker bars.

2.5.4 RQ3: What are open challenges for using machine learning in self-adaptive systems?

To answer this research question, we analyze data items: Limitations (F13) and Challenges (F14).

Limitations. Table 2.5 lists the limitations reported in the papers. As illustrated in the extracted quality scores, only a limited number of papers reported limitations of the applied learning methods. The results show that a variety of limitations of the learning methods have been reported. The most frequently reported limitation is the limited scalability of the proposed learning approach. Other reported limitations relate to impact on qualities, in particular performance and reusability, the scope in terms of uncertainty and guarantees that can be provided, and the need for expertise of humans to tune parameters.

Challenges. Table 2.6 lists the challenges reported in the papers. In total 21 papers (19%) discussed challenges. Consequently, the reported challenges do not represent consensus or the importance of the challenges. However, most of the challenges apply

Table 2.5: Reported limitations of the learning methods applied in the papers.

Scalability	Learning approach is not scalable	6	[48, 267, 61, 44, 340, 339]
	No test of scalability due to data sensitivity	1	[302]
Performance	High computation time	3	[212, 189, 340]
	High computational load for feedback loop	1	[146]
	Slow convergence	1	[241]
Reusability	Solution is domain specific	2	[129, 146]
	Applicable for optimization problems only	1	[235]
Uncertainty	Cannot handle new situations	2	[127, 331]
	Cannot detect sudden changes	1	[339]
Guarantees	Optimization without satisfying all SLAs	1	[277]
	Might be trapped in a sub-optimal solution	1	[276]
Design	Need parameter tuning	4	[241, 127, 128, 182]

to many other papers; yet, these authors have not explicitly mentioned them. The challenges are organized into five groups. Learning performance challenges primarily relate to timing aspects of learning. Learning effect challenges relate to uncertainties in terms of the effects of using learning in self-adaptation. Domain-related challenges are concerned with the characteristics of domains and the transfer of solutions to other problems. Policy-related challenges relate to the ability of learning methods to support the principles and rules for decision-making in self-adaptation. Finally, goal-related challenges relate to the need for machine learning techniques to support adaptation in practical systems that are characterized by multiple, possible evolving goals. We now zoom in on a few of the interesting open challenges and outline potential starting points to tackle them.

An open challenge in machine learning for self-adaptive systems is effect uncertainty [63, 276]. Effect uncertainty refers to uncertain effects on the system that may occur when a learner selects a configuration or an adaptation plan that is applied to the system. Relying on the results of machine learning comes with some degree of (statistical) uncertainty that may affect the decision-making of a self-adaptive system. Detecting and handling this type of uncertainty is an open challenge. Note that this challenge is not specific to machine learning methods. However, we raise it here as many studies have adopted machine learning methods for proactive decision-

Table 2.6: Open challenges for learning in self-adaptation reported in the papers.

Learning Performance	Balance time and accuracy	3	[323, 256, 266]
	Handle oscillations in early learning stages	1	[182]
Learning Effect	Understand the effect of learning on adaptation decisions over time	3	[63, 276, 401]
	Guarantees on results of machine learning	1	[284]
Domain-Related	Handle sudden changes	2	[270, 278]
	Handle open world changes	1	[355]
	Balancing diverse sources of input data	1	[328]
	Extend to other application domains	1	[80]
	Define similarity measures to transfer to other planning problems	1	[266]
Policy-Related	Deal with conflicting policies	2	[270, 177]
	Improve policy evolution speed	1	[171]
Goal-Related	Handle multiple goals	3	[184, 111, 284]
	Dynamically define utility function	1	[313]

making, where effect uncertainty gets more challenging by the uncertainty introduced by learning methods.

Learning about open-world changes is another open challenge in self-adaptive systems. Open world changes have been studied in the field of machine learning under the umbrella of “lifelong machine learning” [341], in particular in relation to dealing with new learning tasks. A lifelong machine learner relies on an online learning pipeline that exploits historical knowledge to evaluate and update an existing learner to deal with new tasks. It may be possible to exploit such an approach to support a data-driven self-adaptive system with changes that were not fully anticipated.

In large-scale self-adaptive systems, the feedback loop’s monitor component may be distributed over many different nodes (for instance sensor nodes) deployed on the managed system or in the environment. An example is described in [391] where distributed monitoring components are used in online games that run on a client-server infrastructure. A crucial aspect of the sensed data on the decision-making for adaptation is the impact of heterogeneous sensor data [328], which may dynamically change. Automated weighting of heterogeneous data sources based on the current situation of the system to assure proper decision-making for adaptation is an open challenge.

One of the characteristic use cases of machine learning is reducing large adaptation spaces to support efficient analysis of different configurations based on model checking at runtime, see for instance [284]. An open problem is to understand the impact of the learning process on the results of the model checker as this will affect the guarantees of the decisions made by the feedback loop. Such understanding will not only provide bounds on the expected impact of learning on the guarantees for decision-making in self-adaptive systems, but it will also pave the way to dynamically balance the guarantees that are required with the resources that are available to provide them.

Transfer learning focuses on storing knowledge obtained from solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize anomalies in one type of communication network could then be exploited to recognize anomalies in another type of network. Transfer learning can help self-adaptation by reducing the cost of continuous training and data collection [184]. However, this study highlights that transfer learning has rarely been used in self-adaptation so far. Inspiration to tackle this open challenge is provided [171].

Another important open challenge when using learning is handling multiple goals. Different approaches exist to deal with multiple goals in self-adaptation, such as utility functions [313], (semi-)ordered rules [284], and multi-objective functions [111]. A key issue of handling multiple goals is balancing time and resource usage with finding a (close to) optimal solution. Hence, exploring the use of machine learning methods for efficient multi-objective optimization with guarantees on the precision of the results is an open challenge in machine learning for self-adaptation.

Answer to RQ3 - What are open challenges for using machine learning in self-adaptive systems? Based on the reported limitations and challenges, we identified three broad categories of open challenges. The first category is about quality-related challenges. These include the scalability and performance of learning, and the reusability of solutions. The second category is about effect-related challenges. Central here are guarantees when learning is applied to support the feedback loop, uncertainties caused by learning, and the effects of learning on decision-making. The third category is about design challenges. These include challenges related to the domain at hand, and policy and goal-related challenges.

Table 2.7: Themes of challenges with concrete focus as reported in the papers.

Challenge Theme	**Concrete Focus**
Qualities	Scalability of learning, remove performance penalty
Uncertainty	Monitor uncertainty, detect novelty, support open world
Goals	Deal with changing goals, conflict of goals, new types of goals
Guarantees	Ensure quality goals, avoid sub-optimality, support explainability
Domain / Design	Deal with parameter tuning, transfer solutions, reusability of solution

2.6 Insights Derived from the Study and Threats to Validity

Based on the insights derived from this systematic literature review we start this section by outlining an initial design process for applying machine learning in self-adaptive systems. Then, we discuss a number of remarkable observations of the survey that open interesting opportunities for future research. Finally, we discuss threats to the validity of the research presented in this chapter.

2.6.1 Towards a Design Process for Using Machine Learning in Self-Adaptive Systems

...

2.6.2 Opportunities for Future Research

The reported limitations of learning methods applied in self-adaptation (Table 2.5) as well as the open challenges for this area (Table 2.6) identify a number of shortcomings of existing learning approaches and highlight demands that may require the use of other or new learning methods to support self-adaptation. Table 2.7 summarizes the themes of the reported challenges.

The themes in Table 2.7 take the stance of the stakeholders of self-adaptive systems, looking from the perspective of the characteristics, demands, and open problems of these systems. We complement this view now with a set of additional opportunities for future research. To that end, we took a step back and explored prospects for advancing the field by looking at opportunities provided by learning methods. In particular, we

Table 2.8: Additional opportunities for future research driven by learning methods.

Learning Method	**Concrete Opportunities**
Unsupervised learning	Detecting new structures in complex data, support other learning methods
Active learning	Involve stakeholders in decision-making, reduce learning cost, increase speed of learning
Adversarial learning	Improve rules and policies, detect anomalies
Other learning methods	Detection of novel phenomena in environment, synchronize execution workflows in complex settings

looked at machine learning methods that received less or no attention in existing work, and explored how self-adaptation may benefit from further investigation into the use of these learning methods. Table 2.8 summarizes the opportunities.

Only a small fraction (roughly 10%) of the papers apply unsupervised learning methods. This is remarkable given that one of the key drivers for applying self-adaptation is automating tasks in systems that are subject to uncertainty [362]. Since unsupervised learning methods can work independently of external input and do not require labeled data, we observe an interesting opportunity here to further explore unsupervised learning in self-adaptation. Unsupervised learning methods can be used as independent learning techniques to support self-adaptation; one interesting use case is the detection of new structures in complex high-dimensional data [106]. Unsupervised learning methods can also be used to support supervised or even interactive learning methods. A good example here is auto-encoders that have been used to increase the precision of other learning methods by providing a denser representation of data [384, 267, 128].

On the other hand, most papers apply passive learning. Active learning methods [309] interact with the environment or stakeholders to obtain the desired outputs at new data points. This helps to improve the performance of the machine learner by exploring the most informative data. In the context of self-adaptation, applying active learning provides an opportunity to involve stakeholders in the decision-making process, which has been highlighted as a key aspect of establishing trust [372]. Active learning can be exploited to reduce the learning cost and increase the convergence speed of learning. It can also be particularly useful to learn updating goals through interaction with stakeholders. In this way, the system can gradually learn essential knowledge of the stakeholder.

Adversarial learning aims at enabling a safe adoption of machine learning techniques in adversarial settings [216]. An adversarial machine learner tries to fool a learning model by supplying deceptive input. Adversarial learning can be particularly useful in domains that are sensitive to privacy and security issues, e.g., for signature detection and biometric recognition. Our study shows that only two papers have exploited non-adversarial learning in self-adaptation that both adopt a game-theoretical learning approach. This observation opens opportunities to exploit various adversarial learning methods. One example is the use of generative adversarial networks to improve rules and policies in rule- and policy-based systems, or detect new anomalies for self-protection.

The majority of papers (85%) apply learning to support decision-making in self-adaptation, i.e., the analysis and planning functions. The main use cases are updating and changing rules and policies and predicting and analyzing resource usage. Only a limited number of papers (14%) apply learning to support monitoring, and here the main use case is keeping runtime models up-to-date. Only a single paper applies learning to support execution. This clearly opens opportunities for other use cases. One interesting opportunity is to use learning to support the detection of novel phenomena in the environment that have an effect on the self-adaptive system. Tackling this type of uncertainty is broadly seen as a key challenge in self-adaptive systems [47]. Another opportunity is to exploit learning in the execution of adaptation plans. In complex settings, for instance, in large-scale applications with distributed feedback loops, the workflow and synchronization of adaptation actions are often very difficult to establish manually. Machine learning can then be exploited to learn the best possible execution of the workflow under changing conditions.

2.6.3 Threats to Validity

We list the main threats to the validity of this study and the measures we took to mitigate them.

Internal validity: refers to the extent to which a causal conclusion based on a study is warranted. Potential bias of reviewers is a common validity threat of literature reviews. For instance, a reviewer may be biased in the interpretation of fundamental concepts, i.e., machine learning and self-adaptive system. To mitigate this risk, we took two measures. First, the three researchers involved in the study defined a protocol before starting the review process to clarify the definition of fundamental concepts and the process to follow. Second, the three researchers were involved in the selection of papers, the data collection, and the analysis. A subset of the papers was handled independently by two researchers. The decisions on including or excluding papers and collecting data from the selected papers were based on an agreement between the two researchers. In case of disagreement, a third researcher was consulted, and after discussion, a decision was made in consensus.

External validity: refers to the generalizability of findings. Applied to this study, this threat is about the generalization of the outcome and conclusions of the literature review. By limiting the automatic search to three online libraries, we may have missed some papers. To mitigate this threat, we applied the search string to the main libraries for publishing research in this area. This aligns with other literature reviews. In addition, we crosschecked that established venues for publishing papers in self-adaptation are covered. Furthermore, the search string we used may not provide the right coverage of papers. We mitigated this threat by starting the search process with pilot searches to define and tune the search string by collecting data from specific venues via the scientific search engines and comparing the results with manual inspection of the papers of the searched venues.

Construct validity: refers to the degree to which a study measures what it aims to measure. Here, the quality of reporting of studies may be a threat as this element affects the validity of the collected data. To anticipate this threat, we extracted data about reporting quality. The analysis of this data shows that the quality of reporting of the papers is of sufficiently good quality. This result provides a solid basis to derive conclusions from extracted data. Moreover, to mitigate this threat, we excluded all short papers and papers that do not provide a minimum level of assessment. For instance, we excluded [136] although the topic is relevant for our study, but this is a short paper. Similarly, we excluded [312] since this regular paper does not provide a sufficient level of assessment.

Reliability: refers to assuring that the research findings can be replicated by another researcher. Here bias of researchers is also a potential validity threat. As explained above, to mitigate this threat, we defined a detailed protocol that provides the necessary guidelines for performing the different steps of the study. Multiple researchers did the paper selection, data extraction, and analysis. Another technical threat concerns the methods used to collect papers from search engines. For example, a search engine may change the operator to select any paper that complies with a query from a star (*) operator to an “ANY” operator (ignoring the initial version of the operator). To anticipate this threat, all the review material is available online, enabling replication of the study.

2.7 Conclusion

This literature review aimed at shining a light on the state of the art of using machine learning in self-adaptive systems. The review confirms the rapidly growing research interests in this area. We identified six types of problems in self-adaptation that are solved by using machine learning: updating and changing adaptation rules and policies, predicting and analyzing resource usage, keeping runtime models up-to-date, reducing large adaptation spaces, detecting and predicting anomalies, and collecting unavailable prior knowledge. These problems are primarily solved to support analysis and planning in self-adaptation. Supervised and interactive learning dominate, primarily to solve regression, classification, and reinforcement learning tasks. The reported limitations and challenges relate to quality properties when learning is used in self-adaptation, the effects of learning on decision-making, and managing challenging aspects of the domain at hand.

From the data analysis, we identified an initial process to support designers that want to apply machine learning in the realization of self-adaptive systems. We defined an open process that can be extended with new knowledge as we learn more about applying learning in self-adaptive systems.

Finally, we outlined a number of interesting opportunities for further research in this area, in particular, managing effect uncertainty, dealing with open world changes, dealing with distribution and heterogeneity of data, determining the bounds on guarantees for the adaptation goals implied by the use of machine learning, exploiting transfer learning to related problems, and finally dealing with more complex types of adaptation goals. We hope that the results of this systematic literature review will inspire researchers to tackle these and other problems in this fascinating research area.

Chapter 3

Reducing Large Adaptation Spaces Using Machine Learning

Publication details. This chapter is based entirely on a journal publication in the Journal of Systems and Software (JSS) [289].

Personal contributions. Conceptualization (65%), Methodology (75%), Software (80%), Validation (80%), Formal analysis and interpretation results (80%), Writing (60%), Visualization (90%).

Positioning. Modern software systems often have to cope with uncertain operation conditions, such as changing workloads or fluctuating interference in a wireless network. For software systems with a large number of adaptation options, deciding which option to select for adaptation may be time-consuming or even infeasible within the available time window to make an adaptation decision. This is particularly the case when rigorous analysis techniques are used to select adaptation options. One technique to deal with the analysis of a large number of adaptation options is reducing the adaptation space using machine learning. In this Chapter, we present ML2ASR+. Central to ML2ASR+ is a configurable machine learning pipeline that supports effective analysis of large adaptation spaces. We evaluate ML2ASR+ for two applications: an Internet-of-Things application and a service-based system. The results demonstrate that ML2ASR+ can be applied to deal with different types of goals and is able to reduce the adaptation space to make adaptation decisions with over 90 %, with negligible effect on the realization of the adaptation goals.

3.1 Introduction

Engineering modern software systems is complex. One of the important factors that underlie this complexity is the dynamic and complex environment in which systems need to operate, requiring the systems to deal with uncertain conditions that are often difficult to predict before they are in operation [153]. These uncertainties may jeopardize the system's goals. Network interference can for example affect the availability of the system if not properly dealt with.

To mitigate such uncertainties, self-adaptation has become prevalent in modern software systems [64, 301, 361]. Self-adaptation enhances a software system with a feedback loop mechanism that monitors the system and its environment, resolves the uncertainties, and adapts the system to maintain its goals, or degrades gracefully if necessary. Hence, self-adaptive systems consider system goals as first-class runtime entities; we refer to these goals as *adaptation goals*. Adaptation goals commonly refer to quality properties of the system [370].

In this chapter, we apply architecture-based adaptation [263, 142, 210, 374, 242], where the feedback loop implements four functions: Monitor-Analyze-Plan-Execute (MAPE in short) [193]. The MAPE functions are centered around Knowledge that typically includes various forms of runtime models [29], such as architectural models of the managed system and environment, goal models, and parameterized quality models that allow predicting qualities of different system configurations. We focus on uncertainties that can be represented as parameters of runtime models, e.g., stochastic automata or Markov models. The values of the uncertainty parameters are updated by the monitor function that monitors the system and its environment. We consider three types of adaptation goals: *threshold goals* that require a system to keep a system property above/below a given threshold, *optimization goals* that require a system to minimize or maximize a system property, and *setpoint goals* that require a system to keep a system property at a given value or as close as possible to it. An example of a threshold goal for a client-server system is to keep the failure rate of service invocations below a given threshold, an example of an optimization goal is to minimize the cost of operation, and an example of a setpoint goal is to keep the response time of service invocations at a required level.

Our particular focus is on the analysis function of the MAPE loop that (1) determines whether the current configuration complies with the adaptation goals, and if this is not the case, (2) predicts the qualities of alternative configurations. An alternative configuration is a configuration that can be reached from the current configuration by applying one or more adaptation actions. We refer to the alternative configurations as *adaptation options*, and the set of all adaptation options as the *adaptation space*. A common technique used to analyze the adaptation space is formal modeling and verification. Formal models represent the system and its environment from the angle of one or more quality properties. These quality models are parameterized. One set of parameters allows the instantiation of the models for a particular configuration of the system. Another set of parameters represents uncertainties that are instantiated based on the actual conditions of the system. During analysis, the parameters of the quality models are instantiated. Commonly used analysis techniques are model checking [45, 51] and runtime simulation [176, 369, 371]. Based on the analysis results, a decision can then be made to adapt the system compliant with the quality goals. Recently, there is an increasing use of machine learning techniques to support the adaptation functions [151].

For systems with a limited number of adaptation options, i.e., small adaptation spaces, the analysis can be done fairly quickly ensuring that adaptation decisions are made within the available time frame to handle the dynamics of the system properly. However, for larger and more complex self-adaptive systems, the time required for analysis may dramatically increase and formal assessment of the whole adaptation space may not be feasible in such situations.

Different techniques have been proposed to deal with the problem of analyzing large adaptation spaces. E.g., Cheng et al. [65] applied search-based software engineering techniques to generate and analyze models of dynamically adaptive systems in order to deal with uncertainties both at development time and runtime. Our particular focus in this chapter is on a conceptually different technique that relies on machine learning to support the reduction of adaptation spaces, see e.g., [284, 349, 88]. While promising, current approaches do not provide a systematic solution with first-class support for reducing large adaptation spaces during operation that is able to handle different types of goals.

This chapter contributes ML2ASR+, short for “Machine Learning to Adaptation Space Reduction Plus”, a novel approach for reducing large adaptation spaces.[1] ML2ASR+ relies on classic supervised machine learning techniques, particularly classification and regression. We evaluate ML2ASR+ on two self-adaptive systems in distinct domains with varying sizes of adaptation spaces. We compare the approach with a reference approach that exhaustively analyzes the whole adaptation space, and a state-of-the-art learning-based approach that we developed in previous work [349], called DLASeR, which exploits deep neural networks to achieve adaptation space reduction. In addition, we perform a sanity check where we compare ML2ASR+ with an approach that randomly selects a subset of adaptation options in each adaptation cycle.

The remainder of this chapter is structured as follows. Section 3.2 presents the state of the art and pinpoints the problem we tackle in this chapter. In Section 3.3, we explain the model we use for self-adaptation in this chapter, we elaborate on the different

[1] An initial version of the approach that was limited to only one type of adaptation goal was denoted ML2ASR [284]. The “+” emphasizes that ML2ASR+ significantly extends ML2ASR. We elaborate on this in the state-of-the-art overview in Section 3.2.

types of adaptation goals, and we introduce a running example. Section 3.4 then describes the core contribution: ML2ASR+, with its runtime architecture and workflow. Section 3.5 explains the metrics that we use for evaluating ML2ASR+. In Section 3.6, we evaluate ML2ASR+ for two application domains. Section 3.7 elaborates on the results, presents insights, and discusses threats to validity. Finally, we wrap up and conclude in Section 3.8.

3.2 State of the Art and Problem Description

We have divided the state-of-the-art into three main areas of research. For each area, we summarize a number of representative efforts and we conclude with the open problems in the area. From this analysis, we pinpoint the research problem we tackle in this chapter.

3.2.1 Machine Learning to Support the Analysis of Large Adaptation Spaces

We start with approaches that apply machine learning to deal with the analysis of large adaptation spaces.

The FUSION framework learns the impact of adaptation decisions on the system's goals [107]. The approach utilizes M5 decision trees to learn the utility functions that are associated with the qualities of the system. The results show a significant improvement in analysis. FUSION targets the feature selection space, focusing on proactive latency-aware adaptations relying on a separate model for each utility. Chen et al. [62] study feature selection and show that different learning algorithms perform significantly different depending on the types of quality of service attributes considered and the way they fluctuate. The work is centered on an adaptive multi-learners technique that dynamically selects the best learning algorithms at runtime. The focus of this work is also on features instead of adaptation options. Metzger et al. [250] apply online learning to explore the adaptation space of self-adaptive systems using feature models with an emphasis on the adaptation and evolution of adaptation rules.

Jamshidi et al. [179] present an approach that learns a set of Pareto optimal configurations offline that are then used at runtime to generate adaptation plans. The approach reduces adaptation spaces, while the system can still apply model checking to quantitatively reason about adaptation decisions. Camara et al. [53] use reinforcement learning to select an adaptation pattern relying on two long short-term memory (LSTM) deep learning models. The focus is on the use of runtime quantitative verification, with support for threshold goals. Thallium exploits a combination of automated formal modeling techniques to significantly reduce the number of states that need to be considered with each adaptation decision [329]. Thallium addresses the adaptation state explosion by applying utility bounds analysis. Diallo et al. [88] present a framework consisting of a MAPE-K feedback loop with an explainable AI module to tackle the issue of reducing adaptation spaces. Their framework leverages convolutional neural networks to efficiently reduce adaptation spaces, alongside using explainable AI to build trust in the system.

In our initial work [284] we applied classification and regression to reduce large adaptation spaces. The work also only considered threshold goals. In [349], we investigated the use of deep learning to reduce the adaptation space of self-adaptive systems. That work focused on handling threshold and optimization goals only.

Open problems. Several approaches that apply machine learning to enhance the runtime analysis of self-adaptive systems look at a coarse-grained level of system features rather than a fine-grained level of adaptation options. The approaches that look at the reduction of large adaptation spaces propose solutions that inherently mix the reduction of the adaptation space with the way analysis is performed, while other approaches (including our own earlier work) only consider specific types of adaptation goals. In conclusion: existing approaches in this area do not provide explicit support for adaptation space reduction, or they cover only specific types of adaptation goals.

3.2.2 Reinforcement Learning to Support Decision-making in Self-Adaptation

We look now at reinforcement learning techniques used to support decision-making in self-adaptation.

Porter et al. [273] study the dynamic composition of software elements using a reinforcement learning algorithm, covering the analysis and planning stages in the self-adaptation process. The approach reduces the adaptation space to a single option, hence integrating adaptation space reduction and decision-making. Idziak et al. [173] study different machine learning algorithms to deal with the so-called virtual machine placement problem. These algorithms similarly take over the analysis and planning stages of the self-adaptation process. Lui et al. [234] use a reinforcement learning algorithm to improve resource efficiency in autonomous electrified vehicles. Similarly to the previous two works, the approach reduces the adaptation space to a single option that is used for decision-making. Bu et al. [40] and Metzger et al. [251] propose strategies to explore the adaptation options in reinforcement learning algorithms.

Open problems. While relying on different learning techniques compared to the approaches discussed above, the approaches proposed in this area also inherently integrate the reduction of adaptation spaces with the decision-making to select the best adaptation options for the goals at hand. An advantage of relying on reinforcement learning to realize this integration is that it does not require a (formal) model of the system, which may be a benefit if creating such a model is problematic. In conclusion: the proposed approaches do not support a separation of concerns between an explicit and tuneable reduction of adaptation spaces and the decision-making of selecting the best option.

3.2.3 Efficient Analysis in Self-Adaptive Systems

A number of approaches have been proposed to enhance the efficiency of analysis in self-adaptive systems.

Filieri et al. [130] propose an approach to generate a static set of expressions from a reliability model with a set of requirements. By using these expressions more efficient analysis is possible at runtime. That approach targets formal models based on PCTL (Probabilistic Computation Tree Logic). Calinescu et al. [46] combine compositional verification with model checking to effectively adapt large-scale systems. The authors employ assume-guarantee reasoning to reduce the cost of analyzing system properties, compared to infeasible exhaustive model checking approaches. Gerasimou et al. [144] explore caching, lookahead, and nearly-optimal reconfiguration techniques to optimize the response time and overhead of Runtime Quantitative Verification to enhance scalability.

Ghahremani et al. [149] look at ways of reducing the cost of realizing self-adaptation in self-healing systems by combining utility-driven approaches with rule-based adaptation. Moreno et al. [257] present an approach for proactive latency-aware adaptation that relies on stochastic dynamic programming to enable more efficient decision-making. Experimental results show that this approach is close to an order of magnitude faster than runtime probabilistic model checking to make adaptation decisions while preserving the same effectiveness.

El-Kassabi et al. [191] use a deep neural network to support proactive system adaptation by providing predictions of cloud resource usage. The predictions enable the suggestion of adaptation decisions to anticipate future quality of service violations. Di Sanzo et al. [87] equip a client-server application with a framework that provides proactive management of the application. The framework exploits a multitude of machine learning methods such as linear regression and support vector machines to build and use failure prediction models at runtime. The predictions are then used to proactively adapt the system before failures take place. Ghahremani et al. [148] evaluate machine learning algorithms for the prediction of system utility in adaptive systems, without relying on detailed system information.

Open problems. The approaches proposed in this area can be structured into three groups. The first group focuses on improving the verification process. These approaches do not deal with the problem of adaptation space reduction but can be combined with an approach for adaptation space reduction. A second group focuses on alternative solutions to enhance the efficiency of decision-making in self-adaptive systems. Yet, as with other related approaches discussed above, these approaches inherently integrate an implicit reduction of adaptation spaces with the decision-making to select an adaptation option. A third group applies machine learning techniques to make predictions of qualities and other properties to support the decision-making process. These solutions are complementary to the problem of adaptation space reduction. In conclusion: two groups of related efforts do not solve the problem of adaptation space reduction, but can be combined with approaches to reduce the adaptation space in order to enhance the efficiency of analysis; another group of related efforts does not separate the reduction of adaptation spaces with decision-making.

3.2.4 Research Problem

The analysis of the related work highlights the need for systematic approaches that provide explicit first-class support for adaptation space reduction while covering different types of goals. To that end, we formulate the following research question that we tackle in this work:

> *How can machine learning be used to reduce large adaptation spaces of self-adaptive systems with different types of adaptation goals to perform more efficient analysis without compromising the goals?*

To answer the research question, we propose ML2ASR+, a novel approach for adaptation space reduction. Leveraging classification and regression, ML2ASR+ offers a modular approach for efficient reduction of adaptation spaces for self-adaptive systems with threshold, optimization, and setpoint goals. We translate the research question to six requirements for ML2ASR+ that serve as drivers for devising the solution and evaluating it.

The first four requirements – reusability, automatic operation at runtime, modularity adaptation goals, and granularity of adaptation space reduction – are of a qualitative nature. The last two requirements – negligible utility penalty and efficiency – are of a quantitative nature.

Reusability. As a first requirement, the solution should be reusable, i.e., the solution should offer distinct functionalities and modules that can be instantiated and applied across application domains. We evaluate this requirement by demonstrating that the proposed solution can be applied to applications in two different domains.

Automatic Operation at Runtime. As a second desirable requirement, we want the solution to operate at runtime without human involvement. We evaluate this requirement by demonstrating that the proposed solution fully automatically reduces adaptation spaces at runtime for different application domains.

Modularity Adaptation Goals. As a third requirement, we want the solution to be able to handle different types of adaptation goals. The approach should be able to handle independent types of adaptation goals, as well as a combination of different types of goals in one system. We evaluate this requirement by demonstrating that the proposed solution can be applied to instances of the same applications with different types and combinations of adaptation goals.

Granularity of Adaptation Space Reduction. As a fourth requirement, we want our solution to have the option to specify the granularity of adaptation space reduction, i.e., the degree to which the solution reduces the adaptation space. Granularity applies to optimization and/or setpoint goals, enabling to determine which adaptation options to include based on well-defined criteria. E.g., for a setting with a setpoint goal, we may require the solution to find all the adaptation options within a given window around the setpoint value. Differentiating the granularity offers flexibility when the available adaptation time may be different under different conditions. We evaluate this requirement by demonstrating that the proposed solution can be applied for different levels of granularity of adaptation space reduction.

Negligible Utility Penalty. As a fifth requirement, we desire that the solution reduces the adaptation space with little or no penalty on the quality properties that are the subject of adaptation compared to an ideal solution where no adaptation space reduction is applied. Utility denotes here the effect on the quality properties due to the adaptation decisions made by using learning. We evaluate this requirement by comparing the differences in mean values of the relevant quality properties over time with and without learning. Depending on the type of goal (elaborated in Section 3.3) we either compare the satisfaction of the goal or compare the difference in the quality tied to that specific goal. We provide a concrete metric for the evaluation of utility penalty in Section 3.5.

Efficiency. As a sixth and final requirement, the solution should be efficient, i.e., the adaptation space should be reduced such that the analysis can be performed within the time window available to make adaptation decisions. We evaluate this requirement by demonstrating that the proposed solution effectively reduces the adaptation space in two different domains. We use three metrics to judge the efficiency of the adaptation space reduction: (1) the Average Adaptation Space Reduction (AASR in short) that compares the average number of adaptation options selected by learning over multiple adaptation cycles with the average of the total number of adaptation options over these adaptation cycles; (2) the total percentage of time saved as a result of the space reduction; and (3) the percentage of overhead in time of ML2ASR+ due to learning compared to the verification time required to verify the reduced adaptation space. We provide concrete metrics for the evaluation of efficiency in Section 3.5.

3.3 Model of Self-Adaptive System with Adaptation Goals and Running Example

We briefly outline the model for self-adaptation that we use in this research. Then, we give a simple example of a self-adaptive system that we use as a running case in the chapter. Finally, we explain different types of adaptation goals.

3.3.1 Model of Self-Adaptive System

Figure 3.1 shows a high-level model of a self-adaptive system as we follow in this chapter, leveraging on [361].

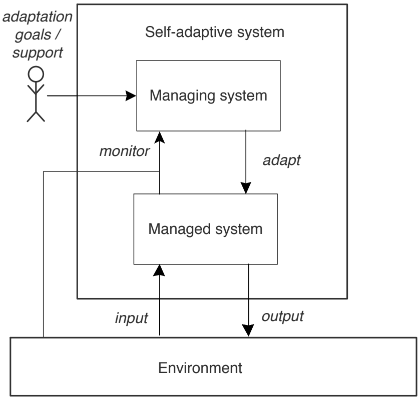


Figure 3.1: Model of a self-adaptive system

A self-adaptive system consists of two parts: a *managed system* and a *managing system*. The managed system can be any regular software-intensive system or a part of it. Hence, the managed system may refer to an entire system, a subsystem, one or more components, just a particular feature of a larger system, infrastructure, or resources used by a system, etc. Other terminology used to refer to self-adaptive systems are auto-tuned systems, elastic systems, controlled systems, context-controlled systems, and autonomic systems, among others.

The managed system takes *input* from an *environment* and produces *output* to the environment. While the managed system can be controlled, the elements in the environment cannot. The environment may include other software systems, hardware, communication networks, users, the operating context, and so forth. The managing system acts upon the managed system with a particular purpose, for instance, to improve its performance when operating conditions change or to deal with errors that may suddenly appear. The purpose is provided by stakeholders in the form of *adaptation goals*. The managing system *monitors* the managed system and/or its environment during operation, resolves uncertainties, and based on the adaptation goals *adapts* the managed system or parts of it when needed. A common approach to realize the managing system is by means of combining four basic functions: Monitor-Analyze-Plan-Execute that share a common Knowledge, which is often referred to as MAPE-K or MAPE in short [193]. The types of adaptations of the managed system may range from adjusting parameter settings to architectural re-configurations. Hence, the managed system needs to provide the necessary support to be monitorable and adaptable. Operators or other stakeholders may *support* the managing system in its tasks, but this is optional.

3.3.2 Running Example

We introduce a small example of a self-adaptive system that we use as a running case to illustrate ML2ASR+. The managed system in this example is a simple service-based system that handles service requests of clients through the invocation of a series of services. These services are deployed on two machines named **M1** and **M2**. The system has to deal with two uncertainties: fluctuations in network bandwidth and the workload of both machines respectively. These uncertainties affect three qualities that form the adaptation goals: the *failure rate*, *response time*, and the *cost* of service requests. To make sure that the qualities comply with the service level agreements of users, the system is equipped with a managing system. This managing system realizes a feedback loop that monitors the service system and has the ability to adapt the distribution of service requests between **M1** and **M2**.

3.3.3 Adaptation Goals

One of the requirements and a distinct feature of ML2ASR+ is support for different types of adaptation goals. We start with describing the types of adaptation goals ML2ASR+ supports one by one, and then we explain how multiple types of goals can be combined. We illustrate the goals with the running example.

Threshold Goals

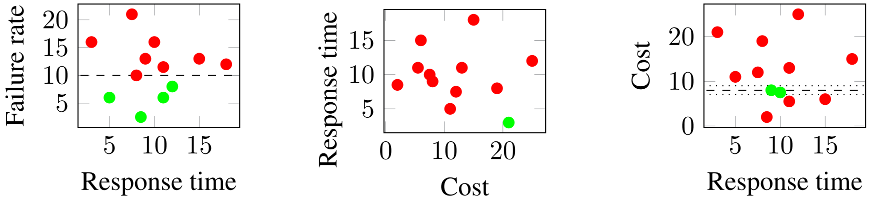
The first type of adaptation goal that we cover in this work is a threshold goal. A threshold goal imposes a restriction on one of the system's quality properties in the form of a threshold value that should not be exceeded. Exceeded in this context can refer to either an upper bound value that the quality property should not cross, or a lower bound value that acts as a minimum requirement for the quality property. We define the satisfaction of a threshold goal $\mathcal{T} \in \mathbb{T}$ with a threshold value $\bar{x}$ for any value of the quality property q (or quality value in short) as follows:

$$\mathcal{T}_{<\bar{x}}(q) = \begin{cases} True \ : q < \bar{x} \\ False : q \geqslant \bar{x} \end{cases} \tag{3.1}$$

$$\mathcal{T}_{>\bar{x}}(q) = \begin{cases} True \ : q > \bar{x} \\ False : q \leqslant \bar{x} \end{cases} \tag{3.2}$$

A threshold goal allows a self-adaptive system to categorize adaptation options in two distinct classes: compliant with the threshold goal or in violation of the threshold goal. Hence, threshold goals form a perfect candidate for the classification of adaptation options, a classic supervised machine learning technique.

Example: Applied to the running example, we can define a threshold goal for the system to keep the failure rate below a given time percentage, say 10%, as shown in Figure 3.2a. In this case, the set of quality values q that satisfy the threshold goal, i.e., $\mathcal{T}_{<10\%}(q) = True$, correspond to classification class 1, while the set of quality values q that do not satisfy the threshold goal, i.e., $\mathcal{T}_{<10\%}(q) = False$, correspond to classification class 0.



(a) Example threshold goal of 10 for the failure rate.

(b) Example optimization goal (minimize) for response time.

(c) Example setpoint goal of 8 with error margin 1 for the cost.

Figure 3.2: Example scenarios for each type of adaptation goal.

Optimization Goals

The second type of adaptation goal that we cover is an optimization goal. As the name suggests, an optimization goal aims to optimize a quality property of the system, which can either maximize or minimize the value of the quality property. We define the satisfaction of an optimization goal $\mathcal{O} \in \mathbb{O}$ for any quality value q as follows:

$$\mathcal{O}_{min}(q) = \begin{cases} True & : q = min(\{q_1, q_2, ..., q_n\}) \\ False & : \text{otherwise} \end{cases} \tag{3.3}$$

$$\mathcal{O}_{max}(q) = \begin{cases} True & : q = max(\{q_1, q_2, ..., q_n\}) \\ False & : \text{otherwise} \end{cases} \tag{3.4}$$

with $\{q_1, q_2, ..., q_n\}$ the set of quality values of all the adaptation options in the adaptation space.

The natural approach to predict the values of the quality property and judge the adaptation options accordingly is regression. After the prediction, different strategies can be applied to perform the analysis. One strategy is selecting and analyzing a subset of adaptation options that were predicted to have quality values close to optimal. This way a small margin of error for the applied regression technique is taken into account. Another strategy is to restrict the analysis to only the adaptation option with the optimally predicted value of the quality property. This strategy can be applied if the time for computing the adaptation option is critical; yet, it may miss the best adaptation option since the predictions with regression are subject to errors. The strategy chosen represents the requirement of granularity of adaptation space reduction, see Section 3.2.4.

Example: For the running example, we can define an optimization goal that minimizes the response time of service requests to the system, i.e., $\mathcal{O}_{min}(q)$. Here we reduce the adaptation space by looking at the top 10 adaptation options in terms of predicted response time.[2] Alternatively, we could opt to reduce the adaptation space to just a single option, when choosing a more strict granularity. Figure 3.2b shows the optimization goal when we choose to reduce the adaptation space to just one option.

Setpoint Goals

The third and final type of adaptation goal covered in this chapter is a setpoint goal. A setpoint goal aims to keep the quality property of interest at (or close to) a given target value (i.e., the setpoint value or just the setpoint). We define the satisfaction of a setpoint goal $\mathcal{S} \in \mathbb{S}$ with target μ and error margin ϵ for any quality value q as follows:

$$\mathcal{S}_{\mu,\epsilon}(q) = \begin{cases} True & : |q - \mu| < \epsilon \\ False & : \text{otherwise} \end{cases} \tag{3.5}$$

[2] In this chapter we define granularity in terms of an absolute number. An alternative approach could be to define granularity as a percentage of the total number of adaptation options. Using a number has the advantage that the time and resources required for analysis can be estimated; the advantage of using percentages is that the relative number of options considered is fixed. So there is a tradeoff between the two options. The proposed solution can be easily adjusted for both options.

For this type of goal, both classification and regression are candidates to predict quality values. Regression allows the identification of adaptation options with predicted quality values close to the setpoint value. Classification on the other hand enables the classification of adaptation options as either (1) being inside the specified epsilon window around the setpoint value or (2) outside the window.

Example: For the running example, we can specify a setpoint goal to keep the average cost of service invocations in the system at 8 cents with an error margin of 1 cent, i.e., $\mathcal{S}_{8c,1c}(q)$, as shown in Figure 3.2c. Depending on the granularity set for adaptation space reduction, the adaptation space is reduced to adaptation options within a limited window around the setpoint value.

Combination of Multiple Goals

In practice, self-adaptive systems usually have to deal with multiple adaptation goals. ML2ASR+ supports adaptation space reduction for an arbitrary set of adaptation goals. However, in this chapter, we restrict ourselves to combinations of multiple threshold goals $\mathbb{T}$, multiple setpoint goals $\mathbb{S}$, and a single optimization goal $\mathcal{O}$, representing a large class of practical systems, as illustrated with the running example and the cases used for the evaluation of ML2ASR+ in Section 3.6. The combined set of goals, denoted as $\mathbb{G}$, is defined as:

$$\mathbb{G} = \, < \mathbb{T}, \mathbb{S}, \{\mathcal{O}\} > \tag{3.6}$$

Hence, self-adaptive systems that rely on multi-objective optimization of adaptation goals to make adaptation decisions are not in the scope of the work presented in this chapter. The following sections explain in detail how ML2ASR+ reduces adaptation spaces when a combination of goals $\mathbb{G}$ needs to be satisfied.

Example: For the running example, we can combine different types of goals as specified above, for instance keeping the failure rate below a given threshold while minimizing the response time of service requests to the system.

3.4 Machine Learning To Adaptation Space Reduction

We now present ML2ASR+, addressing the research question we presented in Section 3.2.4. ML2ASR+ is a modular approach for adaptation space reduction in self-adaptive systems, meaning it can be instantiated in multiple ways, depending on the needs of the domain at hand. We focus specifically on the use of two classic supervised machine learning methods: classification and regression, applied to systems with different types of adaptation goals.

We start with presenting the runtime architecture of ML2ASR+ that integrates a machine learning module in the architecture of a self-adaptive system. Then, we give a high-level overview of the workflow of ML2ASR+. Finally, we zoom in on the design time and runtime stages of the workflow.

3.4.1 Runtime Architecture of ML2ASR+

Figure 3.3 shows the high-level runtime architecture of a MAPE-based self-adaptive system extended with a *Machine Learning Module* that realizes adaptation space reduction. The *Monitor* tracks the uncertainties and properties of the underlying managed system (1) and updates the information in the *Knowledge* repository. The *Analyzer* then evaluates the need for adaptation, based on the current conditions (2). When this is the case, the analyzer composes a set of possible adaptation options, i.e., the configurations that can be reached from the current configuration by applying adaptation. This set is then passed to the *Machine Learning Module* (3) that makes predictions of the adaptation options using the machine learning models. Based on these predictions and the adaptation goals, the *Machine Learning Module* filters the options, reducing the set of adaptation options. These adaptation options are verified by the *Verifier Module* using a set of runtime models of the quality properties that correspond with the adaptation goals (4). The resulting estimates of the quality properties per adaptation option are then used by the *Machine Learning Module* to further train its internal learning models (5), resembling the online learning part of ML2ASR+. The *Planner* then evaluates the verified adaptation options, determines the best adaptation option available based on the adaptation goals, and composes a plan to adapt the managed system (6). Finally, the *Planner* triggers the *Executor* (7) that executes the steps of the plan adapting the managed system (8).

In the remainder of this section, we explain how the Machine Learning Module is designed for a problem at hand (design stage of the ML2ASR+ workflow) and how the module reduces adaptation spaces at runtime (runtime stage).

3.4.2 High-level Overview of the ML2ASR+ Workflow

We start with a high-level overview of the workflow of ML2ASR+, shown in Figure 3.4. We explain the two stages of the workflow in general here and discuss them in detail in the next sections.

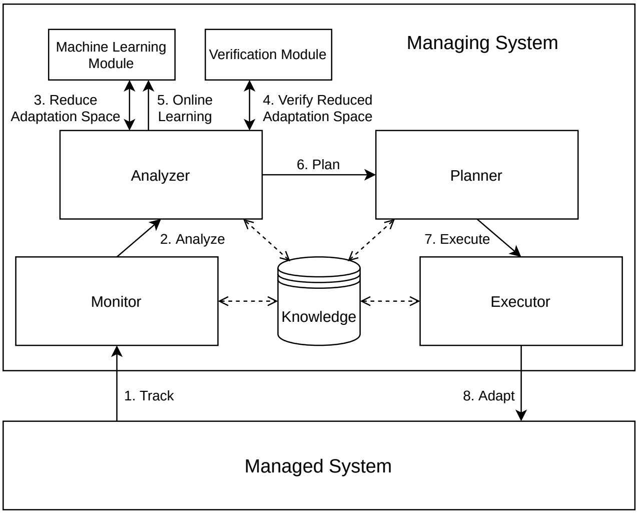


Figure 3.3: General MAPE-K architecture extended with a Machine Learning Module that reduces adaptation spaces.

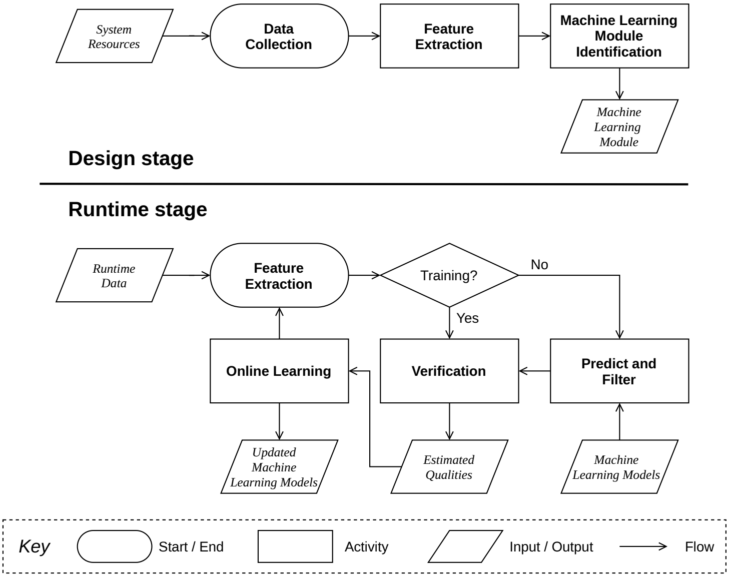


Figure 3.4: High level overview of the workflow of ML2ASR+.

The design stage starts with the collection of data from the managed system and its environment. This data captures information relevant to the adaptation of the system over a period of time. This includes properties in the environment that affect the behavior of the system (e.g., actual workloads of the machines in the running example), system configurations (e.g., the distribution of service requests between the machines), and quality properties (e.g., the response time of service requests). Besides the system in operation, other suitable resources can be used to collect the data, such as a simulator or files with historical data. Next, features are extracted from the data. Features are measurable properties of the system and its environment that are relevant for self-adaptation. Uncertainties in the running example are the fluctuating workload of the machines and the bandwidth of the network. The extracted features are then used for the identification of the Machine Learning Module. To that end, different configurations of the Machine Learning Module (based on different types of learning models and other attributes such as scalers that are used to normalize the collected data) are compared and the best configuration is selected. The output of the design stage is a Machine Learning Module that comprises machine learning models with a set of attributes (for instance scalers) and a predictor with a filter that allows predicting the qualities of adaptation options that can then be filtered to reduce the adaptation space. The Machine Learning Module is then ready for deployment and use at runtime.

The runtime stage works in cycles, each representing an opportunity for the system to perform adaptation. The workflow starts with gathering runtime data from the managed system and its environment that is relevant for adaptation. An example for the running example is the actual value of the workload of the two machines used in the service-based system. From this data, features are extracted, similarly to the design stage, yet now based on the data collected at runtime. Then two sub-stages are distinguished: training and testing. Immediately after deployment of the Machine Learning Module, the machine learning models need to be trained to make accurate predictions about quality properties in the system, filter the adaptation options, and reduce the adaptation space. The adaptation options in the running example are determined by the different settings that are available for distributing service requests between the two machines. In the training sub-stage, the system does not make any predictions yet. Consequently, as many adaptation options as possible are analyzed (i.e., the qualities are estimated using a verifier). Different heuristics can be used to select adaptation options from the total set, for instance, options may be selected randomly, or the options may be divided into batches that are analyzed in subsequent slots. The number of cycles that are used for the training sub-stage is a parameter that is determined during the design stage.

The second sub-stage of the runtime workflow is called testing. During testing, the trained machine learning models are effectively used to reduce the adaptation space. In addition, the new verification results for adaptation options of the reduced adaptation space are used to continue the learning of the machine learning models. In the testing sub-stage, the Machine Learning Module predicts the quality properties of

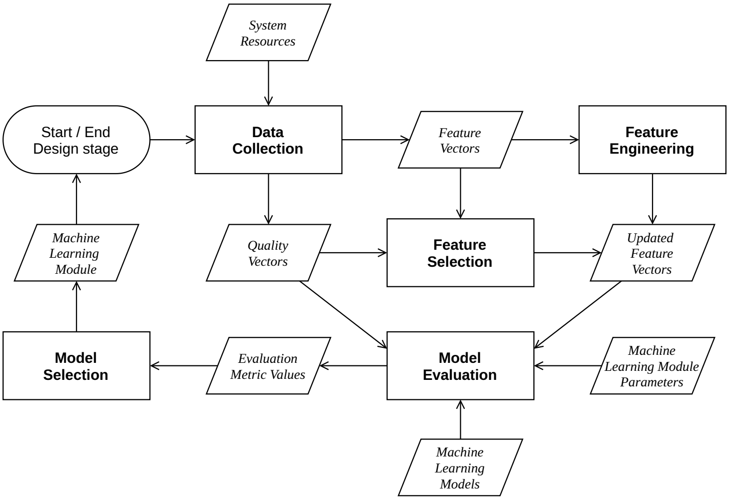


Figure 3.5: Workflow of the design stage activities for ML2ASR+.

the adaptation options, and based on these results and the adaptation goals set for the system, a subset of adaptation options is selected for verification. The verification results, i.e., estimates of the quality properties of the adaptation options of the reduced adaptation space, are used for online learning. The updated machine learning models are then ready to perform adaptation space reduction for the next cycle, and the verification results can be used by the planner to make an adaptation decision.

In the following sections, we elaborate on the different steps of the two stages of the workflow. To precisely describe the different activities, we use a lightweight formalization.

3.4.3 Design Stage of the ML2ASR+ Workflow in Detail

Figure 3.5 describes the workflow of the design stage activities in detail. The design stage comprises five distinct activities: *Data Collection*, *Feature Selection*, *Feature Engineering*, *Model Evaluation*, and *Model Selection*. The output of the design stage is a configuration for the Machine Learning Module that can then be deployed and used to support a self-adaptive system by reducing large adaptation spaces at runtime.

Before explaining the activities in detail, we highlight the software artifacts used for each activity and the responsibilities of the engineer; the various software artifacts are at the disposal of the engineer to perform the different activities. Table 3.1 gives an overview of the artifacts used for the activities with the responsibilities of the engineer.

Data collection is initiated by an engineer who selects a system resource and configures an artifact that is then used to collect data. Feature selection uses the collected data as input in a feature importance function that is used to filter out unimportant features. Feature engineering is automatically initiated after feature selection which adjusts individual feature values according to the feature scaling algorithm. Model evaluation automatically initiates after feature engineering by taking the updated features and collected system qualities to run and evaluate different machine learning algorithms. Lastly, model selection is performed by an engineer who inspects the evaluation metrics from model evaluation to make a final decision about the configuration of the Machine Learning Module.

Table 3.1: Responsibilities engineer with supporting software artifacts for the design stage activities.

Activity	**Artifacts**	**Responsibilities engineer**
Data collection	System resource with data collection artifact	Determine system resource and configure data collection artifact
Feature selection	Feature importance function	Select feature importance function
Feature engineering	Feature scaling algorithm implementations	Choose feature scaling algorithm to use
Model evaluation	Machine learning algorithm implementations	Determine machine learning algorithms to evaluate and evaluation metrics to measure
Model selection		Make final decision about configuration by inspecting evaluation metrics

Data Collection

During *Data collection*, data concerning adaptation is gathered from the managed system and the environment in which the system operates. We categorize the data into two categories: potential features and system qualities. A potential feature f is any type of property of the system or the environment that could have an influence on at least one quality property of the system. A system quality q represents a non-functional property of the system. Data is collected for a period of time. At each time instance, the potential features and the associated qualities are collected.

We introduce the following definitions[3]:

$\Pi_i = \{\pi_1, ..., \pi_n\}$: A set of adaptation options in the system.

$\Pi = \{\Pi_1, ..., \Pi_n\}$: The set of all sets of adaptation options.

$U_i = \{u_1, ..., u_n\}$: A set of uncertainties that can be monitored.

$U = \{U_1, ..., U_n\}$: The set of all possible sets of uncertainties that can be monitored.

$\lambda_i = \{f_1, ..., f_{n+m}\}$: A set of features comprising n features that represents a system configuration and m features that represent uncertainties.[4] We call λ_i a feature vector.

$\Lambda_i = \{\lambda_1, ..., \lambda_n\}$: The set of possible feature vectors.

$\Lambda = \{\Lambda_1, ..., \Lambda_n\}$: The set of possible sets of feature vectors.

$\phi_i = \{q_1, ..., q_n\}$: A set of qualities of a system. We call ϕ_i a quality vector.

$\Phi_i = \{\phi_1, ..., \phi_n\}$: The set of possible quality vectors.

$\Phi = \{\Phi_1, ..., \Phi_n\}$: The set of possible sets of quality vectors.

$s \in S$: A system resource, with S the set of all resources of managed systems.

The standard resource used for data collection is the system deployed in its real-world setting. This resource ensures that the most accurate data is collected to design and configure the Machine Learning Module. However, collecting real-world data may be hard, for instance for large-scale distributed systems, or it may be an expensive and time-consuming process. Alternative approaches can then be applied, such as simulating the system or using historical data collected from the system. Such techniques may be more convenient to generate large amounts of data covering a wide range of different system states. We formally define the $CollectData$ function as follows:

$$CollectData : S \rightarrow \Lambda \times \Phi$$
$$CollectData(s) =< \{\lambda_1, ..., \lambda_n\}, \{\phi_1, ..., \phi_n\} >$$

Data collection from resource s results in a list of feature vectors $\{\lambda_1, ..., \lambda_n\}$ and quality vectors $\{\phi_1, ..., \phi_n\}$. The potential features of λ_i correspond with quality values ϕ_i. The feature vectors and quality vectors provide the input to the next design stage

[3] We use the variable n in multiple definitions to denote the number of elements in a set. However, each of the scope of these numbers is a single definition, and values of n can be different in different definitions.

[4] We use the term *potential feature* to refer to a feature that may have an effect on any quality property of the system. A potential feature becomes a *feature* if it has an actual effect on any quality property of the system, which is determined during feature selection. We do not distinguish potential features and features in the formal definition of feature sets.

Table 3.2: Excerpt of data collected for the service-based application (abbreviations: M → Machine, WL → Workload, ABW → Available Bandwidth).

Distribution	WL M1	WL M2	ABW M1	ABW M2	Response time
40	75	20	10	10	16ms
50	40	15	50	50	8ms
20	60	80	25	30	14ms
80	5	25	50	75	3ms
50	25	20	70	60	5ms
60	70	40	40	60	11ms

activities.

Example: Table 3.2 shows an excerpt of data collected for the running example application. Each row defines a feature vector with values for {*Distribution, Workload M1, ..., ABW M2*} and a quality vector with values for {*Response time*}. Note that only a subset of potential features and qualities are listed in the table for the sake of clarity. We also consider a small set of feature vectors to keep the example simple. The full data set includes all the features that may have an impact on the qualities of the system, as well as all the associated quality values of the system.

Feature Selection

During the next two activities, relevant features are extracted from the collected data, defined as follows:

$$ExtractFeatures = EngineerFeatures \circ SelectFeatures$$

During *Feature selection*, the potential features and their respective quality values are evaluated using a feature selection algorithm. This algorithm analyzes the impact of individual features on the quality values associated with them. Irrelevant features, i.e., features that do not (or only marginally) influence the qualities, can be filtered out. This will simplify the machine learning model and enhance the performance of the Machine Learning Module.

It is important to note that *Feature selection* carries an inherent risk. In fact, the algorithm determines the relevance of each feature based on its influence on the quality values, yet this evaluation is based on the data collected during *Data collection*. If this data does not cover the scenarios where the feature has an influence on the qualities of the system, this feature will not be selected. For this reason, we leave *Feature selection* as an optional activity in the design stage. It is the task of the engineer to carefully evaluate the data collected from the system to determine whether or not feature selection should be included. Feature selection is formally defined as follows:

$IND = 2^{\{1,\ldots,n\}}$: The set of all possible subsets of indices in the range $[1 \ldots n]$.

$ind \in IND$: The set of indices of features that are deemed relevant.

$$SelectFeatures : \Lambda_i \times IND \rightarrow \Lambda_j$$

$$SelectFeatures(\{\lambda_1, ..., \lambda_n\}, ind) = \\ \{\, \lambda_j^{sel} \subseteq \lambda_i \mid \forall f_n \in \lambda_j^{sel} : Relevant(f_n, ind) = True\}$$

Feature selection uses the *Relevant* function which uses the set of indices (denoted as ind) to decide whether individual features of a feature vector should be included or filtered out. Hence, the features in the resulting feature vector are the subset of the features in the original feature vector that are relevant.

Example: The results of applying feature selection on the excerpt of the data collected from our example system (shown in Table 3.2) are shown in Table 3.3. In this case, feature selection determined that feature *ABW 2* has no influence on the response time and consequently, this feature is excluded.

Table 3.3: Example of performing *Feature selection* on the data from Table 3.2.

Distribution	**WL M1**	**WL M2**	**ABW M1**	~~**ABW M2**~~	**Response time**
40	75	20	10	~~10~~	16ms
50	40	15	50	~~50~~	8ms
20	60	80	25	~~30~~	14ms
80	5	25	50	~~75~~	3ms
50	25	20	70	~~60~~	5ms
60	70	40	40	~~60~~	11ms

Feature Engineering

During *Feature engineering*, the concrete values of the features are inspected and adjusted if this benefits the quality of predictions. As such feature engineering ties in closely with the next activity: *Model selection*. A well-known example of feature engineering is scaling which is used for features with values of varying magnitude, range, and units. One scaling technique is normalization where values of features are shifted and re-scaled to fit in a range between 0 and 1 (known as Min-Max scaling). Another scaling technique is standardization where values of features are centered around the mean with a unit standard deviation. Formally, feature engineering is defined as follows:

$$
\begin{aligned}
&Transform : \lambda_i \rightarrow \lambda_j \\
&EngineerFeatures : \Lambda_i \rightarrow \Lambda_j \\
&EngineerFeatures(\{f_1, ..., f_n\}) = \\
&\qquad \{\, f_j \mid f_j = Transform(f_i),\ f_i \in \{f_1, ..., f_n\}\,\}
\end{aligned}
$$

Feature engineering is centered around $Transform$ that transforms the values of the features according to a concrete engineering method that is used (e.g. scaling with normalization or with standardization). The result of feature engineering is a set of normalized features.

Example: Table 3.4 shows an example of feature engineering applied to the selected features of our running example shown in Table 3.3. In this particular case, the values of the distribution, workload, and available bandwidth are normalized, i.e., the values are rescaled to a range between 0 and 1 instead of the original values between 0 and 100.

Table 3.4: Example of performing *Feature engineering* on the data from Table 3.3. The engineered feature values are marked in blue.

Distribution	**WL M1**	**WL M2**	**ABW M1**	**Response time**
0.4	0.75	0.2	0.1	16ms
0.5	0.4	0.15	0.5	8ms
0.2	0.6	0.8	0.25	14ms
0.8	0.05	0.25	0.5	3ms
0.5	0.25	0.2	0.7	5ms
0.6	0.7	0.4	0.4	11ms

Evaluation of Models

During the last two activities of the design stage, we identify the machine learning models of the Machine Learning Module, which is defined as follows:

$$IdentifyModels = SelectModels \circ EvaluateModels$$

We start with *Evaluation of models* that uses the features extracted from the data of the system to determine a set of metrics that can be used to evaluate the performance of different machine learning models (in the next activity). Such metrics are determined to evaluate learning models per adaptation goal. To that end, a list of potential machine learning models is composed that combine different learning algorithms with variations on their internal loss and penalty functions[5]. It is important to note that the selected algorithms need to support online learning, i.e., have the ability to continue training and thus update machine learning models after deployment.

Besides the list of machine learning models, two internal parameters of the Machine Learning Module are evaluated during model evaluation. The first parameter, *exploration rate*, represents the percentage of extra adaptation options that are selected for analysis (by the self-adaptive system) on top of the adaptation options that are predicted by the Machine Learning Module as being compliant with the adaptation goals. Exploring an additional percentage of adaptation options ensures that the Machine Learning Module also relearns a sample of options that may otherwise be ignored. The second parameter that we evaluate is called *warm-up count*. This parameter gives an indication for the number of training cycles that the Machine Learning Module should consider before it can be used to make meaningful predictions during operation (i.e., switch from training to testing).

We introduce the following definitions:

$M = \{m_1, ..., m_n\}$: The set of machine learning models to be evaluated.

$\mathbb{M}$: The set of all sets of machine learning models.

$E = \{e\}$: The *exploration rate* internal to the Machine Learning Module.

$W = \{w\}$: The *warm-up count* internal to the Machine Learning Module.

θ: A set of metrics for the evaluation of machine learning models.

Θ: The complete set of possible evaluation metrics.

$\mathbb{E}$: The set of all sets of evaluation metrics for machine learning models.

For the evaluation of the machine learning model, we use a train-test split. Train-test split is an efficient procedure to estimate the performance of classification or regression models. The method can be used if a sufficient large labeled dataset is available [18, 128], which applies to our case where such dataset can be obtained from the system or a simulation as explained above.[6] The evaluation of a machine learning model involves two steps: (1) training the model with a set of feature- and quality vectors and (2) testing the efficacy of the model by making predictions over a different set of feature vectors, examining the predictions through analyzing the according machine learning metrics. This process can range from splitting up the complete data set into two partitions (a train- and test dataset) to dividing the complete data set into multiple pairs of train- and test datasets (cross-validation). For the interested reader, we refer

[5] Whereas loss corresponds to the inaccuracy of predictions as explained above; a penalty expresses the degree of impact that the loss will have on the model of the learner.

[6] The dataset of DeltaIoT, the first application used in the evaluation, comprises in total 64800 data points; the dataset of SBS, the second application, comprises 1350000 data points.

to A.2.1 where we present a formal foundation for the former. *Model evaluation* is then defined as follows:

$$\begin{aligned}
&EvaluateModels : \mathbb{M} \times \Lambda \times \Phi \times E \times W \rightarrow \mathbb{M} \times \mathbb{E} \\
&EvaluateModels(M_i, ExtractFeatures(\Lambda_i, \{ind\}), \Phi_i, e, w) = \\
&\qquad < \{m_1, ..., m_n\}, \{\theta_1, ..., \theta_n\} >
\end{aligned}$$

Model evaluation results in a set of metrics sets, one set per machine learning model. Recall that metrics are determined per adaptation goal. Hence, we repeat model evaluation per goal, resulting in a set of evaluation metrics sets for each adaptation goal. After model evaluation, the metrics are used in the final design stage activity to select the learning models of the Machine Learning Module that will be used for adaptation space reduction at runtime.

For model evaluation of threshold goals, we apply classification using two evaluation metrics: F1-score, and Matthews correlation coefficient. For model evaluation of setpoint and optimization goals, we apply regression using four evaluation metrics: the R2-score, mean squared error, median absolute error, and maximum error. We elaborate on all aforementioned metrics further in Section 3.5.

Selection of Models

In the last activity of the design stage, we select a learning model from the evaluated learning models relying on the metrics derived from the evaluation of these models. *Selection of models* is formally defined as follows:

$$\begin{aligned}
&SelectModels : \mathbb{M} \times \mathbb{E} \rightarrow M \\
&SelectModels(\{m_1, ..., m_n\}, \{\theta_1, ..., \theta_n\}) = m^{selected}
\end{aligned}$$

During model selection, the designer evaluates the metrics for the different machine learning models to make an informed decision about which model to use at runtime. This is repeated for each adaptation goal. Once the learning models are selected the Machine Learning Module can be configured and deployed to be used at runtime (we explain the elements of a Machine Learning Module configuration below).

Example: Table 3.5 illustrates model evaluation and model selection for our running case. The data in the table builds on the previous examples and considers three machine learning models for classification, denoted with Model 1, 2, and 3. To keep it simple, we restrict the evaluation to a single threshold goal: the response time of service requests should not exceed 10ms. We also consider the accuracy of the model as a single evaluation metric[7]. The table at the top shows the predictions of the different learning models. E.g., the first line for the features with a response time of 16ms exceeds the threshold goal (of 10ms) and should be classified as 0. This is correctly done by Model 1 and Model 2, but not by Model 3. The table at the bottom shows the accuracy of each Model. Based on these results, selecting a model is straightforward: the engineer selects model 3 in this example which has the highest accuracy.[8] In case, multiple metrics are used, the engineer needs to make an informed decision taking into account the different results.

Table 3.5: Excerpt of *Model selection* and *Model evaluation* in the service-based application. Values in columns Model 1, 2, and 3 represent the classes of predictions of the models. Class 0 means that the response time goal is violated, class 1 means that the goal is achieved. The values of the classes that are predicted correctly are marked in green; the values of the classes that are predicted incorrectly are marked in red and are underlined.

Response time	**Response time < 10ms?**	**Model 1**	**Model 2**	**Model 3**
16ms	0	0	0	1
8ms	1	1	1	1
14ms	0	0	1	0
3ms	1	1	1	1
5ms	1	0	0	1
11ms	0	1	0	0

	Model 1	**Model 2**	**Model 3**
Accuracy	50%	66.6%	83.3%

Setup and Configuration of the Machine Learning Module

After completing the design stage activities, the Machine Learning Module can be set up and configured. Figure 3.6 shows the architecture of the Machine Learning Module that comprises four main components.

The *Feature constructor* is responsible for assembling and extracting feature vectors. It takes as input a set of adaptation options and the values of uncertainties. The feature constructor combines this input using *Feature composition* (we explain this runtime activity below) and *Feature engineering*. The output is a set of feature vectors

[7] Evidently, in practice, multiple metrics will be used, but we simplify this here for illustration purposes.

[8] Accuracy is computed as the fraction of correct predictions expressed as a percentage.

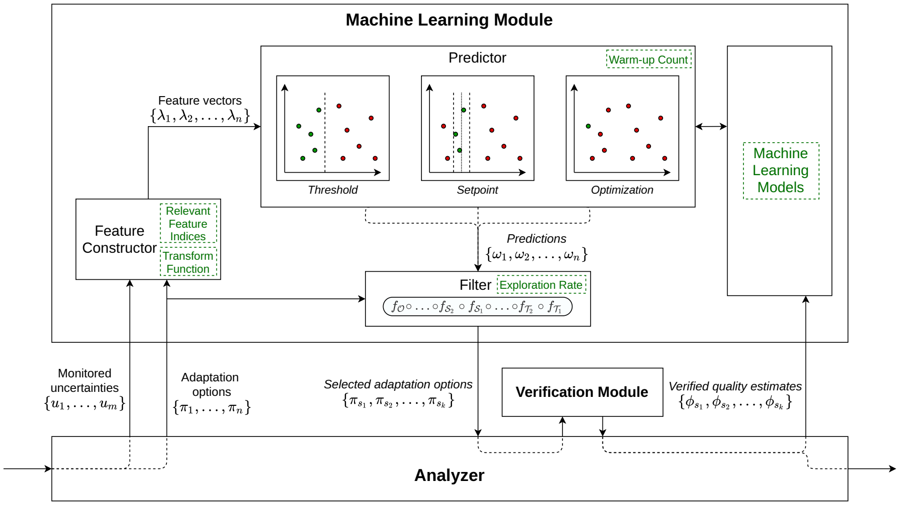

Machine Learning Module
Predictor
Warm-up Count
Feature vectors
$\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$
Threshold
Setpoint
Optimization
Machine Learning Models
Feature Constructor
Relevant Feature Indices
Transform Function
Predictions
$\{\omega_1, \omega_2, \ldots, \omega_n\}$
Filter
Exploration Rate
$f_{\mathcal{O}} \circ \ldots \circ f_{\mathcal{S}_2} \circ f_{\mathcal{S}_1} \circ \ldots \circ f_{\mathcal{T}_2} \circ f_{\mathcal{T}_1}$
Monitored uncertainties
$\{u_1, \ldots, u_m\}$
Adaptation options
$\{\pi_1, \ldots, \pi_n\}$
Selected adaptation options
$\{\pi_{s_1}, \pi_{s_2}, \ldots, \pi_{s_k}\}$
Verification Module
Verified quality estimates
$\{\phi_{s_1}, \phi_{s_2}, \ldots, \phi_{s_k}\}$
Analyzer

Figure 3.6: Architecture of the Machine Learning Module; configuration elements are marked in green dotted boxes.

obtained from the runtime data. The feature constructor is configured using two specific parameters: the indices of relevant features $\{ind\}$ (determined in *Feature selection*) and a *Transform* function (determined in *Feature engineering*).

The *Machine Learning Models* that are determined during the design stage are maintained in a data repository. Conceptually, the learning models are part of the Machine Learning Module. However, in practice, the models may be stored in the *Knowledge* repository of the MAPE-K feedback loop.

The *Predictor* is responsible for making predictions of the adaptation options (i.e., the feature vectors produced by the feature constructor). In particular, the predictor makes predictions about the satisfaction of adaptation goals of the adaptation options (as specified by the $Predict$ function), leveraging on the machine learning models. The output of the predictor is a set of predictions for the different adaptation options that need further filtering. The predictor is configured using the internal parameter *warm-up count* that determines the period that is used for training the machine learning models. We explain the predictor below in the section about testing.

Finally, the *Filter* is responsible for filtering the adaptation options based on the predictions for the adaptation goals made by the predictor. Besides determining relevant adaptation options, the filter selects a subset of additional features to be explored based on the *exploration rate* parameter. The output of the filter is a reduced set of adaptation options that are used for verification. The verification results are then used for online

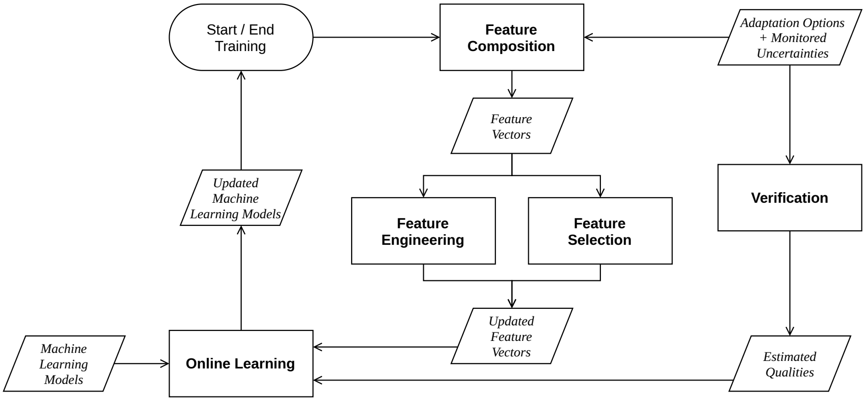


Figure 3.7: Workflow of the runtime stage activities in training cycles for ML2ASR+.

learning of the machine learning models. We explain filtering further in the section about testing below.

3.4.4 Runtime Stage of the ML2ASR+ Workflow in Detail: Training

The runtime stage consists of two sub-stages: *Training* followed by *Testing*. In contrast to the design stage activities, the runtime stage activities work fully automatic and require no human input.

We start with *Training*. Training takes place immediately after deployment when the Machine Learning Module has not yet learned nor gathered enough data of the system and its environment to make accurate predictions about the system's qualities. Figure 3.7 gives a detailed overview of the workflow of the runtime stage activities during training. Training is applied for a number of cycles, based on the *warm-up count* that was determined during the design stage.

During training, the Machine Learning Module is not reducing the adaptation space yet. Instead, the available adaptation time of the system is used to formally verify as many adaptation options as possible, and the verification results are used to train the learning models of the Machine Learning Module. If not all adaptation options can be verified within a single time window that is available to make an adaptation decision, different strategies can be applied to select and verify adaptation options. A simple strategy selects adaptation options randomly. Another more balanced approach applies a round-robin strategy to select adaptation options one by one in consecutive time windows. Yet another strategy applies active learning to choose adaptation options for verification such that the Machine Learning Module can learn more efficiently by e.g., selecting options with maximum entropy [309]. ML2ASR+ is flexible and does not prescribe any particular strategy.

We use the following basic definitions to formally describe the activities of the runtime stage:

$\Omega_i = \{\omega_1, ..., \omega_n\}$: A set of predictions made by a learning model.

$\Omega = \{\Omega_1, ..., \Omega_n\}$: The set of all sets of predictions made by a learning model.

$\mathbb{Z}$: The set of all sets of predictions.

It is important to note that ML2ASR+ currently focuses on handling discrete adaptation options. System designers can however discretize a continuous adaptation space to apply ML2ASR+.

In the first activity of training, feature vectors are composed, meaning, the set of possible adaptation options is combined with the set of uncertainties monitored by the system. Formally $ComposeFeatures$ is defined as follows:

$$ComposeFeatures : \Pi \times U \rightarrow \Lambda$$
$$ComposeFeatures(\Pi_i, U_i) = \Lambda_i = \{\lambda_{\pi_1,U_i}, \lambda_{\pi_2,U_i}, ...\}$$

Features composition generates a feature vector that combines the features representing a system configuration (adaptation option) with the features representing the monitored uncertainties. The composed features then undergo selection and engineering before they are used for online learning (see below), resulting in updated feature vectors.

Example: Table 3.6 illustrates the composition of features as well as feature extraction, i.e., feature selection and feature engineering. The monitored uncertainties include the workload and available bandwidth of the machines. The different settings of the distribution of service requests represent here the adaptation options, i.e., system configurations. Based on feature extraction, the feature $ABW\ M2$ is not included in the updated feature vectors.

To enable online learning, i.e., which is actually a continued training activity, adaptation options need to be verified, preferably as many as possible (as explained above). We define *Verify* as follows:

$QM = \{qm_1, ..., qm_n\}$: A set of formal quality models used to estimate system qualities.

Table 3.6: Example of composed features and application of feature extraction for the service-based application.

Distribution	WL M1	WL M2	ABW M1	~~ABW M2~~
0.1	0.4	0.2	0.5	~~0.3~~
0.2	0.4	0.2	0.5	~~0.3~~
0.3	0.4	0.2	0.5	~~0.3~~
0.4	0.4	0.2	0.5	~~0.3~~
0.5	0.4	0.2	0.5	~~0.3~~
...	...	...	...	...

Table 3.7: Example of formal verification of a set of adaptation options, marked in blue.

Distribution	WL M1	WL M2	ABW M1	ABW M2	Estimated RT
10	40	20	50	30	3ms
20	40	20	50	30	8ms
30	40	20	50	30	6ms
40	40	20	50	30	11ms
50	40	20	50	30	13ms
...	...	...	...	...	...

$\mathbb{Q}$: The set of all sets of formal quality models.

$$Verify : \Pi \times U \times \mathbb{Q} \rightarrow \Pi \times \Phi$$

$$Verify(\{\pi_1, ..., \pi_n\}, \{u_1, ..., u_m\}, \{qm_1, ..., qm_k\}) = \\ < \{\pi_1, ..., \pi_n\}, \{\phi_1, ..., \phi_n\} >$$

Verification generates a set of quality values (one per goal) for each adaptation option. It is important to note that the quality values for the different adaptation options are estimates. The accuracy of these estimates is determined by the precision of the quality models used, the measurements of the uncertainties, and the verification method applied.

Example: Table 3.7 illustrates the verification results of a quality model for the response time of a sample of adaptation options from our example service-based application.

Lastly, we use the updated feature vectors and the estimated quality vectors to train the machine learning models. More specifically, we employ online learning to continuously update and refine the machine learning models during the testing cycles. Online learning, also referred to as incremental learning, allows a learner to incrementally learn from newly provided data samples. We refer the interested reader to the following articles [56, 217]. $LearnOnline$ is defined as follows:

$$LearnOnline : \Lambda \times \Phi \times M \rightarrow M$$
$$LearnOnline(ExtractFeatures(\Lambda_i, ind), \Phi_i, m) = m^{updated}$$

Online learning is defined for a single learning model and hence needs to be repeated for all models. Initially, online learning starts from the model selected during the design stage ($m^{selected}$). Online learning then uses the features extracted from the composite feature vectors that are derived from the runtime data, the quality vectors associated with the adaptation options obtained from verification, and the model that is subject to training. The result is an updated learning model.

3.4.5 Runtime Stage of the ML2ASR+ Workflow in Detail: Testing

Once the machine learning models are trained (based on the *warm-up count* determined during the design stage), the Machine Learning Module switches to *Testing*. As opposed to training, during testing cycles, the Machine Learning Module uses the machine learning models to make effective predictions about the qualities of adaptation options. These predictions can then be used to reduce the adaptation space, improving the efficiency of the analysis. Figure 3.8 shows the workflow with the activities of the testing cycles.

Similar to training, feature vectors are composed of adaptation options and monitored uncertainties. These feature vectors undergo selection and engineering resulting in updated feature vectors. These feature vectors are then used together with machine learning models to make predictions about the qualities of the adaptation options. We define $Predict$ as follows:

$$Predict : M \times \Lambda \rightarrow \Lambda \times \Omega$$
$$Predict(m, \{\lambda_1, ..., \lambda_n\}) = < \{\lambda_1, ..., \lambda_n\}, \{\omega_1, ..., \omega_n\} >$$

Predictions are made per machine learning model. Prediction takes as input a learning model and a set of feature vectors. The result is a set of predictions associated with the adaptation options (represented as feature vectors). The types of predictions depend on the machine learning model at hand. For instance, a classifier uses classes as representations, while a regressor uses values.

Example: Table 3.8 illustrates predictions for a sample of feature vectors done by

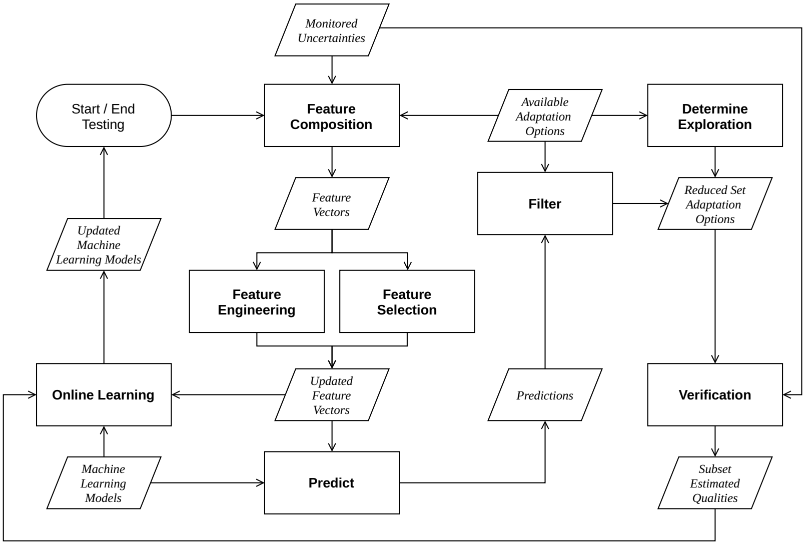


Figure 3.8: Workflow of the runtime stage activities in testing cycles for ML2ASR+.

a classifier that predicts the classes for each feature vector. Class 1 and 0 refer to predictions for the satisfaction and violation of a threshold goal respectively, as defined in the examples of the design stage.

Table 3.8: Example of predictions made for feature vectors taken from Table 3.6.

Distribution	WL M1	WL M2	ABW M1	Predicted class
0.1	0.4	0.2	0.5	1
0.2	0.4	0.2	0.5	1
0.3	0.4	0.2	0.5	0
0.4	0.4	0.2	0.5	0
0.5	0.4	0.2	0.5	0
...	...	...	...	...

The predictions made by the machine learning models are then used to filter the adaptation options. Only the adaptation options that are predicted to meet the adaptation goals are included for verification. For a full explanation and formal foundation of the filter operation, we refer the interested reader to A.2.1. To summarize the filter operation: first adaptation options that are predicted to not meet any of the threshold or setpoint goals are filtered out. Second, out of the remaining adaptation options,

Table 3.9: Example of the selection of adaptation options for verification based on predictions made in Table 3.8 and an additional set of options to explore (in the excerpt, only one such option is shown).

Distribution	WL M1	WL M2	ABW M1	Predicted class	Verify?
0.1	0.4	0.2	0.5	1	Verify
0.2	0.4	0.2	0.5	1	Verify
0.3	0.4	0.2	0.5	0	Explore
0.4	0.4	0.2	0.5	0	Discard
0.5	0.4	0.2	0.5	0	Discard
...	...	...	...	...	...

the adaptation space is further reduced according to the specified granularity value (typically denoted by the letter g from this point on) and the predictions affiliated with the quality of the optimization goal of the system (if present).

A consequence of the filtering approach is that adaptation options that do not meet all the adaptation goals are not selected for verification (since these options are filtered beforehand). However, these adaptation options may be of interest to the system. To deal with this issue we introduced the internal parameter *exploration rate*; the internal parameter that we discussed in *Model selection* during the design stage. The *exploration rate* specifies the percentage of adaptation options that are included for verification, despite these options being filtered out by the filter. The *exploration rate* offers the Machine Learning Module the ability to relearn and correct potentially outdated predictions. For a formal definition of the selection of explored adaptation options, we refer to the A.2.1.

Example: Table 3.9 illustrates the application of filtering based on the predictions made by the Machine Learning Module, as well as extending this set with a selection of additional adaptation options based on the *exploration rate*.

3.5 Algorithms, Models, and Metrics for Evaluating ML2ASR+

Before we present the evaluation of ML2ASR+, we give an overview of the algorithms and models we used for the design of the learning modules, and we define the metrics that we use for the evaluation in Section 3.6.

3.5.1 Algorithms and Models for the Design of the Machine Learning Modules

For the design of the Machine Learning Modules of the evaluation cases, we evaluated different machine learning algorithms and models. The selection of the algorithms and models is based on their common use in the community; for some recent examples see [86, 143, 5]. In addition, the algorithms and models are supported by the widely used scikit-learn implementation kit [269] (see for instance [183, 348, 88]) that we also used for implementing the Machine Learning Module. We summarize now the algorithms and models that we used in the different design steps.

Feature Extraction Algorithms Feature extraction involves two steps: *Feature selection* and *Feature engineering*. For *Feature selection* we have used *Extremely Randomized Tree* algorithms [147] to determine the importance of individual features based on their influence on the target values, i.e., the qualities of the system. The algorithms are based on random forest algorithms (composed of an ensemble of classical decision trees). On top of the random forest algorithms, the extremely randomized tree algorithms introduce extra randomness with the objective of reducing the variance of the machine learning algorithm further (reducing overfitting). We utilized two implementations of the extremely randomized tree algorithms: an implementation of the algorithm for classification and an implementation of the algorithm for regression. After applying the algorithms and detecting relevant features, we adjusted the collected data accordingly for the subsequent activities. For *Feature engineering* we considered 4 scaling algorithms: no scaling algorithm, min-max scaling, max-abs scaling, and standard scaling. We described the min-max and standard scaler briefly in Section 3.4.3. The max-abs scaler rescales the feature values in the range between 0 and 1 relative to their absolute value (e.g. the maximally encountered absolute feature value rescales to a value of 1).

Machine Learning Models For our evaluation, we considered classification and regression machine learning models from the scikit-learn library [269] that are commonly used and support online learning. More specifically, we evaluated Stochastic Gradient Descent classifiers, Passive-Aggressive classifiers, Perceptron classifiers, Stochastic Gradient Descent regressors, and Passive-Aggressive regressors. The Perceptron classifier is a single-layer classifier that utilizes the broadly known Perceptron algorithm (written and published by Frank Rosenblatt [298]). The Stochastic Gradient Descent classifier and regressor both use an SGD learning routine to train their internal models. This learning routine tries to approximate the true gradient of the regularized training error of the model by analyzing a single training data sample at a time (based on [295]). The Passive-Aggressive classifier and regressor [71] both utilize a more aggressive strategy compared to the previously mentioned Perceptron or SGD

Table 3.10: List of used machine learning metrics when evaluating classification machine learning models.

Name	Description	Objective
F1-score	A combined metric of recall (percentage of samples that were retrieved using the classifier) and precision (percentage of samples that were correctly predicted), defined in the interval [0, 1].	Maximize
MCC	The Matthew's correlation coefficient: a metric representing how well the classifier performs compared to making random predictions, defined in the interval [-1, 1] (-1 representing completely incorrect predictions, 0 representing on-par predictions with random predictions, and 1 representing perfect predictions).	Maximize

models by correcting its model in case the internal loss exceeds a threshold, regardless of the step size required to amend the model.

It is important to note that ML2ASR+ supports any other type of machine learning model (classification or regression) that supports online learning and fits within the architecture as described in Figure 3.6.

3.5.2 Metrics for Evaluating Learning Models of ML2ASR+

Learning models need to be evaluated both during the design stage and the runtime stage. During design, evaluation is used to select the best model. For hyper-parameter tuning, we varied a number of parameters, in particular the loss function, penalty function (if applicable), scaler, exploration rate, and warmup-count for the learners that we evaluated[9] (for the list see Section 3.4.3). During runtime, we monitored the selected learning models to validate that they perform well after deployment. Table 3.10 and Table 3.11 show the metrics we used for the evaluation of the learning models for classification and regression respectively.

F1-score combines recall (the percentage of samples that were retrieved by prediction from a specific class) and precision (the fraction of samples that have been predicted to

[9] For scenarios with two threshold goals experimental evaluation revealed that using a single classifier to predict the satisfaction of the two threshold goals was slightly more accurate than using separate classifiers. Therefore, for such settings, we use a single classifier that classifies the adaptation options into four classes as follows: C_0: No threshold goals are satisfied; C_1: Only the first threshold goal is satisfied; C_2: Only the second threshold goal is satisfied; and C_3: Both threshold goals are satisfied. We use these classes in the evaluation in Section 3.6.

Table 3.11: List of used machine learning metrics when evaluating regression machine learning models.

Name	Description	Objective
R2-score	A metric representing how well the model predicts the target value by looking at the variance of the predictions, defined in the interval [0, 1].	Maximize
MSE	The mean of the squares of errors on predictions made by the regressor.	Minimize
MAE	The median absolute error on predictions made by the regressor, less susceptible to outliers.	Minimize
ME	The maximum error on predictions made by the regressor.	Minimize

be of a specific class that are actually part of that class). The F1-score is defined as a value in the interval [0, 1]. A higher F1-score means in general a better-performing classifier. The F1-score is commonly used to judge the performance of classifiers, e.g., [18, 128]. The Matthews correlation coefficient has a value in the range [-1, 1], where 1 represents perfect predictions, 0 represents predictions that are equal to random predictions, and -1 represents incorrect predictions. Hence, this metric enabled us to compare the predictions made by the machine learning model with an approach that predicts based on random selections.

The R2-score represents how well the predictions of the model fit the actual quality values by looking at the variance of the predictions compared to the actual quality values. The R2-score is defined as a value in the interval [0, 1]. A higher R2-score indicates that the model is a better fit for the quality under consideration. The mean squared error refers, as the name suggests, to the mean of the squares of the errors in predictions made by the regressor. The median absolute error offers an alternative to the mean squared error, which is not as susceptible to outliers in the predictions. Lastly, the maximum error gives a good indication of the worst-case prediction made by the regressor. The R2-score serves as a good general metric for evaluating learning models (e.g., used in [303]). The other metrics can provide useful insights depending on the domain and context of the application at hand, see e.g., [132].

3.5.3 Metrics for Evaluating Utility Penalty and Efficiency at Runtime

Table 3.12 summarizes the metrics for utility penalty and efficiency. We refer to these metrics as *Quantitative Metrics* from this point on, since they address the evaluation of

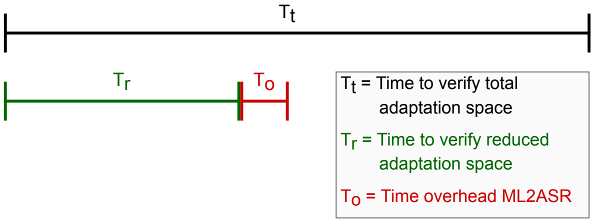

Tt
Tr
To
Tt = Time to verify total adaptation space
Tr = Time to verify reduced adaptation space
To = Time overhead ML2ASR

Figure 3.9: Time variables used in the efficiency metrics.

the *Negligible Utility Penalty* and *Efficiency* requirements, which are both quantitative by nature (see Section 3.2.4). The utility penalty metric is used to address the *Negligible Utility Penalty* requirement of ML2ASR+. In the utility penalty formula, n equals the total number of adaptation cycles, q_o^i represents the quality value in cycle i which would have been chosen in an optimal situation and q_c^i represents the quality value in cycle i chosen by our proposed solution.

To address the *Efficiency* requirement of ML2ASR+, we define three metrics: average adaptation space reduction, learning time overhead, and overall time saved. In the formula of average adaptation space reduction (AASR), $selected$ represents the average number of adaptation options selected by learning over multiple adaptation cycles, and $total$ represents the average of the total number of adaptation options over these adaptation cycles. For the remaining formulas, the parameters T_x refer to one of the time units as defined in Figure 3.9.

3.6 Evaluation ML2ASR+

We evaluate and benchmark ML2ASR+ on two cases from different domains: DeltaIoT [175] and a Service-Based System that is based on TAS [367]. DeltaIoT is a small IoT system with only threshold and optimization goals and a rather small adaptation space of 216 adaptation options. The Service-Based System is a more challenging case with threshold, setpoint, and optimization goals and an adaptation space of 13500 adaptation options.

We start with the evaluation with DeltaIoT and then look at the Service-Based System. We present the results of the evaluation for different scenarios. For the runtime stage, we focus on the evaluation of the requirements with quantitative metrics: utility penalty, average adaptation space reduction, overall time saved, and learning time overhead. We elaborate on the other requirements in the discussion in Section 3.7.

Both applications are evaluated using a simulator. Simulations are run on a computer

Table 3.12: The evaluation metrics used throughout the evaluation section.

Name	Description	Formula	Objective
Utility Penalty	The average difference in the value of quality properties of the system obtained by applying the reference approach, DLASeR, and ML2ASR+.	$\frac{\sum_{i=0}^{n} \lvert q_o^i - q_c^i \rvert}{n}$	Minimize
Average Adaptation Space Reduction	The average proportion of adaptation options that were filtered by DLASeR and ML2ASR+.	$(1 - \frac{selected}{total}) \times 100$	Maximize
Learning Time Overhead	The average proportion of additional time introduced by DLASeR and ML2ASR+ at runtime.	$(\frac{T_o}{T_o + T_r}) \times 100$	Minimize
Overall Time Saved	The average proportion of total time saved of DLASeR and ML2ASR+ (taking into account overhead) compared to the reference approach.	$(1 - \frac{T_r + T_o}{T_t}) \times 100$	Maximize

system with an AMD Ryzen 7 Pro 3700u CPU with 13.7GB of RAM. For the learning approaches, we have used the implementations of the *Scikit-Learn* algorithms (classifiers, regressors, scalers) [269]. The full replication package is available online.[10]

3.6.1 Evaluation with DeltaIoT

We start with introducing DeltaIoT. Then we present two evaluation scenarios and we explain the benchmarks we use. Next, we present the results of the design stage activities and finally the results of the runtime stage activities.

DeltaIoT Application

DeltaIoT is a small Internet-of-Things (IoT) application that offers a smart environment monitoring service. The application is developed by VersaSense.[11] The IoT network comprises 15 Long-Range (LoRa) motes that are deployed at the KU Leuven Computer Science Campus as shown in Figure 3.10. Each mote is equipped with a sensor (temperature, RFID, and infrared) that periodically collects data and sends this data to a gateway. An end-user application processes the data allowing users to monitor the Campus area and take action when needed.

The network uses time-synchronized communication organized in cycles. Each cycle consists of a number of communication slots between a sender and a receiver mote. The slots are allocated from the leaf nodes of the network towards the gateway. Each mote has an internal buffer to store its own generated data and data received from other motes. When a mote is allocated a communication slot, it sends the data of the buffer to the receiving mote of the slot.

Uncertainties We consider two important types of uncertainties: dynamics in the traffic load and interference of the wireless network. Dynamics in the traffic load result from variations in the frequency that sensors take samples and transmit data. For example: a temperature sensor collects and sends measurements periodically, while an RFID sensor only sends data when it is available, e.g., when a person scans an RFID badge. As a result, the load of packets that need to be sent to the gateway fluctuates. Interference of the wireless network arises from dynamic conditions in the environment, such as weather conditions or the presence of other wireless networks. Interference may result in the loss of packets communicated over the link. Figure 3.11 shows excerpts

[10] https://people.cs.kuleuven.be/danny.weyns/material/ML2ASR/

[11] www.versasense.com

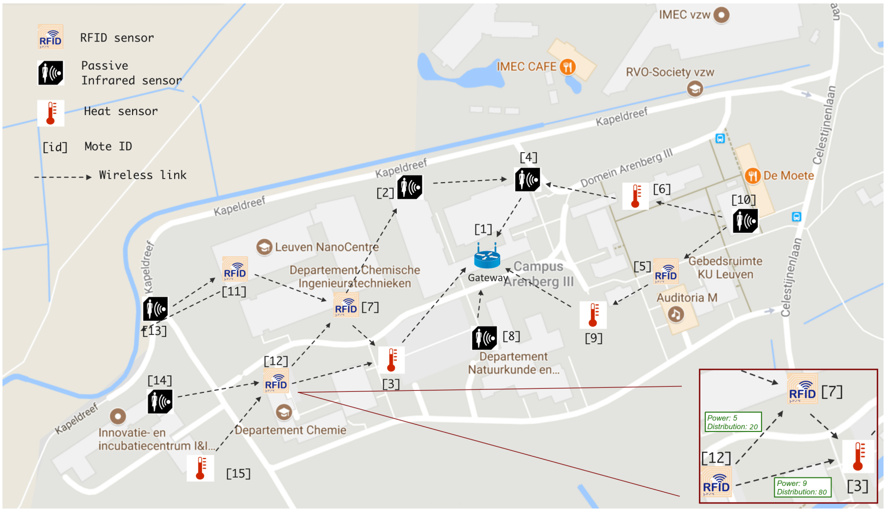


Figure 3.10: Deployment of DeltaIoT at the KU Leuven campus.

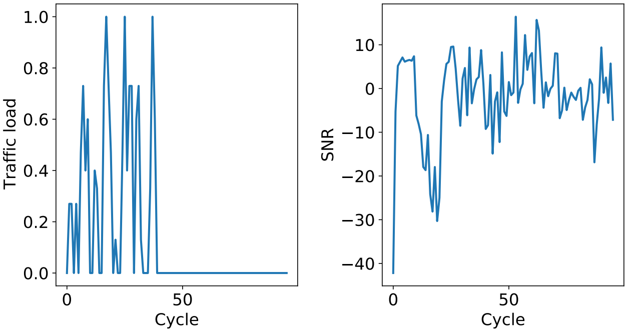


Figure 3.11: Example of traffic load generated by mote 13 and network interference on the link between mote 12 and mote 3 over 100 cycles.

with data of both types of uncertainties over a period of time. This data is based on measurements of DeltaIoT in the field.[12]

Quality Goals Besides *what* the network should do, i.e., collecting data at the gateway, stakeholders of DeltaIoT also have demands on *how* this is done, i.e., the

[12] Network interference is represented as the Signal-to-Noise ratio (SNR). An SNR below 0 may lead to the loss of packets.

quality of the transmission. We consider three quality goals of DeltaIoT: packet loss, latency, and energy consumption. As explained above, packet loss depends on network interference. Latency depends on the traffic load in the network since only a limited number of packets can be transmitted during a time slot. The remaining packets remain in the buffers for communication in the next slot, causing delays in the transmission of data. Lastly, energy consumption depends on the number of packets that motes need to communicate and the power that is used to communicate packets. Evidently, stakeholders prefer to keep the packet loss, latency, and energy consumption low. However, these qualities are conflicting, for instance, using less energy (lower power) over a network link may result in higher packet loss as the signal may get lost in the noise along the link.

Adaptation of the IoT Network To ensure quality goals during operation, DeltaIoT offers a management interface that is connected to the gateway. This interface can be used to observe the behavior of the network (e.g., the interference along links, the packets lost over a time period, etc.) and change the settings of the motes in the network. Here, we consider two types of settings. First, the power used to transmit packets over an outgoing link of a mote can be set in a range $[0 \ldots 15]$ (0 is minimum power and 15 maximum power). Sending packets with a higher power setting reduces the chance of packets being lost over a noisy link, but it consumes more power. Second, for motes with more than one outgoing link, the distribution of the packets over these links can be set. This way, the transmission of packets along paths with high interference or high traffic can be reduced or avoided, yet the packets may follow a longer path requiring more energy. Since motes in DeltaIoT have at most two parent motes, we consider the following distribution settings for these motes: $0 - 100$, $20 - 80$, $40 - 60$, $60 - 40$, $80 - 20$, and $100 - 0$. An example configuration is shown in Figure 3.10 (bottom right corner). Here a power setting of 5 is used for the upper link that transmits 20% of the packets, and a power setting of 9 is used for the bottom link that transmits 80% of the packets.

Without self-adaptation, an operator is responsible for ensuring the quality goals by monitoring the network and adjusting the settings using the management interface. This is a tedious and costly task that is often not very efficient. To that end, we add a managing system (MAPE feedback loop) to the system that connects with the management interface to automate the adaptation of the settings. We use such a setting for the evaluation of ML2ASR+.

Evaluation Setup

For the evaluation with DeltaIoT, we used a simulation of the network with 15 motes as shown in Figure 3.10. We applied 300 communication cycles of the network that correspond with a wall clock time of around three days. We used uncertainty profiles for the traffic load of motes and network interference that are based on measurements of the physical network. For the traffic load, motes generate between 0 to 10 packets per cycle. The level of interference (SNR) fluctuates between -40dB and +15dB. Figure 3.11 shows two example profiles we used.

Adaptation Goals We devised two evaluation scenarios with learning tasks for different adaptation goals summarized in Table 3.13. In scenario 1, learning needs to predict and filter adaptation options based on these two threshold goals. In scenario 2, learning needs to additionally predict and filter adaptation options for an optimization goal. Note that a threshold goal that should keep the average packet loss under 10% over a period of 12 hours, implies that on average 90% of the transmitted packets should be received by the gateway. On the other hand, a threshold goal that should keep the average latency under 5% over a period of 12 hours, implies that on average at least 95% of packets generated in a cycle should be received by the gateway within that cycle.

Adaptation Settings Adaptation options are composed in each cycle following two steps. Firstly, the power setting is determined for each link of each mote. These settings are determined such that the current Signal to Noise ratio (SNR) over each link is at least 0dB. The adaptation options are then determined based on the possible distribution settings for outgoing links of motes with two parents ($0-100$, $20-80$, etc.). As such, the complete adaptation space for the DeltaIoT case consists of $6^3 = 216$ adaptation options.[13] The MAPE feedback loop and the quality models have been designed as networks of timed automata models. These models are directly executed at runtime using the ActivFORMS execution engine [174]. The analysis of the adaptation options is performed using the runtime models by applying statistical model checking at runtime using runtime statistical model checking with Uppaal-SMC [79].

Benchmarks We benchmark ML2ASR+ using three approaches. First, we use a baseline approach that analyzes the whole adaptation space without using machine learning. Second, we use a competing approach, called DLASeR, that applies a deep neural network to reduce adaptation spaces [349].[14] We have rerun the results presented in [349] to ensure that the same settings were used to compare ML2ASR+ and DLASeR.

[13] The power setting for each individual mote remains fixed within a single adaptation cycle. As such, adaptation options vary based on the different distribution settings. For the configuration of DeltaIoT, there are six variations of distribution settings for a mote with two parents, and there are three motes that have two parents, hence the total of 216 adaptation options.

[14] We selected [349] since this approach is conceptually similar to ML2ASR+, relying on learning to provide first-class support for adaptation space reduction. Most other related approaches mix adaptation space reduction with decision-making, while our main focus is on adaptation space reduction. Since the

Table 3.13: Evaluation scenarios for DeltaIoT with the adaptation goals considered for learning.

	Adaptation goal 1	**Adaptation goal 2**	**Adaptation goal 3**
Scenario 1	*Threshold*: the average packet loss over 12 hours should not exceed 10% of the messages sent.	*Threshold*: the average latency over 12 hours should not exceed 5% of the cycle time.	N/A
Scenario 2	*Threshold*: the average packet loss over 12 hours should not exceed 10% of the messages sent.	*Threshold*: the average latency over 12 hours should not exceed 5% of the cycle time.	*Optimization*: the average energy consumption over 12 hours should be minimized.

Third, as a sanity check, we used an approach that selects a subset of adaptation options randomly. We average the obtained results over 10 runs to reduce variability. We highlight the results of this random approach separately and focus on statistically relevant differences.

In the next sections, we start with the evaluation results of the design stage. Then we present the results of the runtime stage. To conclude, we summarize the machine learning activities in both stages.

Design Stage Evaluation with DeltaIoT

Data Collection We collected data from 300 cycles of DeltaIoT to derive the machine learning modules for both scenarios, each cycle containing 216 data points. Experiments showed that 300 cycles for the design stage activities ensured that the learners performed well during runtime. As explained in Section 3.4, the collected data consists of a set of feature vectors that represent adaptation options with uncertainties, and quality vectors that represent the qualities of the corresponding adaptation options.

Feature Extraction After collecting the data, we applied *Feature extraction*. The first activity, *Feature selection*, removes features from the collected feature vectors that do not have an influence on the resulting qualities in the system. Based on feature extraction 34 of the original 65 individual features were selected as relevant. For instance, all features related to SNR were selected. An example of a feature that was not selected is the load of motes that generate a constant number of packets, for instance, motes that periodically track the temperature in the environment. Next, we use the pruned data to perform the second activity of feature extraction: *Feature engineering*. For both scenarios, we selected the Min-Max scaler for threshold goals. For the optimization goal, no scaler was selected as this provided the best results. As *Feature engineering* closely ties with *Model selection* we explain the results below. Section 3.5.1 describes how *Feature extraction* was done. For detailed results, we refer to the website with the replication package.

Machine Learning Model Identification In the first activity, *Model evaluation*, we evaluated three different types of classifiers and two types of regressors. For the second activity, *Model selection*, we closely examined the evaluation metrics for each model to make a decision on the learning models to be used at runtime; Section 3.5.2 describes how this was done. Table 3.14 summarizes the chosen machine learning

preliminary version of ML2ASR [284] only supports one type of goal, we have not used it as a benchmark in this study.

Table 3.14: Summary of the chosen machine learning models during *Model evaluation* and *Model selection* for DeltaIoT (abbreviations: F1 → F1-score, MCC → Matthews Correlation Coefficient, R2 → R2-score, MSE → Mean Squared Error, MAE → Median Average Error, ME → Maximum Error, Pl → Packet loss, La → Latency, Ec → Energy Consumption, S1 → Scenario 1, S2 → Scenario 2).

	Goal(s)	Model	Metrics
S1	$\mathcal{T}^{\text{Pl}}_{<10\%}$, $\mathcal{T}^{\text{La}}_{<5\%}$	SGD Classifier $^{\text{(log loss, l1 penalty)}}$ MinMax Scaler	F1: 0.818, MCC: 0.715
S2	$\mathcal{T}^{\text{Pl}}_{<10\%}$, $\mathcal{T}^{\text{La}}_{<5\%}$	SGD Classifier $^{\text{(log loss, l1 penalty)}}$ MinMax Scaler	F1: 0.818, MCC: 0.715
	$\mathcal{O}^{\text{Ec}}_{min}$	Passive Aggressive Regressor (squared epsilon insensitive loss) No Scaler	R2: 0.833, MSE: 0.004, MAE: 0.043, ME: 0.269

models and their corresponding metric values obtained during the evaluation process. A.2.2 provides a detailed description of the chosen machine learning models.

Exploration Rate and Warm-up Count Finally, we tested different exploration rates (extra random adaptation options selected for verification) and warm-up counts (the number of training cycles to initialize the learning model) that are required for the runtime stage, see Section 3.5.2. We selected 5% as the exploration rate and 45 cycles (of 300) as the warm-up count. For detailed results, see Appendix A.2.

Runtime Stage Evaluation with DeltaIoT

Hypothesis For the evaluation of the runtime stage of ML2ASR+ with DeltaIoT we use the following hypotheses:

H1: The utility penalties when applying ML2ASR+ are negligible compared to the reference approach.

H2: The utility penalties when applying ML2ASR+ is not significantly higher compared to DLASeR.

H3: ML2ASR+ significantly reduces the adaptation space and hence the time required for verification compared to the reference approach.

Table 3.15: Values of the machine learning metrics for the runtime stage evaluation of the machine learning models of DeltaIoT (abbreviations: Pl → Packet loss, La → Latency, Ec → Energy Consumption, S1 → Scenario 1, S2 → Scenario 2).

		F1-score	**Matthews correlation coefficient**
S1	$\mathcal{T}^{\text{Pl}}_{<10\%}, \mathcal{T}^{\text{La}}_{<5\%}$	0.757	0.646
S2	$\mathcal{T}^{\text{Pl}}_{<10\%}, \mathcal{T}^{\text{La}}_{<5\%}$	0.743	0.608

		R2-score	**Mean squared error**	**Median absolute error**	**Maximum error**
S1	$\mathcal{O}^{\text{Ec}}_{min}$	0.799	0.0045	0.0446	0.270

H4: The reduction of adaptation spaces with ML2ASR+ is not significantly lower compared to DLASeR, nor does ML2ASR+ require significantly more time for adaptation space reduction.

Granularities for Adaptation Space Reduction with an Optimization Goal In scenario 2, ML2ASR+ predicts the energy consumption in the network (optimization goal), on top of predicting packet loss and latency (threshold goals). After filtering out options that are predicted to satisfy the threshold goals (be of class C_3, see Section 3.5.2), ML2ASR+ reduces the adaptation space further based on the energy consumption predictions. We evaluate two cases: a reduction to at most 25 options and at most 10 options, corresponding to granularity values of 25 and 10, respectively.

Quality of the Learning Models Table 3.15 summarizes the results for the quality of the machine learning models during runtime. The F1-score acquired is 0.757 and the Matthews Correlation Coefficient is 0.646 in scenario 1. This is in line with what we expect when comparing to the metrics retrieved during *Model selection* in the design stage: an F1-score of 0.818 and a Matthews Correlation Coefficient of 0.715. For scenario 2 we obtained similar results for classification, albeit slightly worse results due to the increased reduction of the adaptation space. The regressor, which handles the energy consumption optimization goal, has an R2-score of 0.799, a mean squared error of 0.0045, a median absolute error of 0.0446, and a maximum error of 0.27. Overall, the results in the runtime stage are good, showing similar results to other studies [260, 22].

Table 3.16: Values of the metrics for the runtime stage evaluation of requirements of DeltaIoT (abbreviations: Pl → Packet loss, La → Latency, Ec → Energy Consumption, S1 → Scenario 1, S2 → Scenario 2, G → Granularity).

		Utility penalties			**AASR**	**Overall time saved**	**Time overhead**
		Pl	***La***	***Ec***			
S1	**N/A**	0.045%	0.025%	N/A	56.5%	62.81%	0.05%
S2	**G 25**	0.091%	0.073%	0.008mC	88.5%	90.82%	0.26%
	G 10	0.515%	0.299%	0.019mC	95.4%	96.37%	0.56%

Summary of Results for Quantitative Metrics Table 3.16 summarizes the results of the evaluation for the quantitative metrics. We discuss these results now in detail.

Utility Penalties Figure 3.12 shows the results for utility penalties. Note that the reference approach that exhaustively verifies all adaptation options provides optimal adaptation[15]. First, we take a closer look at the threshold goals. Afterward, we look at the optimization goal.

Threshold Goals When inspecting the results in detail, we notice that the values for the threshold goals with ML2ASR+ are very close to those obtained with the reference approach. The average values of the penalties are respectively 0.045% and 0.025% for packet loss and latency in scenario 1, and 0.515% and 0.299% for the worst-case of scenario 2 with a granularity value of 10. The marginal increases that result from adaptation space reduction with ML2ASR+ do not impede on the satisfaction of both goals compared to the reference approach in scenario 1 and scenario 2 with a granularity of 25. On the other hand, in scenario 2 with a granularity of 10, we notice that the threshold goals were not satisfied in 7 additional adaptation cycles after adaptation space reduction took place (cycles for which the reference approach does not violate the requirements). These results show that a lower granularity that substantially reduces the adaptation space for analysis may result in penalties for the quality properties of interest. This trade-off has to be carefully considered when making decisions about the granularity of adaptation space reduction.

Comparing the results of ML2ASR+ with DLASeR, we observe that the satisfaction of the threshold goals is not impeded in both scenarios as opposed to the few violations in scenario 2 with a granularity value of 10 when using ML2ASR+ (which corresponds

[15] Note that due to the nature of the application, in some adaptation cycles, no adaptation option might exist that meets both threshold goals.

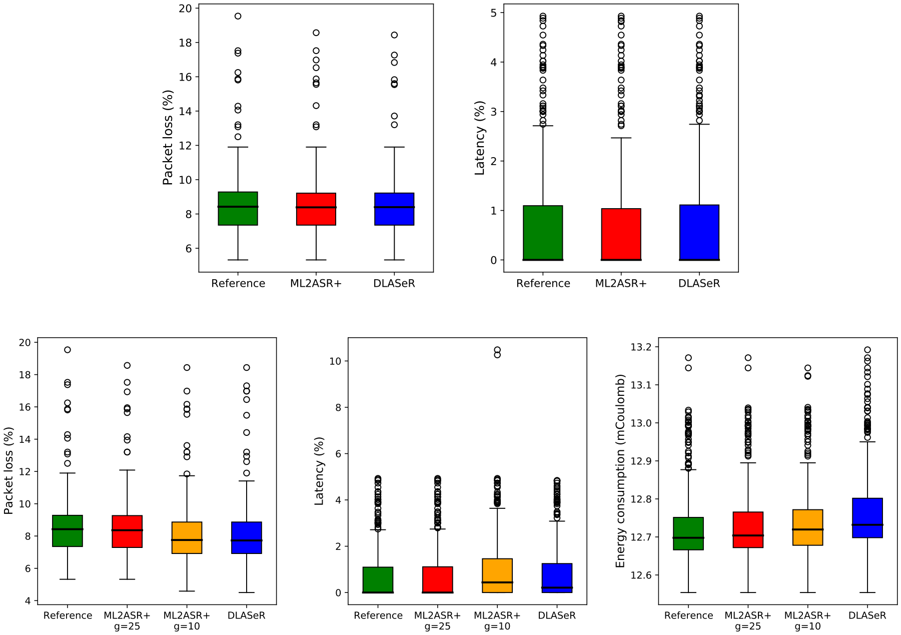


Figure 3.12: Utility penalties for scenario 1 (top) and scenario 2 (bottom) of DeltaIoT compared to the reference approach and DLASeR.

to the strategy DLASeR employs: rank adaptation options based on predicted energy consumption, and subsequently look for adaptation options that meet both threshold goals).

Optimization Goal The results for the optimization goal show that the differences between the average values of energy consumption with the reference approach ($12.719mC$), ML2ASR+ ($12.723mC$), and DLASeR ($12.724mC$) are marginal. This indicates that the reduced adaptation space most of the time also includes the adaptation option with the lowest energy consumption. For scenario 2 we observe values of $0.008mC$ and $0.019mC$ with ML2ASR+ for granularity values of 25 and 10, respectively. Here we also notice a similar trade-off between granularity values and utility penalty: a more fine-grained reduction carries the risk of adapting the system less optimally compared to a less constrained strategy. Note that the penalty compared to the reference approach is still acceptable ($12.719mC$ mean energy consumption for the reference approach, $12.727mC$ for ML2ASR+ with a granularity value of 25 and $12.739mC$ for ML2ASR+ with a granularity value of 10) considering the significant time gain that both approaches offer (see below). For DLASeR we observe an average

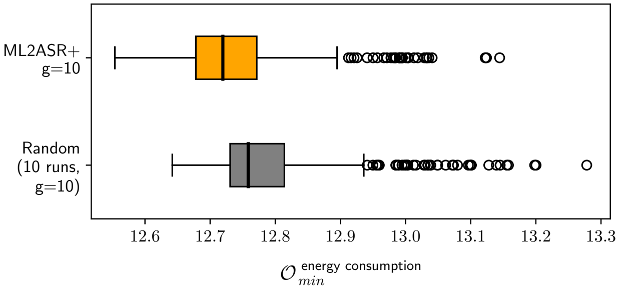


Figure 3.13: Comparison of energy consumption for 10 random runs compared to ML2ASR+ in scenario 2 of DeltaIoT.

energy consumption of $12.769mC$ with a utility penalty for energy consumption of $0.038mC$, meaning that ML2ASR+ performs quite well compared to the competing approach.

Sanity Check with Random Approach We compared ML2ASR+ with a simple approach that randomly selects adaptation options (using an average of 10 random runs). For the threshold goals, packet loss and latency, both approaches satisfy the goals in both scenarios. However, the results show that the random approach violates the threshold goals for 28 adaptation cycles (of a total of 300 cycles), which is 16 cycles more compared to ML2ASR+. For the optimization goal of energy consumption in scenario 2, the Wilcoxon signed rank statistical test [380] showed a significant difference between the random approach and ML2ASR+ both for each random run and on the average of 10 random runs (for the latter we measured a p-value of $1.2e^{-15}$ with alpha level 0.05)[16]. Figure 3.13 shows the distribution of the energy consumption of the IoT networks in scenario 2 with both approaches. The average energy consumption with the random approach is $12.800mC$ compared to an energy consumption of $12.739mC$ for ML2ASR+. While the differences in the absolute value of energy consumption are relatively small, the difference is statistically relevant. The second case will show that for more complex application scenarios, the impact is practically relevant.

[16] We add the caveat here that the results of these tests are not a general claim: the findings confirm a statistical difference in the 10 random runs we did, but this claim may not necessarily hold for other sets of 10 random runs.

Hypotheses H1 (negligible utility penalties compared to reference approach) and H2 (utility penalties not significantly higher compared to DLASeR). The results show that the utility penalties when applying ML2ASR+ are negligible compared to the reference approach. ML2ASR+ with a low granularity value in one scenario did not satisfy the threshold goals in all cycles, emphasizing the importance of a good selection of granularity. Comparing ML2ASR+ with DLASeR, we notice that the utility penalties remain negligible for both threshold goals and the optimization goal. In conclusion, we can accept hypotheses H1 and H2.

Average Adaptation Space Reduction Figure 3.14 (left) shows the size of the adaptation spaces for the three evaluated approaches. During the first 45 training cycles, when the Machine Learning Module of ML2ASR+ is not exploited, all adaptation options are analyzed (multiple data points overlapping at 216 adaptation options for ML2ASR+). In the case of DLASeR, there is only a single entry at 216 adaptation options corresponding to the only training cycle.

Applying ML2ASR+ results in an Average Adaptation Space Reduction (AASR) of 56.5% for scenario 1, 88.5% for scenario 2 with a granularity value of 25, and 95.4% with a granularity value of 10. For scenario 1 this means that, on average, more than half of the adaptation options available in the adaptation space are filtered out before verification is applied. For scenario 2, we obtained results that match in most cases the granularity value.

For DLASeR we obtain an Average Adaptation Space Reduction of 58.8% for scenario 1, a result similar to the one obtained with ML2ASR+. However, for scenario 2, DLASeR works differently: the approach relies on deep learning models starting with predicting the energy consumption of all adaptation options; then it iterates over the adaptation options (from low energy consumption predictions to high) until an adaptation option is found that meets both threshold goals. This way, DLASeR achieves an average adaptation space reduction of 94.19% in scenario 2.

Learning Time Overhead Figure 3.14 (right) shows the overhead introduced by ML2ASR+ (red and yellow lines) and DLASeR (blue line). The learning overhead is on average less than 1% of the total time necessary to both reduce and verify the reduced adaptation space for ML2ASR+ in both scenarios. Concretely, the overhead of ML2ASR+ is at most 4.28ms, which is less than 10% of the time required to verify a single adaptation option. We conclude that this overhead is negligible compared to the time necessary to verify the selected subset of adaptation options.

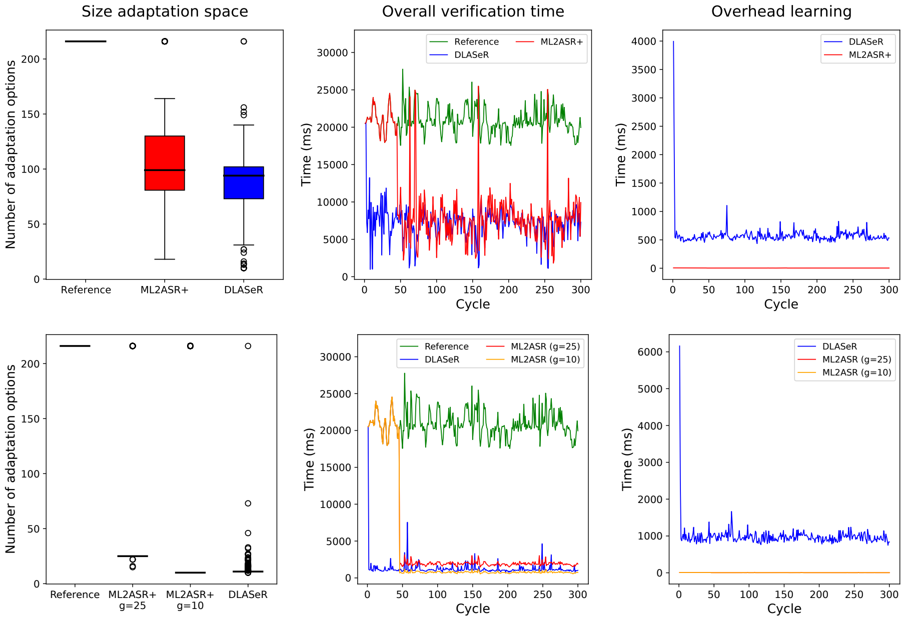


Figure 3.14: Number of verified adaptation options with ML2ASR+ and DLASeR (left), overall time used (middle) and overhead (right) compared to the reference approach for scenario 1 (top row) and scenario 2 (bottom row) of DeltaIoT.

DLASeR on the other hand introduces a slightly higher overhead of 8.34% in scenario 1. Even though this number is higher, it is important to bear in mind that the overhead is still a minor part of the overall time required for verification of the reduced adaptation space. However, for scenario 2 we notice a significantly higher overhead of 45.96% for DLASeR due to the strategy DLASeR employs for adaptation space reduction. Even though DLASeR reduces the adaptation space to a small subset, the overhead is significantly higher than ML2ASR+.

Overall Time Saved Figure 3.14 (middle) shows the overall time used to analyze all the adaptation options in each cycle with the reference approach (green), and the time used to reduce and verify the adaptation space with ML2ASR+ (red and yellow) and DLASeR (blue). We observe that in scenario 1 ML2ASR+ saves more than half of the time (62.81%) for verifying the reduced adaptation space compared to the reference approach. This observation is in line with the average adaptation space reduction, resulting in a significant time gain compared to the reference approach. DLASeR shows results that are also in line with the average adaptation space reduction, albeit slightly worse due to the higher overhead introduced by the approach (62.59% of the time saved). Similarly, for scenario 2 we observe results closely aligned with the adaptation space reduction metric since the learning time is negligible: 90.82% and 96.37% for granularity values 25 and 10 respectively. For DLASeR in scenario 2 we notice an average time saved of 89.69%.

Hypotheses H3 (significant reduction of adaptation spaces and time gain) and H4 (adaptation space reduction comparable to DLASeR). ML2ASR+ realizes a significant reduction of the adaptation space of 56.5% for scenario 1 and over 90% for scenario 2, resulting in an overall time saving for analysis of 62.81% compared to the reference approach in scenario 1 and again over 90% for scenario 2. ML2ASR+ and DLASeR realize a similar adaptation space reduction in scenario 1 and scenario 2 with a granularity value of 10. Yet, the time required for adaptation space reduction with ML2ASR+ is negligible ($< 1\%$), and small to significantly larger for DLASeR (8.34% to 45.96%). In conclusion, we can accept hypotheses H3 and H4.

Summary of the design stage and runtime stage machine learning activities

Table 3.17 summarizes the number of inputs, features, objective variables and metrics of the learning activities for DeltaIoT in each of the design stage and runtime stage activities.

3.6.2 Evaluation with the Service-Based System

We present now the evaluation results of the second case: a service-based system. We follow the same structure: we start by introducing the application, evaluation scenarios, and the benchmarks we use. Then, we present the results of the design stage activities and finally the results of the runtime stage activities.

Service-Based System Application

Self-care enables patients to self-manage their illness [32, 294]. We consider a concrete example of a smartwatch application that analyzes data of patients and visualizes the result for the patient [367]. Our focus is on the underlying service-based system that processes patient data. Figure 3.15 describes the workflow of this system.

The system consists of a set of services that perform tasks. The services are composed of a workflow with two main branches. The “sleep branch” analyzes the data of patients

Table 3.17: The number of inputs, features, objective variables, and metrics for the activities of the machine learning pipeline of scenario 2 of DeltaIoT in the design and runtime stage. The prediction column is marked in red to indicate that it is not used yet in the training cycles. The number of inputs for online learning is determined by the number of options that could be verified by the Verifier. Abbreviations: “f vectors” → feature vectors, “q vectors” → quality vectors, “Pl” → packet loss, “Ec” → energy consumption, “La” → latency.

	Design stage		*Runtime stage*			
	Feature Extraction	**Machine Learning Model Identification**	**Feature Extraction**	**Prediction**	**Verification**	**Online Learning**
Number of inputs	$300 * 216$ f vectors $300 * 216$ q vectors	$300 * 216$ f vectors $300 * 216$ q vectors	216 f vectors	216 f vectors	216 f vectors	X f vectors X q vectors
Number of features	65 features	34 features	65 features	34 features	x	65 features
Objective variables	1. Pl + La class 2. Ec value	1. Pl + La class 2. Ec value	x	1. Pl + La class 2. Ec value	x	1. Pl + La class 2. Ec value
Metrics	x	1. F1, MCC 2. R2, MSE, MAE, ME	x	x	x	x

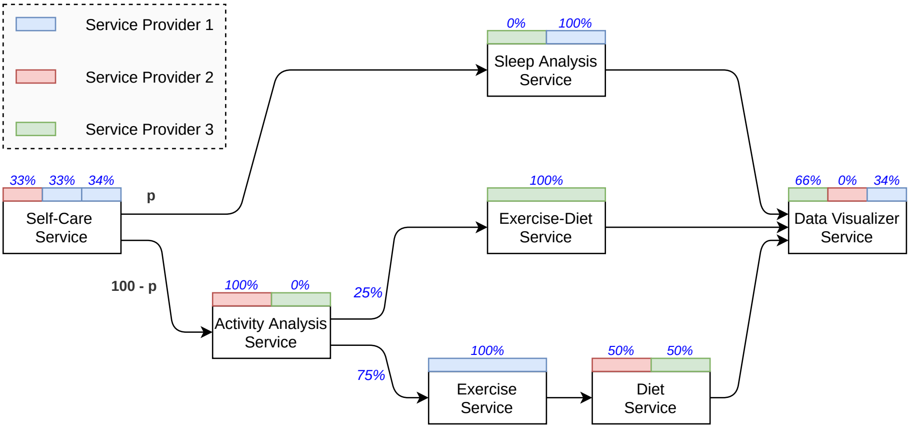


Figure 3.15: Workflow of the Service-Based System health monitoring and processing application. Blue percentages denote an example system configuration of the application (further elaborated on in Section 3.6.2).

when they sleep that can be visualized for the patient afterward. The "awake branch" analyzes data from different activities, processing the data using exercise and diet services and visualizing the results to the patient. Each branch fulfills its tasks using different services. For example, *Exercise service* processes activity data regarding exercises and makes recommendations. Similarly, the *Diet service* processes activity data regarding dietary information and makes recommendations. The *Exercise-Diet service* combines both these responsibilities in a single service providing an alternative path in the workflow.

The workflow defines service types that need to be instantiated. Three different service providers offer such service instances. These instances are marked in the workflow by small colored rectangles at the top of each service symbol. Service instances differ in the qualities they provide (e.g., response time) and also the cost of using them. During operation, a concrete set of service instances is selected and used to handle incoming service requests.

Uncertainties Each service provider is characterized by two parameters in our evaluation: its workload and the available bandwidth of its network. Both these parameters fluctuate at runtime representing uncertainties. These fluctuations in turn affect the qualities of the service instances they provide, including the failure rate, response time, and cost (see below). Figure 3.16 shows the models we used for the fluctuations of the qualities of service instances per service provider. Failure rate and cost increase with higher load, while response time decreases with lower bandwidth.

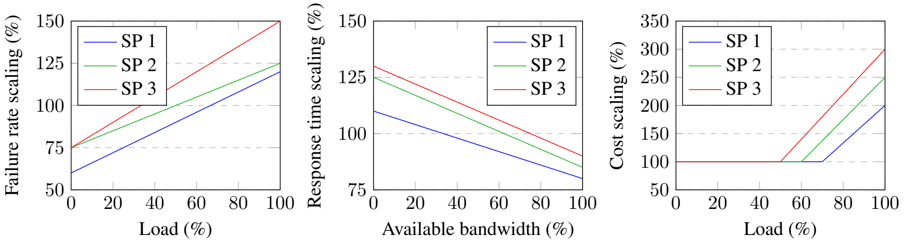


Figure 3.16: Effect of load and available bandwidth of individual service providers (SPs) on system qualities.

Besides fluctuations in workload and the available bandwidth, the system has to deal with an additional uncertainty, namely the distribution of service requests for sleep analysis (sleep branch) or activity analysis (awake branch). This distribution, denoted by the value p in the workflow, may change over time depending on the patient's behavior.

Quality Goals In the evaluation, we consider three key qualities for stakeholders of the service-based system: the failure rate of service invocations, the response time, and the cost of invocations. Each service instance is characterized by a specific failure rate, response time, and cost. Hence, the overall quality values for service requests are determined by the individual service instances that are selected to handle these requests. In particular, the overall failure rate is determined by the multiplication of each failure rate associated with the selected service instances. As an example, assume we invoke two service instances, each characterized by a failure rate of 5%. The overall failure rate of a service request then corresponds to $1 - (0.95 * 0.95) = 0.0975$, i.e., 9.75%. The overall response time of service requests is simply determined by the sum of the individual response times associated with these selected service instances. Similarly, the overall cost is determined as the sum of the costs associated with individual service instances.

Clearly, stakeholders want to keep the failure rate, response time, and cost as low as possible. Yet, these qualities conflict. Invoking a service with a lower failure rate and/or lower response time will usually imply a higher cost. However, the selection of services is complicated by uncertainties. For instance, the cost to invoke a service of a service provider may increase when the service provider is under heavy load. Similarly, the failure rates and the response times of the provided service instances fluctuate in time.

Adaptation of the Service-Based System Given the fluctuations in load and available bandwidth of service providers and changes in patient behavior, the selection of service instances may be changed dynamically based on the changing conditions. To that end, the system can be configured such that the requests are distributed in a particular way over different instances. In the evaluation setting, we use service types with 2 and 3 instances. For services with 2 instances, the system offers 3 possible configurations: 0/100%, 50/50%, and 100/0%. For services with 3 instances, there are 10 possible configurations: 0/0/100%, ..., 0/33/67%, ... 100/0/0%. This way, preference can be given to services with better actual quality values, or services can even be (temporally) avoided if necessary. In addition, the parameter α that determines which path is taken in the awake branch (distinct services for the exercise and diet tasks or a combined service) can be set to one of four values: 0%, 25%, 50%, 75%, and 100%. Figure 3.15 shows (on top of the general workflow) an example configuration of the workflow (with concrete selections for service instances and α set to 25%).

Without self-adaptation, it is practically infeasible for an operator to change the service selection dynamically. Hence, the only option for an operator would be to allocate a predefined set of possible service instances to the system and perform a coarse-grained adaptation. However, this would result in a sub-optimal solution or even worse in case particular services would fail unexpectedly. To that end, we add a managing system (MAPE-based feedback loop) to the system that monitors the changing conditions and adapts the service instances of the workflow dynamically when needed to maintain the stakeholder goals (failure rate, response time, and cost).

Evaluation Setup

We used a simulation of the setup as shown in Figure 3.15. We considered 30.000 service requests that are generated sequentially and processed individually by the system. Adaptation is triggered every 100 requests, resulting in 300 feedback loop iterations. The workload and available bandwidth are modeled as stochastic variables that gradually change during the operation of the system (between 0 and 100%). The change in values occurs by sampling a normal distribution with a standard deviation of 1.7, increasing or decreasing the bandwidth and workload of service providers. The factor p is initially set at 50% and is modeled similarly to the workload and available bandwidth.

Adaptation Goals We devised two scenarios of the service-based application as illustrated in Table 3.18. In scenario 1 we consider three adaptation goals: two threshold goals and an optimization goal. Learning first filters out adaptation options based on the threshold goals, and subsequently orders and reduces the adaptation space according to the optimization goal. In scenario 2, learning has to deal with all three types of goals (threshold, setpoint, and optimization).

Adaptation Settings The adaptation space for the Service-Based System is fixed. The adaptation options are determined based on the distribution of available service instances per service type and the setting of α. Concretely, the total adaptation space comprises $5 * 10 * 10 * 3 * 3 * 3 = 13500$ adaptation options.[17] Similarly to the DeltaIoT case, we designed a MAPE feedback loop and quality models as networks of timed automata models that are directly executed using ActivFORMS [174]. For the analysis of the parameterized quality models, the feedback loop applies statistical model checking at runtime using Uppaal-SMC [79].

Benchmark We benchmark ML2ASR+ again with a reference approach that analyzes the whole adaptation space without using learning and with DLASeR [349], a state-of-the-art approach. As a sanity check, we use again an approach that selects adaptation options randomly over a set of runs to reduce variability. It is important to note that the results for the different approaches are obtained from identical configurations and parameter settings of the application. For ML2ASR+ and DLASeR the data is collected during simulation. Yet, for the reference approach, the data is collected during the design stage since analyzing the complete adaptation space for one cycle takes around 2 hours.

In the next sections, we start with the results of the design stage. Then we present the results of the runtime stage. To conclude, we summarize the machine learning activities in both stages.

Design Stage Evaluation with the Service-Based System

Data Collection, Feature Extraction, Machine Learning Model Identification The design stage activities for the Service-Based System followed the same procedure as the activities for DeltaIoT (see Section 3.6.1). First, we collected data from the system used to derive the machine learning modules for both scenarios. This data consisted of a set of feature vectors (composed of adaptation options and uncertainties in the application) and a set of quality vectors. We collected data for 100 adaptation cycles corresponding to 10.000 service requests, each adaptation cycle containing 13500 data points. Then we performed *Feature extraction*. During *Feature selection*, all 22 features were selected as relevant, e.g., all features concerning the distribution of

[17] The numbers are composed following the description in Section 3.6.2: 5 represents the number of instantiations for α, 10 represents the number of options for services with 3 instances (in total 2) and 3 represents the number of options for services with 2 instances (in total 3).

Table 3.18: System scenarios with their adaptation goals for the self-care application.

	Goal 1	**Goal 2**	**Goal 3**
Scenario 1	*Threshold*: the average failure rate should not exceed 10%.	*Threshold*: the average response time should not exceed 10ms.	*Optimization*: the average cost should be minimized.
Scenario 2	*Threshold*: the average failure rate should not exceed 10%.	*Setpoint*: the average response time should be kept at 10ms.	*Optimization*: the average cost should be minimized.

Table 3.19: Summary of the chosen machine learning models during *Model evaluation* and *Model selection* in the Service-Based System; Abbreviations: F1 → F1-score, MCC → Matthews Correlation Coefficient, R2 → R2-score, MSE → Mean Squared Error, MAE → Median Average Error, ME → Maximum Error, Fr → Failure rate, Rt → Response time, C → Cost, S1 → Scenario 1, S2 → Scenario 2

	Goal(s)	**Model**	**Metrics**
S1	$\mathcal{T}^{\text{Fr}}_{<10\%}$, $\mathcal{T}^{\text{Rt}}_{<10ms}$	SGD Classifier (hinge loss, l1 penalty) No Scaler	F1: 0.895, MCC: 0.812
	$\mathcal{O}^{\text{C}}_{min}$	Passive Aggressive Regressor (squared epsilon insensitive loss) No Scaler	R2: 0.906, MSE: 1.753, MAE: 0.901, ME: 5.981
S2	$\mathcal{T}^{\text{Fr}}_{<10\%}$	SGD Classifier (hinge loss, elasticnet penalty) Standard Scaler	F1: 0.933, MCC: 0.866
	$\mathcal{S}^{\text{Rt}}_{10ms,\epsilon=0.25ms}$	Passive Aggressive Regressor (squared epsilon insensitive loss) No Scaler	R2: 0.860, MSE: 0.035, MAE: 0.123, ME: 0.976
	$\mathcal{O}^{\text{C}}_{min}$	Passive Aggressive Regressor (epsilon insensitive loss) No Scaler	R2: 0.906, MSE: 1.753, MAE: 0.901, ME: 5.981

service requests over service instances and all the features concerning the load of the service providers. During *Feature engineering*, we determine which scaling algorithms to use to adjust feature values, following Section 3.5.1. Lastly, for *Model evaluation* and *Model selection*, we evaluated and selected machine learning models based on the criteria listed in Section 3.5.2.

Table 3.19 summarizes the results for both scenarios, including the selected scaling algorithms, the selected classifier and regressor models, and their corresponding machine learning metric values obtained during model evaluation. We refer to A.2.2 for a detailed description of the chosen machine learning models.

Exploration Rate and Warm-up Count Finally we selected 5% as the exploration rate (extra random adaptation options selected for verification) and 60 cycles (of 300) as the warm-up count (the number of training cycles to initialize the learning model). For detailed results, see A.2.2.

Runtime Stage Evaluation with the Service-Based System

Hypothesis For the evaluation of the runtime stage activities of ML2ASR+ we use the same hypotheses H1 to H4 as for DeltaIoT, see Section 3.6.1. However, we test hypothesis H2 (the utility penalties when applying ML2ASR+ is not significantly higher compared to DLASeR) and H4 (the reduction of adaptation spaces with ML2ASR+ is not significantly lower compared to DLASeR, nor does ML2ASR+ requires significantly more time for adaptation space reduction) only for scenario 1 as DLASeR does not support setpoint goals yet.

Granularities for Adaptation Space Reduction with an Optimization Goal In both scenarios, ML2ASR+ has to deal with an optimization goal to keep the cost in the application minimal. After filtering out adaptation options based on the predicted satisfaction of threshold and setpoint goals in the system, ML2ASR+ further reduces the adaptation space based on the cost predictions. We evaluate two cases for each scenario: a reduction to at most 1000 adaptation options and a reduction to at most 100 adaptation options. This corresponds to granularity values 1000 and 100.

Quality of the Learning Models Table 3.20 shows the results for the quality of the learning models at runtime. We highlight the most important metrics. The classifier used to make predictions for both threshold goals in scenario 1 (failure rate and response time) has an F1-score of 0.841, and the classifier used to predict the failure rate threshold goal in scenario 2 has an F1-score of 0.935. The regressor used to predict the optimization goal in scenario 1 (cost) has an R2-score of 0.862. For scenario 2, the regressor used to predict the setpoint goal (response time) has an R2-score of 0.902 and the regressor used to predict the optimization goal (cost) has an R2-score of 0.913. These results confirm that the machine learning models can make accurate predictions for the quality properties of the system.

Summary of Results for Quantitative Metrics Table 3.21 summarizes the evaluation results for the quantitative metrics for the Service-Based System. We discuss these results now in detail.

Utility Penalties Figure 3.17 shows the results for the utility penalties for both scenarios when applying ML2ASR+ with a granularity value of 1000 in red and a granularity value of 100 in orange, and DLASeR in blue (only for scenario 1). Subsequently, we zoom in on the threshold goals, setpoint goal, and optimization goals.

Table 3.20: Values of the machine learning metrics for the runtime stage evaluation of the machine learning models of the Service-Based System (abbreviations: Fr → Failure rate, Rt → Response time, C → Cost, S1 → Scenario 1, S2 → Scenario 2).

		F1-score	**Matthews correlation coefficient**
S1	$\mathcal{T}^{\text{Fr}}_{<10\%}, \mathcal{T}^{\text{Rt}}_{<10\%}$	0.841	0.737
S2	$\mathcal{T}^{\text{Fr}}_{<10\%}$	0.935	0.863

		R2-score	**Mean squared error**	**Median absolute error**	**Maximum error**
S1	$\mathcal{O}^{\text{C}}_{min}$	0.862	4.680	1.266	15.059
S2	$\mathcal{S}^{\text{Rt}}_{10ms,0.25ms}$	0.913	2.970	1.123	11.251
	$\mathcal{O}^{\text{C}}_{min}$	0.902	0.039	0.130	1.212

Table 3.21: Values of the metrics for the runtime stage evaluation of requirements of the Service-Based System (abbreviations: Fr → Failure rate, Rt → Response time, C → Cost, S1 → Scenario 1, S2 → Scenario 2, G → Granularity).

		Utility penalties			**AASR**	**Overall time saved**	**Time overhead**
		Fr	***Rt***	***C***			
S1	**G 1000**	0.134%	0.107ms	1.381c	92.59%	92.60%	0.04%
	G 100	0.190%	0.229ms	2.653c	99.26%	99.26%	0.38%
S2	**G 1000**	0.138%	0.001ms	1.589c	93.13%	93.13%	0.05%
	G 100	0.157%	0.003ms	1.764c	99.26%	99.26%	0.48%

Threshold Goals The graphs show us that each approach satisfies the threshold goals for all chosen adaptation options in both scenarios. For ML2ASR+ with granularity values 1000 and 100, we notice utility penalties for the failure rate in the interval $[0.13\%, 0.19\%]$ and utility penalties for the response time in the interval $[0.1ms, 0.23ms]$. With DLASeR we notice (for scenario 1) generally lower utility penalties lower than ML2ASR+: 0.002% for failure rate and $0.04ms$ for response time. Note that a higher utility penalty value for failure rate or response time value here is not necessarily relevant or negative since the threshold goals remain satisfied after adaptation space reduction.

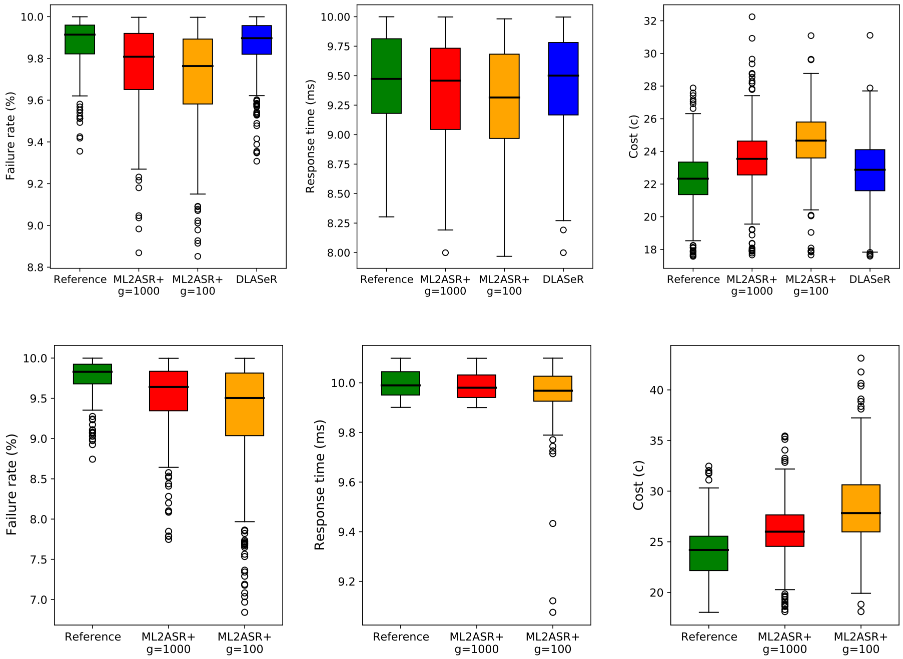


Figure 3.17: Utility penalties for scenario 1 (top) and 2 (bottom) of the Service-Based System compared to the reference approach and DLASeR.

Setpoint Goal For the setpoint goal defined in scenario 2, we similarly notice that none of the chosen adaptation options violate the goal. The utility penalty for the response time of this goal with ML2ASR+ lies in the interval $[0.001ms, 0.003ms]$, showing that the effect of adaptation space reduction is negligible.

Optimization Goals Looking at the cost optimization goal, we see penalties that lay in the interval $[1.38c, 2.66c]$. In scenario 1, we notice a utility penalty of 0.436c for DLASeR, which is slightly better than ML2ASR+. We also observe that the results of ML2ASR+ with a granularity value of 100 are slightly worse compared to a granularity with a value of 1000. This can be explained by the additional restriction put on the reduced adaptation space size: the resulting adaptation space is on average 10 times smaller compared to a granularity value of 1000, leaving fewer adaptation options to be selected from to apply self-adaptation.

Sanity Check with Random Approach We compared ML2ASR+ with batches of 10 runs of an approach that randomly selects adaptation options. For the threshold goals, failure rate and response time in scenario 1, and failure rate only in scenario 2, the random approach manages to always select at least one adaptation option that satisfies the goals. For the optimization goal in both scenarios and the setpoint goal in scenario 2, the Wilcoxon signed rank tests showed statistically significant results between ML2ASR+ and the batches of 10 runs with the approach that randomly selects adaptation options (p-values of $4.88e^{-21}$ for cost in scenario 1, $1.44e^{-4}$ for response time in scenario 2, and $2.55e^{-66}$ for cost in scenario 2). Note that we could not identify statistically relevant differences for the response time in scenario 2 for all individual runs with the approach that randomly selects adaptation options. Figure 3.18 shows the distributions for the optimization and setpoint goals of the IoT network in the two scenarios.[18] The average cost (optimization goal) for scenario 1 is $24.58c$ with ML2ASR+ compared to $26.38c$ with random selection (Random), a difference of $1.8c$ (6.82%). For scenario 2, the values are $25.57c$ with ML2ASR+ compared to $31.51c$ with Random, a difference of $5.94c$ (18.85%). For the average response time (setpoint goal at 10ms±0.25ms), we noticed that the approach that randomly selects adaptation options violated on average the goal in 48 cycles (of a total of 300 cycles) compared to no cycles for ML2ASR+. These results show that ML2ASR+ performs substantially better for more complex adaptation scenarios compared to an approach that randomly selects adaptation options.

Hypotheses H1 (negligible utility penalties compared to reference approach) and H2 (utility penalties not significantly higher compared to DLASeR). The results show that the utility penalties incurred by ML2ASR+ are negligible compared to the reference approach. Specifically, the penalties for cost (optimization goal) are very low in both scenarios (at most $1.589mC$ and $2.653mC$ with granularity values of 1000 and 100, respectively). A smaller granularity value reduces the adaptation space significantly but implies higher utility penalties. The satisfaction of the threshold and setpoint goals remains unaffected with ML2ASR+. The slight increase in cost is acceptable, especially considering that it is not feasible to use the reference approach in practice due to time constraints. In scenario 1, DLASeR shows slightly better results compared to ML2ASR+ with a granularity value of 1000 (with a cost penalty of 1.381c vs 0.436c). However, this cost is acceptable considering that DLASeR does not support all types of adaptation goals yet. In conclusion, we can accept hypotheses H1 (for scenarios 1 and 2) and H2 (for scenario 1) in the Service-Based System application.

[18] Note that similar to the results for DeltaIoT, we cannot generalize the statistical differences we observe between ML2ASR+ and the random approach to other sets of random runs.

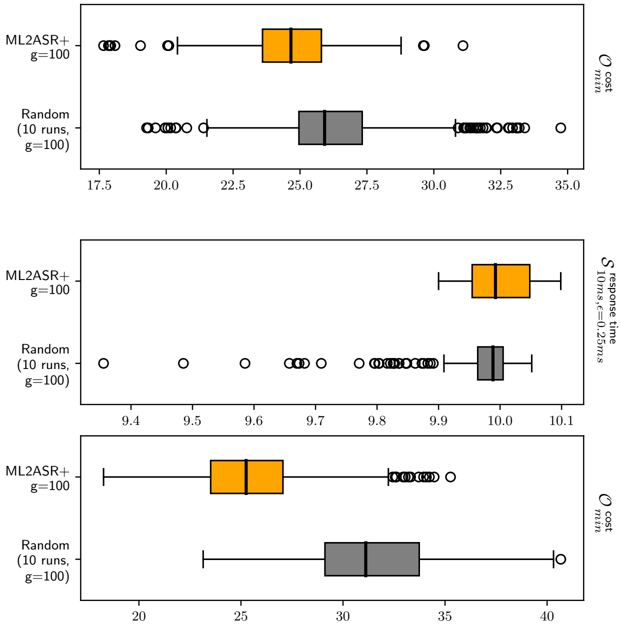


Figure 3.18: Comparison of results for ML2ASR+ and random selection of adaptation options (10 runs) in scenario 1 (optimization goal top graph) and scenario 2 (setpoint goal middle graph, and optimization goal bottom graph) of the Service-Based System.

Average Adaptation Space Reduction Figure 3.19 (left) shows the number of adaptation options remaining after reduction. We used a *warm-up count* of 60 cycles for both ML2ASR+ and DLASeR. The total number of options in these training cycles is limited by the available time for adaptation (30m for the Service-Based System). Note that the reference approach is not subject to this time restriction for the purpose of evaluation; i.e., the reference approach fully analyzes the whole adaptation space with 13500 adaptation options, which is infeasible in practice due to time constraints. During testing, the number of adaptation options with ML2ASR+ is restricted by the granularity value, here 1000 and 100. This results in an Average Adaptation Space Reduction of 92-93% and 99% respectively in both scenarios. For DLASeR in scenario 1, we observe a similar Average Adaptation Space Reduction of 92.66%.

Learning Time Overhead Figure 3.19 (right) shows the learning time overhead introduced by ML2ASR+ with the two evaluated granularity values, and DLASeR (scenario 1). The overhead of ML2ASR+ is very small compared to the overall verification time (the overhead for learning is denoted in ms, while overall verification

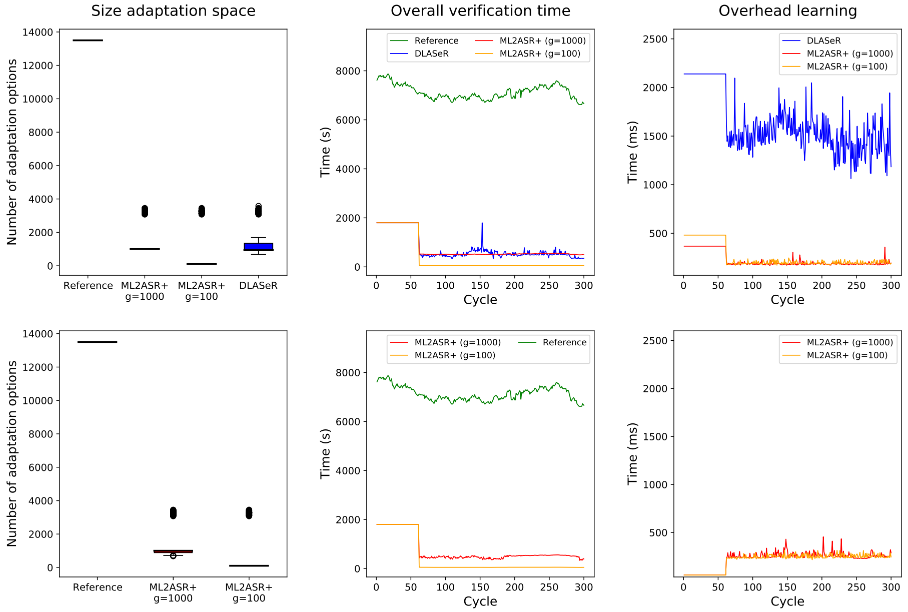


Figure 3.19: Number of verified adaptation options when using ML2ASR+ (left), overall time used (middle), and overhead (right) compared to the reference approach and DLASeR for scenario 1 (top row) and scenario 2 (bottom row) of the Service-Based System.

time is denoted in s). Specifically, the overhead for both granularity values accounts for less than 0.5% of the time required to reduce and verify the adaptation space. In absolute terms, ML2ASR+'s overhead is capped at approximately 500ms during online training cycles. The learning time overhead of DLASeR in scenario 1 is substantially higher compared to ML2ASR+ with a granularity value of 1000 (with close to equal Average Adaptation Space Reduction). Yet, the overhead remains minor compared to the overall verification time; the overhead of DLASeR is 0.30% of the overall verification time. In absolute terms, the overhead of DLASeR is capped at approximately 2200ms during the training cycles as well.

Overall Time Saved Figure 3.19 (middle) shows the overall time used to analyze the (selected) adaptation options. We can clearly see that the overall verification time is significantly reduced, closely aligned to the corresponding Average Adaptation Space Reduction. Concretely, we observe an overall time saved of approximately 92-92% and 99% using ML2ASR+ with granularity values 1000 and 100 respectively. For

DLASeR, we notice an overall time saved of 92.62%, which is in line with ML2ASR+ for a granularity value of 1000 in scenario 1.

Summary of the design stage and runtime stage machine learning activities

Table 3.22 summarizes, similarly to the DeltaIoT case before, the number of inputs, features, objective variables, and metrics for the Service-Based System. Note that no features are removed by *Feature Extraction* since all features were deemed to be relevant (hence the table cells being marked in gray).

Hypotheses H3 (significant reduction of adaptation spaces and time gain) and H4 (adaptation space reduction comparable to DLASeR). The evaluation shows that ML2ASR+ significantly reduces the adaptation space: up to a reduction of 99% depending on the specified granularity value. Paired with this, up to 99% of the time used by the reference approach is saved, closely aligned with the average adaptation space reduction, since the overhead introduced by ML2ASR+ is minimal (constituting less than 0.5%). For scenario 1, DLASeR obtains a similar Average Adaptation Space Reduction (92.66% vs 92.59%) and overall time saved (92.62% vs 92.60%) compared to ML2ASR+ with granularity value 1000. ML2ASR+ with granularity value 1000 outperforms DLASeR on learning overhead with a value of 0.04% vs 0.30%. As such, we can accept hypothesis H3 for scenarios 1 and 2, and H4 for scenario 1 in the Service-Based System.

3.7 Discussion

In Section 3.2.4, we described the research question targeted in this work and we listed the desirable requirements for an approach to tackle the research question. To that end, we proposed ML2ASR+. In the evaluation, we assessed the quantitative requirements. We now discuss the remaining qualitative requirements, answer the research question, highlight insights obtained from this research endeavor, and conclude with a discussion of threats to validity.

Table 3.22: The number of inputs, features, objective variables, and metrics for the machine learning pipeline of scenario 2 of the Service-Based System in the design and runtime stage. The prediction column is marked in red to indicate that it is not being used yet in the training cycles. The number of inputs for online learning is determined by the number of options that could be verified by the Verifier. Abbreviations: “f vectors” → feature vectors, “q vectors” → quality vectors, “Pl” → packet loss, “Ec” → energy consumption, “La” → latency.

	Design stage		*Runtime stage*			
	Feature Extraction	**Machine Learning Model Identification**	**Feature Extraction**	**Prediction**	**Verification**	**Online Learning**
Number of inputs	$100 * 13500$ f vectors $100 * 13500$ q vectors	$100 * 13500$ f vectors $100 * 13500$ q vectors	13500 f vectors	13500 f vectors	13500 f vectors	X f vectors X q vectors
Number of features	22 features	22 features	22 features	22 features	x	22 features
Objective variables	1. Fr class 2. Rt value 3. C value	1. Fr class 2. Rt value 3. C value	x	1. Fr class 2. Rt value 3. C value	x	1. Fr class 2. Rt value 3. C value
Metrics	x	1. F1, MCC 2. R2, MSE, MAE, ME 3. R2, MSE, MAE, ME	x	x	x	x

3.7.1 Qualitative Requirements

Reusability With reusability, we refer to the ability of ML2ASR+ to be instantiated and applied over multiple application domains. To demonstrate that we have covered this requirement, we demonstrated the applicability of ML2ASR+ to the Internet of Things domain and the Service-Based Systems domain. In both applications, we analyzed the performance of ML2ASR+ in two evaluation scenarios, while also assessing different granularity values. From the results, we can conclude that ML2ASR+ has the ability to handle both applications and the different evaluation scenarios.

Automatic Operation at Runtime To evaluate the second requirement, we make a distinction between the design stage and the runtime stage of the approach. In the design stage, the system requires manual input from the system developer(s) to properly configure the *Machine Learning Module*, as described in section 3.4.3. Hence this step is not completely automated. Once the *Machine Learning Module* is deployed, no further input or intervention is necessary from the system operators. This is also demonstrated in the evaluation: during operation ML2ASR+ reduces the adaptation space without any input from an operator or system developer. ML2ASR+ thus satisfies this requirement.

Modularity Adaptation Goals To evaluate the ability of ML2ASR+ to deal with different combinations of adaptation goals, we specifically investigate whether ML2ASR+ is able to deal with threshold, setpoint, and optimization goals. In our evaluation, we defined four scenarios that combine different types of goal types for two different applications. This way, we ensured that different combinations of the three types of adaptation goal types are assessed. We conclude that ML2ASR+ supports dealing with all the goal types in the evaluated application scenarios.

Granularity of Adaptation Space Reduction With granularity we refer to the degree to which ML2ASR+ is able to reduce adaptation spaces, i.e., selecting a specified number or percentage of adaptation options from the original adaptation space. ML2ASR+ allows the specification of a granularity value that constrains the size of the reduced adaptation space. We have demonstrated this for both applications in different evaluation scenarios with granularity values of 10, 25, 100, and 1000. We can thus conclude that ML2ASR+ satisfies this last requirement as well.

Answer to research question "*How can machine learning be used to reduce large adaptation spaces of self-adaptive systems with different types of adaptation goals to perform more efficient analysis without compromising the goals*?" This work demonstrates how classic supervised machine learning techniques can be used to reduce the adaptation space to a more manageable subset. After designing the Machine Learning Module, ML2ASR+ initializes the learning models, trains the models during warm-up, and then uses the models to make predictions about the satisfaction of adaptation goals for individual adaptation options. ML2ASR+ uses classification to predict the satisfaction of a threshold or setpoint goals, and regression to predict the quality value associated with an adaptation option. ML2ASR+ provides the means to reduce the adaptation space with the specified granularity value; this flexibility enables the approach to adjust with the available time window to perform adaptation. Empirical evaluation shows that ML2ASR+ drastically reduces the time required for analysis with a negligible effect on the satisfaction of the adaptation goals in our evaluated systems.

3.7.2 Insights

We share several insights we obtained during the design and evaluation of ML2ASR+:

- When handling multiple adaptation goals, there is a risk that errors in learning models propagate further with each prediction. Accumulating prediction errors may ultimately reduce the efficacy of the approach.
- The overhead introduced by the learning approach directly links with the selected granularity value. Even for small granularity values, the gain in time required for analysis is significant. Yet, such a setting may also have a significant impact on utility penalties. Hence, the right choice for setting the granularity value is important and requires experimentation.
- It is important to highlight that the use of linear machine learning models is not a “one size fits all” solution. The effectiveness of the approach depends on the underlying relation between input data (features) and output (qualities). If this relation cannot be properly modeled linearly, other approaches such as DLASeR that rely on deep neural networks may be preferable as these approaches capture these intrinsic relations better. It is however important to keep in mind that other approaches may follow different workflows and carry their own drawbacks. For instance, DLASeR follows different steps in its design stage and runtime stage workflows and the approach introduces a larger learning overhead compared to ML2ASR+.

- ML2ASR+ relies on the assumption that the (formal) models that are used to estimate quality properties of the underlying system provide reliable and correct results. A quality model that cannot handle concept drift or evolution of the system may yield data that does not capture the real system accurately. This can affect the performance of the machine learning models. Yet, extracting data directly from the real system rather than the (formal) model is not a solution as this data is inherently limited since only a single adaptation option can be applied each cycle. However, exploiting the data retrieved from the real system to detect issues with the model and adapt or evolve the models dynamically is an option; we leave this as future work.
- For the validation of ML2ASR+, we trained the learning models both during the design stage and the runtime stage. In theory, it would be possible to generate all the data and train machine learning models entirely during the design stage. However, due to the exponential growth of the adaptation space when considering all possible uncertainty values in combination with the available adaptation options in the system, generating all the data and training machine learning models would be infeasible in practice. Therefore, we used a representative sample of the data to apply design stage training and then collected additional data after startup to continue the training according to the actual system configuration and uncertainties at runtime.
- In this work, we considered setpoint goals based on a small window ϵ, resembling similarities with steady-state error in control-based approaches [314]. Any configuration within this window complies with the goal. An interesting option for future work is to refine this view and consider the option closest to the setpoint as the optimal one. Combined with an optimization goal this will lead to a multi-objective optimization problem.

3.7.3 Threats to Validity

The empirical evaluation of ML2ASR+ is subject to threats to validity. For each threat, we discuss potential critiques of this study and we explain how we dealt with those.

Internal Validity To make sure that we can draw a causal conclusion based on the study, we took several measures. Concerning the contribution, we specified the approach formally, providing a basis to define that the approach works as described. Concerning the evaluation, we have applied the same settings of the simulator with the same settings for the application parameters when comparing ML2ASR+ with the other approaches. This is particularly relevant in settings with stochastic behavior. As such we provide a basis for deriving the conclusions of comparing the approaches. We also provide a replication package [288] for other researchers to validate the results.

External Validity External validity concerns the generalization of the results beyond the scope of the study. This study contributes an architectural approach for adaptation space reduction in self-adaptive systems that is centered on the *Machine Learning Module* with a corresponding workflow. This approach uses classical supervised machine learning techniques to support the adaptation process. Since we have applied and evaluated ML2ASR+ to a limited set of scenarios with particular characteristics and types of uncertainties, we cannot make general claims about the efficacy of the approach in other settings. To mitigate this threat to some extent, we have evaluated the approach in two distinct domains with different challenges regarding adaptation space reduction for different combinations of adaptation goals.

Construct Validity With construct validity we analyze whether we have obtained the right measures to answer the proposed research question. To minimize threats to construct validity we provided an explicit definition of six requirements to be evaluated. For several of these requirements, we defined concrete metrics that enabled us to evaluate the performance of the approach empirically (in terms of efficiency and overhead). Several of these metrics are based on established practice for the evaluation of learning approaches. In addition, the formal specification of ML2ASR+ provides a rigorous description of how the approach works. Nevertheless, we acknowledge that other metrics may have been considered for evaluating the appropriateness of adaptation space reduction.

Conclusion Validity Threats to conclusion validity concern reaching an incorrect conclusion about a relationship in the observations. To mitigate conclusion validity threats, we applied ML2ASR+ in different scenarios of different domains with different characteristics. Based on a set of well-defined metrics, the results confirm the observation that ML2ASR+ is effective for adaptation space reduction in self-adaptive systems. In addition, we have made all code and experimental data publicly available [288] to reproduce the experiments in order to confirm the findings.

3.8 Conclusion

In this chapter we presented ML2ASR+, a novel approach to analyze large adaptation spaces more effectively by exploiting classic supervised machine learning techniques to reduce adaptation spaces on the fly. ML2ASR+ extends the basic MAPE-K architecture with a *Machine Learning Module* that supports the *Analyzer* component by reducing the adaptation space to a manageable subset. In particular, the *Machine Learning Module* filters adaptation options that are predicted to not meet the adaptation goals in the system. We have demonstrated the effectiveness and viability of ML2ASR+ in our evaluation in two different application domains. We evaluated the effectiveness of ML2ASR+ in reducing the adaptation space as well as the overhead introduced by the approach. The results showed that the overhead introduced by ML2ASR+ is minimal compared to the time required to verify the remaining subset of filtered adaptation options. On top of this, the penalty in system qualities is negligible when choosing a new system configuration from the reduced adaptation space. In future work, we plan to investigate adaptation space reduction in decentralized self-adaptive systems where multiple feedback loops need to coordinate the analysis. In the long term, we plan to expand our study on the use of machine learning and self-adaptive systems, and investigate how evolutionary learning can be used to support self-adaptation in systems that are exposed to unanticipated changes, requiring system evolution. First ideas in this direction are reported in [337].

Chapter 4

A/B Testing: A Systematic Literature Review

Publication details. This chapter is based entirely on a journal article that is under submission [286].

Personal contributions. Conceptualization (80%), Methodology (80%), Data collection (70%), Validation (80%), Formal analysis and interpretation results (90%), Writing (90%), Visualization (100%).

Positioning. A/B testing is widely used in practice to enable data-driven decision-making for software development. While a few studies have explored different facets of research on A/B testing, no comprehensive study has been conducted on the state-of-the-art in A/B testing. To address this gap and provide an overview of the state-of-the-art in A/B testing, this chapter reports the results of a systematic literature review that analyzed 141 primary studies. The research questions focused on the subject of A/B testing, how A/B tests are designed and executed, what roles stakeholders have in this process, and the open challenges in the area. Analysis shows that the main targets of A/B testing are algorithms, visual elements, and workflow and processes. Stakeholders have three main roles in the design of A/B tests: concept designer, experiment architect, and setup technician. Stakeholders have two main roles during A/B test execution: experiment coordinator and experiment assessor. The main reported open problems are related to the enhancement of proposed approaches and their usability. From our study we derived three interesting lines for future research: strengthening the adoption of statistical methods in A/B testing, improving the process of A/B testing, and enhancing the automation of A/B testing.

4.1 Introduction

Iterative software development and time to market are crucial to the success of software companies. Central to this is innovation by exploring new software features or experimenting with software changes. In order to enable such innovation in practice, software companies often employ A/B testing [203, 118, 229, 156]. A/B testing, also referred to as online controlled experimentation or continuous experimentation, is a form of hypothesis testing where two variants of a piece of software are evaluated in the field (ranging from variants with a slightly altered GUI layout to variants of software with new features). In particular, the merit of the two variants is analyzed using metrics such as click rates of visitors of websites, members' lifetime values (LTV) in a subscription service, and user conversions in marketing [187, 358, 100]. A/B testing is extensively used in practice, including large and popular tech companies such as Google, Meta, LinkedIn, and Microsoft [342, 358, 227, 387].

Even though A/B testing is commonly used in practice, to the best of our knowledge, no comprehensive empirically grounded study has been conducted on the state-of-the-art (i.e., state-of-research in contrast to state-of-the-practice) in A/B testing. Such a study is crucial to provide a systematic overview of the field of A/B testing to drive future research forward. Three earlier studies [17, 16, 297] explored a number of aspects of research on A/B testing, such as research topics, type of experiments in A/B testing, and A/B tooling and metrics. Yet, these studies do not provide a comprehensive overview of the state-of-the-art that provides deeper insights in the types of targets to which A/B testing is applied, the roles of stakeholders in the design of A/B tests, the execution of the tests, and the usage of the test results. These insights are key to positioning and understanding A/B testing in the broader picture of software engineering. To tackle this issue, we performed a systematic literature review [192]. Our study aims to provide insights into the state of research in A/B testing as a basis to guide future research. Practitioners may also benefit from the study to identify potential improvements in A/B testing in their daily practices.

The remainder of this chapter is structured as follows. Section 4.2 provides a brief introduction to A/B testing and discusses related secondary studies. In Section 4.3, we outline the research questions and summarize the methodology we used. Section 4.4 then presents the results, providing an answer to each research question. In Section 4.5, we reflect on the results of the study, report insights, outline opportunities for future research, and outline threats to validity. Finally, Section 4.6 concludes the chapter.

4.2 Background and related work

4.2.1 Background

A/B testing is a method where two software variants, denoted as variant A and variant B, are compared by evaluating the merit of the variants through exposure to the end-users of the system [322]. To compare the variants, a hypothesis is formulated together with an experiment to test it, i.e., the actual A/B test. As opposed to regular software testing, A/B testing takes place in live systems. Figure 4.1 shows the general process of A/B testing with three main phases.

The first phase of A/B testing concerns the design of an A/B test. In this experiment design, a range of parameters is specified, such as: the hypothesis, the sample of the population the experiment should be targeted to, the duration of the experiment, and the A/B metrics that are collected during the experiment. The A/B metrics are used to determine the merit of each variant during the experiment. Examples of A/B metrics include the click-through rate (CTR), number of clicks, and number of sessions [99].

The second phase of A/B testing consists of the execution of the A/B test in the running software system. Both variants are deployed in a live system, and the sample of the population is split among both variants. During the execution, the system keeps track of relevant data to evaluate the experiment after it finishes (according to the specified duration). Relevant data may directly correspond to the specified A/B metrics, or it may indirectly enable advanced analysis in the evaluation stage to gain additional insights from the conducted A/B tests.

The third phase of A/B testing comprises the evaluation of the experiment. After the A/B test is finished, the original hypothesis is tested, typically with a statistical test, such as a students test or Welsh's t-test [163, 350]. Based on the outcome of the test, the designer can then take follow-up actions, for instance initiating a rollout of a feature to the entire population or designing new A/B variants to test in subsequent A/B tests.

Controlled experiments vs A/B testing

Traditionally, a controlled experiment is an empirical method that enables systematical testing of a hypothesis [72]. Two types of variables are distinguished in controlled experiments: independent and dependent variables. Independent variables are variables that are controlled during the experiment to test the hypothesis, for instance, a state-of-the-art and a newly proposed approach to solve a particular design problem by a control group and a treatment group respectively. Dependent variables are variables that are measured during the experiment to compare the results of both the control and treatment group, for instance, the fault density and productivity obtained in a design

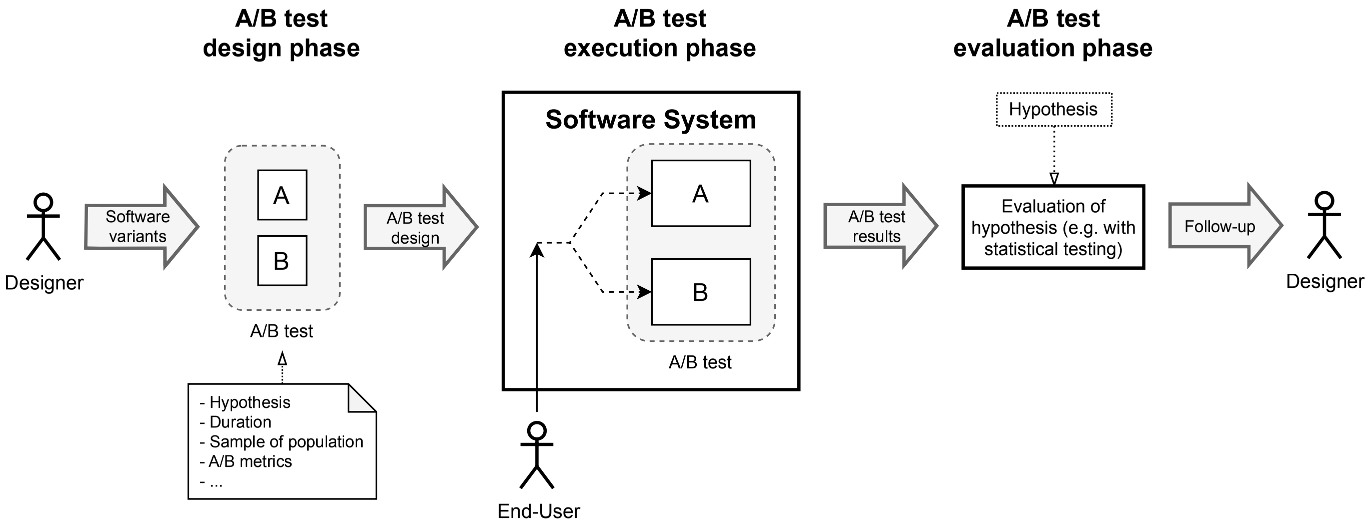


Figure 4.1: General A/B testing process.

task. After conducting the experiment, the hypothesis is tested and conclusions are drawn based on the results; for instance, a newly proposed design approach has a significantly lower fault density compared to the state-of-the-art approach, but more research is required concerning productivity. Controlled experimentation is widely used across all types of scientific fields, such as psychology [70], pharmaceutics [244], education [72], and nowadays also in software engineering [318, 78, 139].

Whereas controlled experiments are typically performed *offline in a controlled setting*, A/B testing uses controlled experiments to evaluate software features or variants on the *end-users of a running system*. For this reason, A/B testing is often referred to as online controlled experimentation [206, 115]. The aim of A/B testing lies in testing hypotheses in live software systems where end-users of these systems form the participants or population of the experiment. Examples of hypotheses that are tested in A/B testing often relate to improving user experience (UX) [290], improving user interface (UI) design [354], improving user click rates [4], or evaluating non-functional requirements in distributed services [19].

DevOps and A/B testing

Development Operations (DevOps in short) has gained popularity in recent years [125]. DevOps consists of a set of practices, tools, and guidelines to efficiently and effectively manage and carry out different tasks during software life cycles. This ranges from the process of software development to the deployment and management of software at runtime. Automation of software processes plays a central part in DevOps to make life easier for developers and ease the burden of software development in general.

Common practices that are part of the DevOps lexicon are continuous integration and continuous deployment (CICD in short) [172]. CICD consists of the automation of software testing, software integration and building, and deployment of software, effectively reducing manual labor required by developers and easing the burden of deploying software. In a similar vein, continuous experimentation [392] aims at continuously setting up experiments in software systems to test new software variants. Put differently, continuous experimentation enriches the software development process by enabling a data-driven development approach (e.g., by measuring user satisfaction of new software features early on in development). To achieve this, A/B testing is used to set up and evaluate online controlled experiments in the software system. Fabijan et al. [118] for example perform a case study on the evolution of scaling up continuous experimentation at Microsoft, providing guidelines for other companies to conduct continuous experimentation.

4.2.2 Related secondary studies

We start with a summary of secondary studies related to the study presented in this chapter. Then we pinpoint the aim of the study presented in this chapter to provide a systematic overview of the state-of-the-art in A/B testing.

Summary of related reviews.

We grouped related studies into three classes: studies with a focus on technical aspects of A/B testing, studies focusing on social aspects of A/B testing, and studies concerned with A/B testing in specific domains.

Technical aspects of A/B testing Rodriguez et al. [296] performed a systematic mapping study on continuous deployment of software-intensive services and products. The authors identify continuous and rapid experimentation as one of the factors that characterize continuous deployment, and elaborate on this through the lens of the deployment of these experiments and DevOps practices associated with it. Ros and Runeson [297] put forward a mapping study on continuous experimentation and A/B testing. The authors explore research topics, organizations that employ A/B testing, and take a deeper look at the type of experimentation that is conducted. Auer and Felderer [16] conducted a systematic mapping study on continuous experimentation. The authors put a focus on the research topics, contributions, and research types, collaboration between industry and academia, trends in publications, popularity in publications on A/B testing, venues, and paper citations. Recently, Auer et al. [17] presented a systematic literature review on A/B testing and continuous experimentation, leveraging the results from previous mapping studies [297, 16]. The authors apply forward snowballing on a set of papers to compose the list of primary studies for the review. They then explore the core constituents of a continuous experimentation framework and the challenges and benefits of continuous experimentation. Closely related, Erthal et al. [110] conducted a literature review by applying an ad-hoc search, followed by snowballing on the initial set of identified papers. The study places emphasis on defining continuous experimentation and exploring its associated processes. While the authors acknowledge A/B testing as one of the strategies for achieving continuous experimentation, this literature review does not delve into the technical aspects of A/B testing.

Social aspects of A/B testing An important social aspect of A/B testing is obtaining user feedback. A significant portion of A/B tests revolves around prioritizing and optimizing the user experience. We identified two studies that focus on this social aspect. Fabijan et al. [120] present a literature review on customer feedback and data collection techniques in the context of software research and development. The authors highlight existing techniques in the literature to obtain customer feedback and organize data collection, in which software development stages the techniques are used, and what the main challenges and limitations are for the techniques. One of the techniques outlined by the authors is A/B testing, which can serve as a valuable tool to obtain user feedback on prototypes. Fabijan et al. [121] discuss the challenges and implications of the lack of sharing customer data within large organizations. One specific case presented by the authors underpins critical issues that manifest from not sharing qualitative customer feedback in the pre-development stage with the development stage, forcing developers to repeat the collection of user feedback or to develop products without this information.

A/B testing in specific domains Beyond A/B testing at Internet-based companies, the use of A/B testing is reported in various other domains. An example is the domain of embedded systems. Mattos et al. [246] explore challenges and strategies for continuous experimentation in embedded systems, providing both industrial- and research perspectives. Another domain is Cyber-Physical Systems (CPS). Giaimo et al. [154] present a systematic literature review on the state-of-the-art of continuous experimentation in CPS, concluding that the literature focuses more on presented challenges rather than proposing solutions to the challenges.

Summary Existing secondary studies examined A/B testing with a focus on realizing tests, associated processes, and the types of experimentation conducted. However, these studies have a particular focus, or they lack a rigorous search process to identify relevant studies. Existing studies fall short in providing insights in the *target* of A/B testing (i.e., "what" is the subject of testing), the *roles of stakeholders* in designing and executing A/B tests, and the *utilization* of A/B test results.

Aim of the study.

To tackle the limitations of existing studies, we performed an in-depth literature study. We define the aim of this study using the Goal Question Metric (GQM) approach [23]:

Purpose: Study and analyze

Issue: The design and execution of A/B testing

Object: In software systems

Viewpoint: From the view point of researchers.

Concretely, we aim to investigate the subject of A/B testing, how A/B tests are designed and executed, and what the role is of stakeholders in the different phases of A/B testing. Finally, we also aim to obtain insights into the research problems reported in the literature.

4.3 Methodology

This study uses the methodology of a systematic literature review as described in [192]. This methodology describes a rigorous process to review the literature for a topic of interest. The process ensures that the review identifies, evaluates, and interprets all relevant research papers in a reproducible manner. The literature review consists of three main phases: planning, execution, and synthesis. During planning a protocol is defined for the study [285], which includes the motivation for the study, the research questions to be answered, sources to search for papers, the search string, inclusion- and exclusion criteria, data items to be extracted from the primary studies[1], and analysis methods to be used. During execution the search string is applied as specified in the protocol, the inclusion and exclusion criteria are applied to identify the primary studies, and all the data items are extracted from these papers. Lastly, during synthesis the extracted data is analyzed and interpreted to answer the research questions and to obtain useful insights from the study.

We conducted the systematic literature review with four researchers. Further details on the process of the literature review (e.g. the roles the researchers play in the literature review) are summarized in the following sections. A complete description of the protocol, with all collected data and the data analysis, is available at the study website [285].

4.3.1 Research questions

To realize the aim of this study ("Study and analyze the design and execution of A/B testing in software systems from the viewpoint of researchers."), we put forward four research questions:

RQ1: What is the subject of A/B testing?

RQ2: How are A/B tests designed? What is the role of stakeholders in this process?

[1] We use the term “research paper” to refer to papers that we considered for the application of inclusion and exclusion criteria in the SLR, and the term “primary study” for the research papers that we selected for data extraction.

RQ3: How are A/B tests executed and evaluated in the system? What is the role of stakeholders in this process?

RQ4: What are the reported open research problems in the field of A/B testing?

With RQ1, we investigate the subject of A/B testing, i.e., the (part of the) system to which an A/B test is applied. Examples include A/B tests on program variables, application features, software components, subsystems, the system itself, and the infrastructure used by the system. We also investigate the domains in which A/B testing is used.

With RQ2, we investigate what is defined and specified in A/B tests before they are executed in the system. We look at the metrics used, and whether statistical methods are used in the experiments and if so which methods. We also investigate which stakeholders are involved in this process and what is their role (e.g., users of the system influencing the tests that should be deployed, or architects deciding on which population the A/B tests should be run).

With RQ3, we investigate how A/B tests are executed in the system, and the results are evaluated. More specifically, we look at the way in which data is collected for evaluation in the test, the evaluation of the A/B test itself (using the collected data and, if applicable, the result of a statistical test), and the use of the test results (e.g., decision about selection of target, input for maintenance, trigger for next test in a pipeline). We also explore the role stakeholders have during this process of A/B testing (e.g., operators deciding when to finish an experiment).

With RQ4, we identify open research problems in the field of A/B testing. The problems can be derived from descriptions of limitations of proposed approaches in the reviewed papers, open challenges, or outlines of future work on A/B testing.

4.3.2 Search query

We first identified a list of relevant terms for A/B testing from a number of known publications [197, 160, 205, 203, 195, 99]. We then identified and applied a gold standard [400] to tune the terms. For a detailed description of the relevant terms and application of the gold standard, we refer to the research protocol [285]. Figure 4.2 (top) displays the final search query after applying the gold standard.

4.3.3 Search strategy

The search query was executed in October 2022. The search query was applied to the title and abstract of each paper in the sources (not case-sensitive). The automatic search

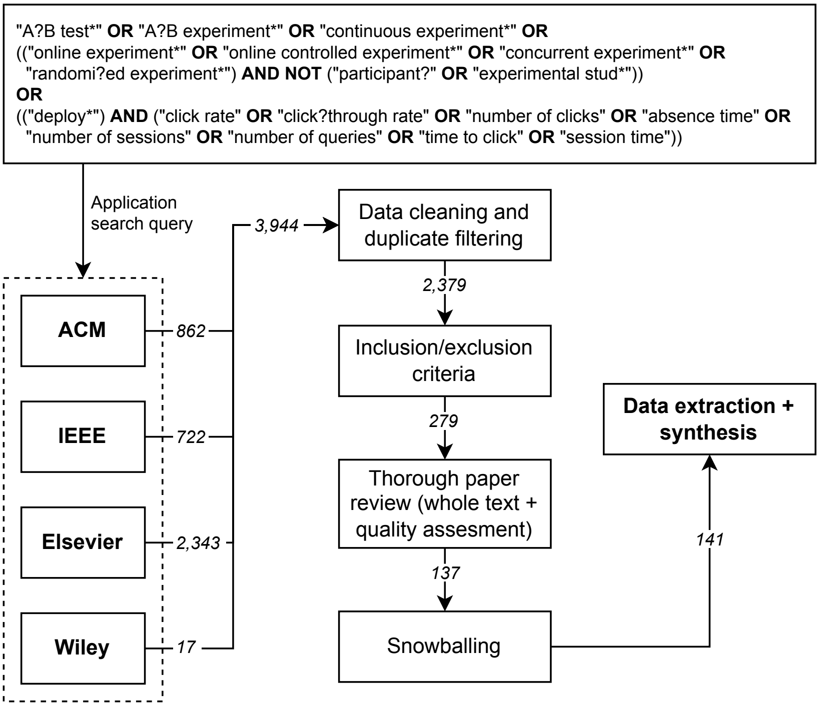

"A?B test*" OR "A?B experiment*" OR "continuous experiment*" OR
(("online experiment*" OR "online controlled experiment*" OR "concurrent experiment*" OR "randomi?ed experiment*") AND NOT ("participant?" OR "experimental stud*"))
OR
(("deploy*") AND ("click rate" OR "click?through rate" OR "number of clicks" OR "absence time" OR "number of sessions" OR "number of queries" OR "time to click" OR "session time"))
Application search query
ACM
862
IEEE
722
Elsevier
2,343
Wiley
17
3,944
Data cleaning and duplicate filtering
2,379
Inclusion/exclusion criteria
279
Thorough paper review (whole text + quality assesment)
137
Snowballing
141
Data extraction + synthesis

Figure 4.2: Primary studies selected for the systematic literature review.

resulted in $3,944$ papers, as shown in Figure 4.2. After filtering duplicate papers and selecting only journal versions of extensions of conference versions, $2,379$ research papers are left for further processing.

4.3.4 Search process

After collecting the papers, we applied the following inclusion criteria:

IC1: Papers that either (1) have a primary focus on A/B testing (or any of its known synonyms) or (2) describe and apply (new) design(s) of A/B tests; for example introducing a proof-of-concept;

IC2: Papers that include an assessment of the presented A/B tests, either by providing an evaluation through simulation with artificial data or field data, or through running one or more field experiments in a real system;

IC3: Papers written in English.

We defined IC1 such that we only include works that are relevant to the posed research questions, i.e., it is essential that the work focuses on A/B testing or their design and evaluation. Note that IC1 includes papers that address and present solutions to known challenges in A/B testing. IC2 ensured that only papers are included that contain data related to the design and/or running of A/B tests. Lastly, we only included papers that are written in English with IC3.

Besides the inclusion criteria above, we also applied the following exclusion criteria:

EC1: Papers that report (systematic) literature reviews, surveys (using questionnaires), interviews, and roadmap papers;

EC2: Short papers (≤ 4 pages)[2], demos, extended abstracts, keynote talks, and tutorials;

EC3: Papers with a quality score ≤ 4 (explained in Section 4.3.5).

EC4: Papers that provide no or only a very brief description of the A/B testing design process or execution process.

EC1, EC2, and EC3 excluded papers that do not directly contribute new technical advancements, preliminary works that have not been fully developed yet, or works that are not of sufficient quality. In this literature review, we focus on mature, state-of-the-art research in the field of A/B testing to answer the research questions. EC4 excluded works that do not contain essential information to answer the research questions.

Papers that satisfied all inclusion criteria and none of the exclusion criteria were included as primary studies in the literature study. The application of inclusion and exclusion criteria to the titles and abstracts of the research papers resulted in 279 papers. A thorough reading of the papers further reduced the number of papers to 137. In addition to the research papers retrieved via the search string and filtered by applying inclusion/exclusion criteria, we applied snowballing on the cited works of these papers to capture potentially missed papers. With snowballing we discovered 4 additional papers, bringing the final number of primary studies to 141, as shown in Figure 4.2.

4.3.5 Data items

To be able to answer the research questions, we extract the data items listed in Table 4.1. For each data item, we provide a detailed description.

D1-4: Authors, year, title, and venue used for documentation purposes.

[2] Papers published in the *Lecture Notes in Computer Science* format with < 8 pages are also considered short.

Table 4.1: Collected data items to answer the research questions

Identifier	Data item	Purpose
D1	Authors	Documentation
D2	Year	Documentation
D3	Title	Documentation
D4	Venue	Documentation
D5	Paper type	Documentation
D6	Authors sector	Documentation
D7	Quality score	Documentation
D8	Application domain	RQ1
D9	A/B target	RQ1
D10	A/B test type	RQ2
D11	Used metrics	RQ2
D12	Statistical methods employed	RQ2
D13	Role of stakeholders in the experiment design	RQ2
D14	Additional data collected	RQ3
D15	Evaluation method	RQ3
D16	Use of test results	RQ3
D17	Role of stakeholder in experiment execution	RQ3
D18	Open problems	RQ4

D5: The type of paper. Options include: focus paper (focus on A/B testing itself, i.e., modifications, suggestions, or enhancements to the A/B testing process), or applied paper (application and evaluation of A/B testing in real software systems).

D6: The sector of the authors of the primary study used for documentation (based on the author's affiliation). Options include Fully academic, Fully industrial, and Mixed.[3]

D7: A quality score for the reporting of the research [242]. The quality score is defined on the following items: *Problem definition of the study*, *Problem context (relation to other work)*, *Research design (study organization)*, *Contributions and study results*, *Derived insights*, *Limitations*. Each item is rated on a scale of three levels: explicit description (2 points), general description (1 point), or no description (0 points). Therefore, the quality score is defined on a scale of 0 to 12 [239].

D8: The application domain that is used in relation to A/B testing in the primary study. Initial options include E-commerce, Telecom, Automotive, Finance, and Robotics. Further options were derived during data collection.

[3] Academic refers to affiliations that are eligible to graduate master and/or PhD students.

D9: The target of A/B tests describes the element that is subject of A/B testing. Initial options include an algorithm, a user interface, and application configurations. Further options were derived during data collection.

D10: The type of A/B test corresponds to the number of A/B variants and the way in which they are tested. Initial options include Single (classic) A/B test, Single multivariate A/B test, Manual sequence of classic A/B tests, Manual sequence of multivariate A/B tests, Automated sequence of classic A/B tests, Automated sequence of multivariate A/B tests. Additional options were derived during data collection.

D11: The metrics that are used in the A/B tests. Initial options include Click rate, Click-through rate, Number of clicks, Number of sessions, Number of queries, Absence time, Time to click, and Session time. Additional options were derived during data collection.

D12: The statistical method that is employed to evaluate the data obtained through the A/B test, if any. Initial options include a Student test, a Proportional test, and No statistical test. Further options were derived during data collection.

D13: The role of stakeholders in the experiment design. Initial options include Determining A/B test goal/hypotheses, Determining A/B test duration, and Tuning A/B test variants. Further options are derived during data collection.

D14: Additional data that is collected during the execution of an A/B test (in addition to direct or indirect A/B metric data). Examples include User geo-location, Browser type, and Timestamps of invocations or requests. Further options are derived during data collection.

D15: The evaluation method used in the primary study[4]. Initial options include an Illustrative example, Simulation, and Empirical evaluation.

D16: The use of the test results gathered from A/B tests. Examples include Subsequent A/B test execution, Subsequent A/B test design, Feature rollout, and Feature development. Further options are derived during data collection.

D17: The role of stakeholders in the process of executing A/B tests. Initial options include A/B test alteration (adjusting individual A/B tests), A/B test triggering (starting subsequent A/B tests manually), A/B test supervision (monitoring A/B tests execution), No involvement, Unspecified. Further options are derived during data collection.

D18: Reported open problems. Open problems are derived from the reported challenges, limitations, and threats to validity. Options are derived during data collection.

[4] We distinguish data retrieved from empirical evaluation in a live system from data retrieved from simulation or an illustrative example to provide targeted insights into the execution of A/B tests during data analysis of the SLR.

Table 4.2: Paper types of the primary studies.

Type	Number of occurrences
Focus	90
Applied	51

Table 4.3: Author backgrounds of the primary studies.

Background	Number of occurrences
Academic	26
Industry	72
Mixed	43

4.4 Results

We start with demographic information about the primary studies. Then we zoom in on each of the research questions.

4.4.1 Demographic information

Demographic information is extracted from data items Paper type (**D5**), Authors sector (**D6**), and Quality score (**D7**).

Of the 141 primary studies, 90 (63.8%) have a focus on A/B testing itself, while 51 (36.2%) apply A/B testing or use it for evaluation purposes, see Table 4.2.

A majority of 72 primary studies (51.1%) have industrial authors, see Table 4.3. Forty-three studies (30.5%) have a mix of industry and academic authors, and 26 studies (18.4%) are from academic authors only.

Figure 4.3 shows the distribution of quality scores with an average of 8.81 [±1.58]. This shows that the reporting of the research in the primary studies is of good quality. Since all papers passed the threshold of 4, none of the papers had to be excluded for the extraction of data to answer the research questions.

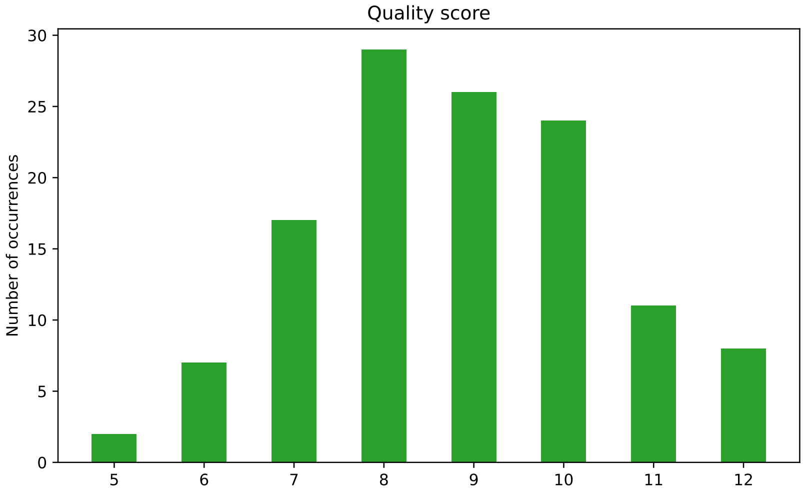


Figure 4.3: Quality scores of the primary studies.

4.4.2 RQ1: What is the subject of A/B testing?

To answer this research question, we look at the following data items: Application domain (**D8**), and A/B target (**D9**).

Application domain Table 4.4 lists the application domains of the primary studies. The average number of domains is 1.13 (131 primary studies applied A/B testing in one domain, three studies in two domains, six studies in three domains, and one study in four domains). Nine studies do not mention any domain. We observe that the most popular application domain is the Web (38 occurrences). Typical examples are social media platforms, such as Facebook [232] or LinkedIn [390], news publishers [396, 119], and multimedia services, such as movie streaming at Netflix [10]. The second most popular domain is search engines (35 occurrences), with studies conducted at Yandex [98, 97], Bing [91, 238], Yahoo [7, 334], among others. A/B testing is also actively applied in E-commerce (27 occurrences), with examples from retail giant Amazon [109], the fashion industry [57], and C2C (consumer-to-consumer) businesses, such as Etsy [188] and Facebook marketplace [342]. Next we observe the application of A/B testing in what we group under ”interaction” (22 occurrences), with digital communication software, such as Snap [386] and Skype [119], user-operating system interaction [162, 116], and application software, such as an App store [76] and mobile games [393]. Lastly, we note the financial application domain (16 occurrences), including studies at Yahoo finance [403] and Alipay [43], transportation (4 occurrences) at for instance

Table 4.4: Identified application domains for A/B testing.

Application domain	Number of occurrences
Web	38
Search engine	35
E-commerce	27
Interaction	22
Finances	16
Transportation	4
Other	8
N/A	9

Didi Chuxing [137]. Other domains are education (3 occurrences) [293] and robotics (2 occurrences) [247], among others.

A/B target The target of the A/B test denotes the element that is subject to testing and of which (at least) two variants are compared. Table 4.5 lists the A/B targets we identified from the primary studies, with a description and examples for each. The average number of A/B targets is 1.21 (120 primary studies applied A/B testing to one element, 26 studies to two elements, and 24 studies to three elements). Note that studies with more than one A/B target typically apply these in multiple experiments. The dominating targets of A/B testing are algorithm, visual elements, and workflow/process which together make up 86.2% of all A/B targets reported in the primary studies. Notable, 32 primary studies did not specify a particular A/B target, for example using datasets from two prior A/B tests in the paper's evaluation without clarifying the details of these tests [385].

Application domain vs A/B target We can now map the application domains with the targets of A/B testing. This analysis provides insights into which elements or components are typically the subject of A/B testing in particular domains, or alternatively which A/B targets remain unexplored in particular domains. Table 4.6 presents this mapping. We highlight a number of key observations:

- A/B testing of algorithms is applied across all application domains and for all major domains it is the primary target of A/B testing. Commonly tested algorithms include feed ranking algorithms for social media websites, recommendation algorithms for news/multimedia websites, search ranking

Table 4.5: Identified A/B targets, with description.

A/B target	Description	Occ.
Algorithm	Updated version of an algorithm such as a recommendation algorithm [396], a search ranking algorithm [196], or an ad serving algorithm [21].	58
Visual elements	Changes to visual components such as updates to a website layout [39] or a general user interface update [89].	33
Workflow / process	Alteration to the workflow of an application, e.g. the addition of a feedback button to a dashboard [233], or a change in a user workflow, e.g. the process of a virtual assistant tool [208].	28
Back-end	Optimization of a software component that is not directly visible to the user, such as testing server optimizations [274] or adjusting application parameters for better performance [119].	10
New application functionality	Newly introduced functionality, such as a new widget on a web-page [59] or additional content that is presented to the user after performing a search query [238].	6
Other	This category comprises three other A/B targets: different timing and content of emails sent [394], varying educational resources presented to the user [293], and the page configuration of a website [351].	3
Unspecified	The target of the A/B test was not specified in the study.	32

algorithms for search engines, and advertisement serving algorithms both in the Web and search engine application domains.

- A/B testing of visual elements is particularly popular for search engines (16 studies) compared to other application domains such as Web (with only 6 studies). Typical examples include changes to the font color of search engine results [201] and changing the position of advertisements on the result page [249].
- Workflow and process elements as A/B targets are commonly applied across the major domains. This target is particularly popular for the Web and E-commerce (with 8 and 7 studies, respectively). Typical examples are changes to the process in which best-performing advertisements are determined in JD's advertisement platform, China's largest online retailer [359], and changes to the order assignment policy for on-demand meal delivery platforms [225].
- For the Web and search engines, all types of A/B targets are applied. The main focus for the Web is on algorithms and workflow/processes, while the focus for search engines is on algorithms, visual elements, and back-end. For the Web, we notice only a single primary study with back-end as the A/B target. This study targets different microservice configurations in A/B testing in order to tune individual microservices for performance improvements [327]. On the other hand, for search engines, we only noted three primary studies that target a workflow or process in A/B testing. One study evaluated a change of wording in digital advertisements [30], one study evaluated a change in advertisement strategies [163], the last study evaluated the option to pay for ”sponsored search” (to prioritize search results) [31].
- For e-commerce, we noticed that A/B testing is mainly used to test changes to ranking and recommendation algorithms, and to processes such as virtual assistants. Notably, we only identified a single primary study that evaluated changes to the user interface [226].
- A/B testing for back-end optimizations was identified to be most common for search engines, while we did not identify a paper in the e-commerce and finances domain where A/B testing was used for back-end changes.

Table 4.6: Application domain × A/B target

A/B target *Application domain*	**Algorithm**	**Visual elements**	**Workflow / process**	**Back-end**	**New app. func.**	**Other**
Web	17	6	8	1	3	0
Search engine	17	16	3	6	2	0
E-commerce	10	2	7	0	0	1
Interaction	5	6	2	2	1	0
Finances	7	2	4	0	1	0
Transportation	2	0	0	1	1	0
Other	2	1	3	0	0	2

Research question 1: What is the subject of A/B testing? The main targets of A/B testing are algorithms, visual elements, workflow and processes, and back-end features. A/B testing is commonly applied in the domains of the Web, search engines, e-commerce, interaction software, and finances. Algorithms are consistently tested across these domains. Visual elements are predominantly evaluated in search engines, and counter-intuitively not in e-commerce. Workflow and processes are popular A/B targets in the Web and e-commerce domains. On the other hand, back-end features such as server performance are popular targets for search engines.

4.4.3 RQ2: How are A/B tests designed? What is the role of stakeholders in this process?

To answer the second research question, we look at the following data items: A/B test type (**D10**), Used metrics (**D11**), Statistical methods employed (**D12**), and Role of stakeholders in the experiment design (**D13**).

Design of A/B tests

To answer the first part of RQ2 (How are A/B tests designed?), we take a deeper look at the design of the A/B tests, focusing on the type of A/B tests, A/B metrics, and statistical methods used in the A/B tests.

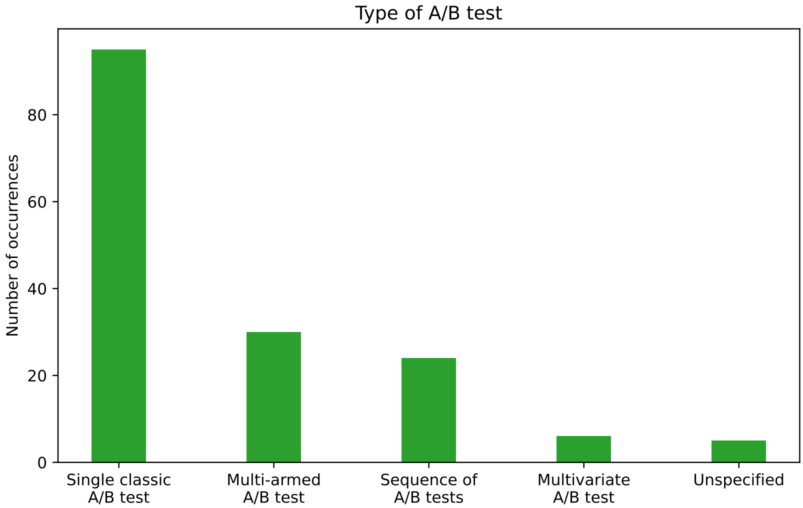


Figure 4.4: Identified A/B test types.

A/B test type. The type of A/B tests includes single classic A/B tests with two variants, A/B tests composed of more than two variants (denoted as multi-armed A/B tests), multivariate A/B tests where combinations of elements are tested in one A/B test, and sequences of all these types. Figure 4.4 shows the frequencies of these different A/B test types extracted from the primary studies.

Overall, we identified 155 occurrences of A/B test types, i.e., an average of 1.13 occurrences per primary study (123 studies considered a single type of A/B test, 17 studies considered two types, and one study considered three test types). The majority of the primary studies employed single classic A/B testing with a control variant and a treatment variant (95 occurrences). These standard tests are used to test a variety of targets. The second most common type of A/B test is a multi-armed A/B test (30 occurrences). This type of test is composed of more than two variants under test; for example one control variant as baseline and three treatment variants with a distinct version each. These tests are commonly used to evaluate multiple versions of a recommendation algorithm, e.g., [317, 333], and to test different advertisement serving algorithms, e.g., [346]. The third most common type of A/B test is a sequence of classic A/B tests (24 occurrences). Examples here include the comparison of multiple variants in a manually executed sequential style (as opposed to a multi-armed A/B test where all variants are deployed simultaneously) [123], manually testing multiple iterations of machine learning algorithms sequentially [228], and automatically executing a sequence of A/B tests to handle controlled feature release in DevOps [308]. The last identified A/B test type is the multivariate A/B test (6 occurrences). This type of test evaluates various combinations of multiple features. As opposed to a multi-armed A/B test, a multivariate A/B test enables testing variants of more than a single feature in a singular A/B test. An example is the comparison of different combinations of varying GUI elements [89].

A/B metrics. Table 4.7 lists the A/B metrics that we extracted from the primary studies. In total, 493 occurrences of A/B metrics were reported in the primary studies. With a total of 198 experiments spread over 141 studies, this gives an average of 2.12 metrics per experiment[5] (ranging from 1 to 8 metrics per experiment). The most common group of A/B metrics are engagement metrics (225 occurrences) that refer to the number of conversions[6], number of user sessions, time users are present on the website, and metrics related to the usage of the application or website (e.g. number of posts rated, number of bookings made).[7] The second largest group are click metrics (82 occurrences). Examples include the number of clicks, clicks per query, and good click rate[8]. The third group of A/B metrics we identified are metrics related to monetization, i.e., revenue and cost (64 occurrences). Examples include number of purchases, order value, revenue per e-mail opening, and advertisement cost. The next group is performance metrics (50 occurrences). Examples include a simple response time of an application, bandwidth used, end-to-end latency, or playback delay of audio. The remaining groups are metrics that track unwanted effects in the A/B tests (34 occurrences, e.g. abandonment rate or the number of un-subscriptions), views (21 occurrences, e.g. the number of page views or the number of product views), and user feedback (17 occurrences, e.g. the number of customer complaints or verbatim feedback).

Statistical methods Table 4.8 groups the types of statistical methods used for A/B tests in the primary studies. The most commonly used statistical method is hypothesis tests that test for equality (94 occurrences in total). The main test used in this group is a student t-test, e.g. [163, 156]. Other tests in this group are the Kolmogorov-Smirnov test, e.g., [311], Mann-Whitney test, e.g., [306], and Wilcoxon signed-rank test, e.g., [350]. Out of the 94 occurrences of this type of hypothesis test, 37 primary studies did not report the concrete test used in the analysis of the results[9]. The second most commonly used method is bootstrapping (11 occurrences). This method constructs multiple datasets by resampling the original dataset [98]. The newly constructed datasets are

[5] We excluded experiments and corresponding metrics of primary studies that analyzed a large number of previously conducted A/B tests.

[6] A conversion is a desired action taken in the A/B test.

[7] Note that some of the primary studies do not specify explicitly the A/B metrics due to business sensitivity. Based on the available information in the study, we have included these in general engagement metrics.

[8] Good clicks are described as clicks that are meaningful during the search query session [35].

[9] However, these studies did report p-values alongside the results, or explicitly refer to confidence intervals and statistically significant results of the A/B tests.

Table 4.7: Identified A/B metrics.

A/B metric	Number of occurrences
Engagement metrics	225
Click metrics	82
Monetary metrics	64
Performance metrics	50
Negative metrics	34
View metrics	21
Feedback metrics	17

Table 4.8: Statistical methods employed during A/B testing.

Statistical methods employed	Number of occurrences
Hypothesis - equality	57
Hypothesis - equality (concrete method unspecified)	37
Bootstrapping	11
Hypothesis - inference	8
Goodness of fit	8
Correction method	7
Estimator	6
Hypothesis - independence	5
Regression method	2

then typically used for equality hypothesis testing. The key benefit of this technique is the sensitivity improvements gained in the analysis of the results. However, a big drawback of the technique is that it is computationally expensive, especially for larger datasets [233]. The third most commonly used statistical method is a hypothesis test that tests for inference and goodness of fit (both 8 occurrences). Examples of inference hypothesis tests include using Bayesian analysis approach to ensure multiple simultaneously running experiments do not interfere [200], and a Bayesian approach to infer the causal effect of running ad campaigns [20]. Examples of goodness of fit methods include sequential testing methods that are based on likelihood ratio tests [188], and a Wald test [186]. The remaining groups are correction methods (7 occurrences) with e.g. Bonferroni correction [398]; custom estimators for observations in A/B testing (6 occurrences), e.g., an estimator that takes variances into account [232]; hypothesis tests for independence (5 occurrences), containing χ^2 tests [334]; and regression methods (2 occurrences), e.g. CUPED [100].

Role of stakeholders

To address the second part of RQ2 (What is the role of stakeholders in the design of A/B tests?), we analyze the role stakeholders play in the design of A/B tests.

Roles of stakeholders Table 4.9 lists the different roles of stakeholders in the design of A/B tests that we extracted from the primary studies, associated with tasks, descriptions, and examples. We identified three main roles: concept designer (127 occurrences), experiment architect (111 occurrences), and setup technician (31 occurrences). The role *Concept designer* consists of conceptualizing new ideas for A/B testing. The role of *Experiment architect* consists of calibrating technical parameters of the experiment such as the experiment duration. The role of *Setup technician* consists of taking the necessary steps required to allow the execution of the A/B test. The top task of the concept designer is designing and tuning variants of A/B tests (67 occurrences). The top task of the experiment architect is determining the duration of A/B tests (60 occurrences). Finally, the main task of the setup technician is performing post-design activities of A/B tests (25 occurrences).

Cross analysis A/B test design

We discuss two mappings of data items: The role stakeholders take in the design of A/B tests versus A/B test type; and the A/B metrics used in experiments versus the statistical methods employed.

Tasks of stakeholders vs A/B test type The mapping of stakeholder's tasks in the design of A/B tests across types of A/B tests is shown in Table 4.10. We observe the following:

- The primary tasks of stakeholders across all types of A/B tests are the design and tune of variants, determining the duration of experiments, the population, and the goal or hypothesis. These numbers confirm that these are essential design tasks for any A/B test.
- A majority of the studies that use multi-armed A/B testing and sequence of A/B tests report the design and tuning of variants as an important stakeholder task (22 and 13 occurrences respectively). Since these types of tests involve multiple variants under test, the studies often specify more details about the variants and the reasoning behind choosing which variants to test.
- Determining the goal or hypothesis for A/B testing is frequently mentioned for multi-armed A/B tests (17 occurrences). In contrast to conventional two-variant

Table 4.9: Roles and tasks of stakeholders in the design of A/B tests (Occ short for the number of occurrences).

Role	Task	Task description	Occ.
Concept designer (127)	Design and tune variants	Designing and tuning the variants to test. Examples are tweaking the A/B variants [317], or designing A/B variants for different kinds of populations (e.g., old vs new users) [36].	67
	Determine goal or hypothesis	Formulating the goal or hypothesis of the A/B test itself. Examples include the specification of a goal to find the better-performing news selection algorithm [103] or the specification of a predetermined hypothesis for the A/B test [6].	48
	Perform pre-design actions	Actions that are taken before designing the A/B test. Examples include providing motivation for A/B tests [351] or performing offline A/B tests before moving to online A/B testing [158].	12
Experiment architect (111)	Determine duration	Determining the duration of the A/B test. Examples include choosing a fixed experiment duration (e.g., 1 week) [6] or via an explicit expiration date [229].	60
	Determine population assignment	Determining the population that should take part in the A/B test. Examples include a simple 50/50 split of all users [381], an assignment where the target population is determined over a two-week period [393], or an assignment where network effects have to be taken into account [225].	51
Setup technician (31)	Perform post-design actions	Actions that are taken after completing the design of the A/B test. Examples include performing A/A testing prior to running the A/B test [403, 76], validation of the A/B test design [233], or scheduling the execution of the A/B test [351].	25
	Perform metric analysis and initialization	Analyzing and potentially initializing metrics for the A/B test. An example consists of instantiating a custom A/B utility metric with negative and positive weights tied to the user's actions during a search session [238].	6

Table 4.10: Tasks of stakeholders × A/B test type

Test type (total occ.) *Task*	**Single classic A/B test (95)**	**Multi-armed A/B test (30)**	**Sequence of A/B tests (24)**	**Multivariate A/B test (6)**
Design and tune variants	33	22	13	2
Duration	45	9	11	2
Population assignment	37	7	8	2
Goal/hypothesis	27	17	8	2
Post-design actions	12	1	5	0
Pre-design actions	6	4	2	1
Metric analysis/init.	5	1	0	0

A/B testing that typically involves a control variant and an altered variant aimed at improving the control variant, multi-armed A/B tests involve more than two variants, so practitioners often formulate hypotheses regarding the potential performance of each variant.

- Post-design actions are more often reported for sequences of A/B tests (5 instances). For instance, one primary study mentions modeling the sequence of A/B tests [308], another study mentions determining the success condition of the A/B tests before executing them [335], and another study refers to providing an outcome range of the A/B tests [338].
- Only a few primary studies report pre-design actions and metrics analysis and initialization, independently of the type of A/B test.

A/B metrics vs statistical methods used The statistical methods used across different types of A/B metrics are shown in Table 4.11.

- Engagement metrics and click metrics are used across all types of statistical methods.
- The concrete method used for hypothesis testing of equality is often not specified across all types of A/B metrics. For monetary and performance metrics in particular, a majority of studies do not mention the concrete hypothesis testing method (8 and 11 occurrences, respectively). This might be due to the sensitivity of reporting results for these types of metrics.
- Negative metrics are primarily used for hypothesis equality tests (10 and 8 occurrences for hypothesis equality and hypothesis equality with no method specified respectively).
- Hypothesis method for independence is most frequently used for the monetary metrics, yet, the use is uncommon (3 instances).
- The use of feedback metrics is also uncommon and if used, the specific statistical method used is not reported (5 occurrences).

Table 4.11: Statistical methods × A/B metrics (H short for hypothesis)

Metric *Method*	**Engag.**	**Click**	**Monetary**	**Negative**	**Perf.**	**View**	**Feedback**
H - equality	31	14	7	10	4	7	2
H - equality (unsp.)	24	12	8	8	11	5	5
Bootstrapping	9	2	2	3	3	1	1
H - inference	5	1	0	0	1	0	0
Goodness of fit	5	1	2	1	0	0	0
Correction method	4	1	1	1	2	0	1
Estimator	4	1	2	1	0	1	0
H - independence	2	2	3	0	0	1	0
Regression method	1	1	1	0	0	1	0

Research question 2: How are A/B tests designed? What is the role of stakeholders in this process? The primary type of A/B test is a single classic A/B test, followed by multi-armed A/B tests and sequences of A/B tests. Engagement metrics are the dominating type of A/B metrics used in A/B testing. Other prominent A/B metrics include click, monetary, and performance metrics. Hypothesis testing for equality is by far the most commonly used statistical method used in A/B testing. Remarkably, about 40 % of these studies that test equality do not specify the concrete method they use for that. Stakeholders have two main roles in the design of A/B tests: concept designer and experiment architecture. Less frequently reported is the third role of setup technician.

4.4.4 RQ3: How are A/B tests executed? What is the role of stakeholders in this process?

Execution of A/B tests

To address the first part of RQ3 (How are A/B tests executed?), we analyze the data collected during A/B tests, the evaluation methods used, and the use of A/B tests.

Data collected Table 4.12 lists the classes of data collected during the execution of A/B tests. We identified four types of data. Product or system data is most commonly reported in primary studies (48 occurrences). This data class includes the type of browser used by the end-user, the operating system of the end-user, hardware-specific information of the device used to interact with the application, and general information related to the usage of the system (e.g. tracking information about item categories of products in an e-commerce application, and types of search queries processed during the A/B test). The second most popular is user-centric data (26 occurrences). This class contains data related to how the end-user interacts with the system as well as personal information of end-users. Examples include scrolling characteristics of users on a web application, the navigation history of end-users, user feedback, and using age or current occupation of the end-user during analysis. The third most commonly reported class is spatial-temporal data (20 occurrences) which groups data related to geographic location and time-related data. Examples include timestamps of requests to an application, the creation date of accounts that take part in the A/B test, and spatial information such as the country and region of end-users. Lastly, a few primary studies report the use of secondary data (6 occurrences). Data in this class correspond to A/B metrics that do not serve as main evaluation metrics for A/B tests. Examples are the number of clicks or page views that are used for additional analysis after conducting the A/B tests.

Table 4.12: Data collected for the A/B tests.

Data collected	Number of occurrences
Product/system data	48
User-centric data	26
Spatial-temporal data	20
Secondary data	6

Table 4.13: Evaluation method used in the primary studies.

Evaluation method	Number of occurrences
Empirical evaluation	100
Simulation based on real empirical data	26
Simulation	15
Illustrative example	10
Case study	5
Theoretical	1

Evaluation method Table 4.13 summarizes the identified evaluation methods. The vast majority of primary studies provide results from an empirical evaluation (100 occurrences), i.e., executing A/B tests in live systems. A substantial number of studies use historical data from previously conducted A/B tests to simulate new A/B tests (26 occurrences), while a handful of studies (15 occurrences) use simulations without historical data as their evaluation method. Lastly, a few studies use illustrative examples (10 occurrences), case studies (5 occurrences), and a single primary study provides a theoretical evaluation [249].

Use of test results Table 4.14 lists the use of test results extracted from the primary studies. Use of test results refers to what stakeholders do with the obtained data and analyses of A/B tests, such as using the results to design additional A/B tests. As the table shows, the main usages of A/B test results are the selection and rollout of a feature (71 and 24 occurrences respectively). A number of studies aim at validating the effectiveness of the A/B testing process itself (12 occurrences). The use of test results to trigger a subsequent A/B test seems not very well explored (4 occurrences).

Table 4.14: Use of test results gathered from A/B test execution.

Use test results	Description	Occ.
Feature selection	The results of the A/B test are used to determine which variant presents an improvement to the application. Examples include selecting a new version of a ranking algorithm [271, 59] or a recommendation algorithm [135], and selecting a different visual design [9].	71
Feature rollout	The results of the A/B test are used to determine if the rollout of a feature should be continued or halted, as for example outlined by practitioners at Microsoft [383, 102].	24
Continue feature development	The results of the A/B test are used as a driving force for further feature development, e.g. fine-tuning newly proposed A/B metrics based on periodicity patterns after obtaining promising results [97], and further developing personalization methods [7].	17
Subsequent A/B test design	The results of the A/B test are used for future A/B test design, for example suggesting alternative A/B variants to test in future A/B tests [208], and designing a new A/B test to further test the quality of an A/B metric prediction model[10] [272].	15
Validation effectiveness of A/B testing process	The results of the A/B test are used to demonstrate the effectiveness of the newly proposed or improved A/B testing approach by the authors. Examples include evaluating a newly proposed counterfactual framework to run seller-side A/B tests in two-sided marketplaces [342], and the validation of a new statistical methodology for continuous monitoring of A/B tests [187].	12
Validation of a RQ	A/B testing is used to validate a research question put forward by the authors. One example consists of investigating the hypothesis under which circumstances companies should pay for advertising in search engines [31].	10
Bug detection / fixing	The results of the A/B test are used to detect potential bugs or validate bug fixes, e.g. probing for data quality issues in A/B tests of ML models to uncover potential bugs [228].	5
Subsequent A/B test execution	The results of the A/B test are used to execute subsequent A/B tests, e.g. using the results of A/B tests to automatically determine which subsequent A/B tests to execute [335].	4
Unspecified	The use of the test results was not specified in the study.	24

Role of stakeholders

To address the second part of RQ3 (What is the role of stakeholders in this process?), we analyze the role of stakeholders in A/B test execution.

Roles of stakeholders Table 4.15 lists the different roles of stakeholders in the A/B test execution we have extracted from the primary studies with associated tasks, a description, and examples. We identified two main roles: experiment contributor (40 occurrences) and experiment assessor (37 occurrences). The role *Experiment contributor* consists of managing the A/B test execution. The role *Experiment assessor* consists of evaluating the A/B test results and potentially undertaking additional actions. The top task of the experiment contributor is experiment supervision (19 occurrences). The top task of the experiment assessor is experiment post-analysis (17 occurrences).

Cross analysis A/B test execution

We take a deeper look at two mappings of data items related to the execution of A/B tests: Use of test results with the tasks of stakeholders in the execution of A/B tests; and the evaluation method with the tasks of stakeholders in the execution of A/B tests.

Use of test results vs Tasks of stakeholders in the execution of A/B tests The first mapping we analyze relates to the use of test results and the tasks stakeholders undertake in the execution of A/B tests. The results are shown in Table 4.16. We highlight some key observations:

- Experiment supervision is applied regardless of the usage of test results. For feature rollout as a use of A/B test results, the task of experiment supervision is often mentioned. Supervision takes on a key task in this context to ensure that the rollout happens in a hazard-free manner (i.e., no harm is caused to users) [383, 59].
- The task of experiment post-analysis is typically only reported for experiments that are fully complete (i.e., do not go through additional rounds of iteration). In the primary studies where the results of the A/B tests are used for subsequent A/B test design, no instances were identified where stakeholders take the task of performing post-analysis on the results of the experiments.
- For subsequent A/B test design, the task of experiment triggering is often mentioned. This is to be expected since the newly designed A/B tests also need to be executed. Additionally, A/B test termination is also mentioned often (e.g., terminating an experiment due to bad results [165]).

Table 4.15: Identified roles and concrete tasks of stakeholders during the execution of A/B tests.

Role	Task	Task description	Occ.
Experiment contributor (40)	Experiment supervision	Monitoring and closely following up on the execution of A/B tests [335, 93].	19
	Experiment alteration	Altering aspects of the A/B test during its execution. Examples include adjusting the population assignment of the experiment [76], or adjusting the A/B variants themselves [338].	12
	Experiment termination	Stopping A/B tests when deemed necessary. Examples include manually stopping A/B tests when sufficient data is collected [208], or stopping the experiment early when harm is observed [200].	9
Experiment assessor (37)	Experiment post-analysis	Various steps that are taken after analyzing the results of the A/B test. Examples include double checking results from executed A/B tests [156], performing a deeper analysis of suspicious results [119], or performing bias reduction techniques on the retrieved data from the A/B tests [233].	17
	Experiment triggering	Starting the execution of (subsequent) A/B tests [383, 39].	13
	Other	This category encompasses a few niche tasks, such as documenting the findings and learning from conducting the A/B test [321], rerunning A/B tests [238], or incorporating user feedback in the analysis of the A/B tests [229].	7

Table 4.16: Use of test results × Tasks of stakeholders in the experiment execution ("cont. feature dev." is short for "continue feature development", "val. eff." is short for "validation of effectiveness", and "val. of a RQ" is short for "validation of a research question").

Task *Use*	**Super-vision**	**Post-analysis**	**Triggering**	**Alteration**	**Termin-ation**
Feature selection	8	11	6	8	4
Feature rollout	10	4	6	6	4
Cont. feature dev.	7	3	5	2	3
A/B test design	6	0	5	2	3
Val. eff. A/B testing	1	2	1	1	1
Val. of a RQ	1	1	0	1	1
Bug detection/fixing	4	0	3	2	2
A/B test execution	1	0	1	0	0

- In the case of bug fixing and detection, stakeholders typically supervise experiments (either to detect possible bugs in the code or ensure the bugfix is effective) [117], and trigger the experiments (i.e. launch an experiment explicitly to fix a known bug in the application) [228].

Evaluation method vs Tasks of stakeholders in the execution of A/B tests

In addition, we analyze the tasks stakeholders undertake during the execution of A/B tests across the evaluation methods. This mapping is shown in Table 4.17. We highlight a number of key takeaways:

- All tasks that stakeholders undertake in the execution of A/B tests are widely encountered in the case of empirical evaluation.
- For the method of simulation based on real empirical data, the task of post-analysis is reported more often than any other task. An example is looking for outliers in the analysis of the results of A/B tests, and using historical experiments to confirm its effectiveness [167].
- Primary studies that use simulation as an evaluation method rarely specify the tasks stakeholders undertake in the execution of A/B tests. We hypothesize that, since simulations allow for a more controlled way of conducting A/B tests, the tasks stakeholders undertake after the design of A/B tests are not pertinent.
- The only stakeholder task reported for theoretical evaluation is experiment alteration (primary study [249]).

Table 4.17: Evaluation method × Tasks of stakeholders in the test execution ("emp. sim." short for "simulation based on real empirical data", "ill." short for "illustrative").

Task *Method*	**Supervision**	**Post-analysis**	**Triggering**	**Alteration**	**Termination**	**Other**
Empirical	14	13	10	10	6	6
Emp. sim.	2	4	1	0	1	0
Simulation	1	1	1	0	0	0
Ill. example	2	0	1	2	2	1
Case study	0	0	0	0	0	0
Theoretical	0	0	0	1	0	0

Research Question 3: How are A/B tests executed in the system? What is the role of stakeholders in this process? The main types of data collected during the A/B test execution relate to the product/system, users, and spatial-temporal aspects. The dominating evaluation method used in A/B testing is empirical evaluation, but a relevant number of studies also use simulation. A/B test results are primarily used for feature selection, followed by feature rollout, and continued feature development. (Automatic) subsequent A/B test execution is only used marginally. The main reported roles of stakeholders in A/B test execution are experiment contributor (with experiment supervisor as the main task) and experiment assessor (with experiment post-analysis as the main task).

4.4.5 RQ4: What are the reported open research problems in the field of A/B testing?

To answer research question 4, we analyze the results of data item Open problems (**D18**).

Table 4.18 presents a categorization of open problems we have identified in the primary studies. For each category, we devised concrete sub-categories of open problems. We elaborate on each type of open problem with illustrative examples.

Evaluation-related open problems

First, we established three sub-categories of open problems that are related to the evaluation of the proposed approach: (1) extensions to the evaluation of the approach presented in the primary study, (2) a more thorough analysis of the approach presented

Table 4.18: List of identified open problems.

Open problem category	Open problem sub-category	Number of occurrences
Evaluation-related	Extend the evaluation	21
	Provide thorough analysis of approach	16
	Other evaluation-related	34
Process-related	Add process guidelines	9
	Automate process	7
Quality-related	Enhance scalability	7
	Enhance applicability	6

in the primary study, and (3) Other evaluation-related open problems in the primary study.

Extend the evaluation Drutsa et al. [97] explore periodicity patterns in user engagement metrics, and their influence on engagement metrics in A/B tests. Moreover, the authors put forward new A/B metrics that take such periodicity patterns into account, resulting in a more sensitive A/B test analysis. The authors evaluated the proposed metrics on historical A/B test data from Yandex, though they state that further evaluation of the approach could be carried out in different domains such as social networks, email services, and video/image hosting services. From a slightly different point of view, Barajas et al. [20] developed a technique to determine the causal effects of marketing campaigns on users, putting the focus on the campaign itself rather than only focusing on the design of advertisement media. The authors put forward specific guidelines on randomizing and assigning users to advertising campaigns, and provide a technique to estimate the causal effect the campaigns have on the users under test. As a point of future work, the authors posit a different evaluation question concerning what would have happened if the technique would have been applied to the whole population.

Provide thorough analysis of approach An example of this category is mentioned by Peska and Vojtas [272]. The authors put forward a way of evaluating recommendation algorithms in small e-commerce applications both offline and online via A/B testing. The approach compares the results of offline evaluation of recommendation algorithms with the results of online A/B testing of the algorithms. Moreover, the authors then used these data to build a prediction model to determine the promising recommendation algorithms more effectively due to the knowledge obtained from online A/B testing. As future work, the authors list that further work is necessary to verify the causality of an effect observed in the analysis of offline and online A/B testing data. In another primary study written by Madlberger and Jizdny [240], the authors perform an analysis of the impact of social media marketing on click-through rates and customer engagement. To accomplish this, they run multiple social media marketing campaigns using A/B testing, evaluating hypotheses related to the impact of visual and content aspects of advertisements on the click rates of end-users. For future research, the authors report that a more comprehensive investigation is necessary to ascertain why some hypotheses in the study have been rejected.

Other evaluation-related An example of other evaluation-related open problems is laid out by Gruson et al. [158]. The authors propose a methodology based on counterfactual analysis to evaluate recommendation algorithms, leveraging both offline evaluation and online evaluation via A/B testing. The approach comprises A/B testing recommendations to a subset of the population, and using the results of these tests to de-bias offline evaluations of the recommendation algorithm based on historical data. In regards to open problems, the authors mention exploring additional metrics for the approach, as well as potential improvements that can be made to the estimators they use in the approach. Another example is specified by Ju et al. [188], who present an alternative to standard A/B testing with a static hypothesis test by putting forward a sequential test. Classically in A/B testing, the hypothesis of the test is tested after a fixed time and conclusions are made based on the final result. The sequential test put forward by the authors does not have a predetermined number of observations, rather at multiple points during the experiment, the test determines whether the hypothesis can be accepted, rejected, or if more observations are required. For future work, the authors wish to support A/B/n experiments in their approach, as well as extend the procedure for data that follows a non-binomial distribution. In a final example, Gui et al. [160] study ways of dealing with interference of network effects in the results of A/B tests. One of the fundamental assumptions of A/B testing is that users are only affected by the A/B variant they are assigned to. However, network effects can undermine this assumption due to interaction between users in the population. The authors demonstrate the presence of network effects at LinkedIn and propose an estimator for the average treatment effect that also takes potential network effects into account. As a line of future research, the authors want to investigate ways of enhancing the approach such that it can deal with more real-life phenomena.

Process-related open problems

Second, we established two sub-categories of open problems that are process-related: (1) guidelines to the A/B testing process and (2) automation of aspects of the A/B testing process.

Add process guidelines In an effort to provide more nuanced A/B testing guidelines in the e-commerce domain, Goswami et al. [156] discuss controlled experiments to make decisions in the context of e-commerce search. Considerations such as how to prioritize projects for A/B testing for smaller retailers and how to conduct A/B tests during holiday time are left as open questions. A different primary study covering the benefits of controlled experimentation at scale is presented by Fabijan et al. [117]. In this study, the authors present multiple examples of conducted A/B tests, and the corresponding lessons learned from these experiments. One of the listed open problems in the study relates to providing "guidance on detection of patterns between leading and lagging metrics".

Automate process Mattos et al. [247] present a step towards automated continuous experimentation. The authors put forward an architectural framework that accommodates the automated execution of A/B tests and the automated generation of A/B variants. To validate the framework, an A/B test was conducted with a robot. One of the open challenges laid out in the study comprises the ability to automatically generate hypotheses for A/B tests based on the collected data. Duivesteijn et al. [102] present A&B testing, an approach that leverages exceptional model mining techniques to target A/B variants to subgroups in the population under test. As opposed to deploying the best-performing variant of the A/B test, the authors put forward running both variants (if ample resources are available) and targeting specific variants to individual users based on their inferred subgroups. One of the potential avenues for future research consists of the development of a framework that would enable automated personalization of websites supported by A/B testing.

Quality-related open problems

Lastly, we established two sub-categories of open problems that are quality-related: (1) enhancing scalability of the proposed approach, and (2) enhancing the applicability of the approach.

Enhance scalability One example of this is presented by Zhao et al. [403]. To obtain a causal explanation behind the results of A/B tests, the authors propose segmenting the population, and consequently analyzing the results of the A/B test in individual segments. For future work, the authors mention developing a more scalable solution that integrates the approach into their existing experimentation platform. To address online experimentation specifically for cloud applications, Toslali et al. [343] introduce Jackpot, a system for online experimentation in the cloud. Jackpot supports multivariate A/B testing and ensures proper management of interactions in the cloud application during the execution of A/B tests. As a venue for future work, the authors mention ways of dealing with the limited scalability of multivariate experimentation due to the number of potential experiments increasing exponentially with the number of elements to be tested.

Enhance applicability One such study explores A/B testing in the automotive industry [236]. The study addresses concerns relating to the limited sample sizes of A/B tests obtain due to the limited nature of participants that can take part in A/B tests in the industry. To overcome this hurdle, the authors provide specific guidelines for performing A/B testing and determining the assignment of users to either the control or treatment variant in the test. However, one limitation pertains to requiring pre-experimental data to ensure a balanced population assignment between both A/B variants. In an effort to increase sensitivity in A/B testing, Liou and Taylor [232] propose a new estimator for A/B testing that takes the variance of individual users into account. To realize this, pre-experiment data of individual users is analyzed and variances are computed. To validate the approach a sample of 100 previously conducted A/B tests were collected and analyzed using the new approach. A big limitation noted by the authors is that "a stronger assumption about the homogeneity of the treatment effect" is required for the approach to remain unbiased.

Research Question 4: What are the reported open research problems in the field of A/B testing? The most commonly reported open problems directly related to the proposed approach, in particular improving the approach, extending the approach, and providing a thorough analysis. Other less frequently reported open problems related to the A/B testing process, in particular adding guidelines for the A/B testing process, and automating the process. Finally, a number of studies report open problems regarding quality properties, specifically enhancing the scalability and applicability of the proposed approach.

4.5 Discussion

In this section, we discuss a number of additional insights we obtained. We start with the research topics studied by the primary studies. Next, we look at environments and tools used for A/B testing. Then we report a number of opportunities for future research. We conclude with a discussion of threats to the validity of the study.

Table 4.19: Research topics of primary studies.

Topic	Occ.	Primary studies
Application of A/B testing	51	[396, 262, 207, 225, 21, 20, 249, 389, 218, 330, 76, 155, 137, 109, 35, 394, 231, 123, 346, 334, 135, 381, 390, 58, 319, 2, 6, 300, 333, 8, 215, 327, 317, 393, 31, 57, 9, 240, 7, 253, 103, 213, 305, 271, 39, 264, 279, 356, 138, 3, 397]
Improving efficiency of A/B testing	20	[1, 59, 274, 382, 41, 195, 99, 84, 96, 196, 89, 97, 98, 232, 221, 188, 30, 167, 82, 126]
Beyond standard A/B testing	18	[385, 83, 187, 100, 342, 168, 238, 299, 158, 308, 272, 60, 247, 163, 102, 245, 68, 335]
Concrete A/B testing problems	17	[307, 160, 387, 228, 326, 93, 359, 19, 226, 236, 156, 43, 343, 306, 208, 224, 52]
Pitfalls and challenges of A/B testing	13	[204, 199, 113, 119, 386, 92, 388, 248, 12, 91, 233, 311, 201]
Experimentation frameworks and platforms	13	[321, 345, 229, 350, 230, 10, 293, 162, 36, 81, 403, 398, 338]
A/B testing at scale	9	[200, 357, 118, 383, 186, 351, 165, 117, 116]

4.5.1 Research topics

During data extraction of the 141 primary studies, we noted the general subject matters of the primary studies and categorized the primary studies along 7 research topics. Table 4.19 summarizes these 7 topics. Note that studies share overlapping topics. We will now briefly explain each category and provide a few examples from the primary studies.

Application of A/B testing

The main focus of the primary study is the use and application of A/B testing as an evaluation tool for the main subject matter of the study (e.g. evaluation new recommendation algorithms, interface redesigns, etc[11]).

[11] See data item *A/B target* in Section 4.4.2 for specific references.

Improving the efficiency of A/B testing

This topic is about improving the process of A/B testing by exploring ways of improving sensitivity in A/B testing data [96, 274, 382, 195], investigating sequential testing techniques to stop A/B tests as soon as reasonable [188, 196, 1], proposing techniques to detect invalid A/B tests[12] [59], and using extra data such as periodicity patterns in user behavior to improve A/B testing [97].

Beyond standard A/B testing

This topic is about techniques that go beyond standard A/B testing, such as the use of new types of A/B metrics [385, 100, 238], use of counterfactuals in the evaluation of A/B tests[13] [342, 299], investigating ways of automating parts of the A/B testing process [308, 247, 245, 335], improving or altering the A/A testing process [168, 60], and investigating ways of combining offline- and online A/B testing [158, 272].

Concrete A/B testing problems

This topic includes A/B testing in specific domains and specific types of A/B testing. Examples include A/B testing for the e-commerce domain [208, 156], network A/B testing or A/B testing in marketplaces [226, 160, 43], A/B testing in the CPS domain with digital twins [93], and A/B testing for mobile applications [224, 387].

Pitfalls and challenges of A/B testing

This topic is about pitfalls related to conducting A/B testing [113, 199, 92, 91], or (particular domain-related) challenges related to A/B testing [388, 248, 233].

Experimentation frameworks and platforms

This topic covers papers that present an A/B testing platform [229, 338, 321], or a framework concerning aspects related to the A/B testing process such as a framework for detecting data loss in A/B tests [162], a framework for the design of A/B tests [81], or a framework for personalization of A/B testing [345].

[12] Invalid refers to badly designed experiments or misinterpretation of the results retrieved from the experiment.

[13] Counterfactual analysis provides answers to the cause and effect of the treatment group and their corresponding outcomes, compared to what would have happened if the treatment would not have been applied.

Table 4.20: Environments and tools used for A/B testing.

Environment	Number of occurrences
In-house experimentation system	20
Research tool or prototype	13
Commercial A/B testing tool	10
Commercial non A/B testing tool	7
User survey	1

A/B testing at scale

Primary studies under this topic focus on conducting A/B testing at a large scale, e.g., considerations for conducting A/B testing at scale [186, 351, 165], process models or guidelines for A/B testing at scale [118, 383], or concrete scalable solutions such as a scalable statistical method for measuring quantile treatment effects for performance metrics in A/B tests [357].

4.5.2 Environments and tools used for A/B testing

In addition to the research topics covered in the primary studies, we also analyze the environments and tools that were used to realize A/B testing, see Table 4.20.

The most commonly mentioned type of environment is an in-house experimentation system for A/B testing (20 occurrences), for instancem dedicated environments developed by companies such as Microsoft [228], Google [338], eBay [351], and Etsy [188]. These environments broadly support executing A/B tests. Furthermore, some primary studies describe concrete features of the experimentation system to help design A/B tests, e.g. controlling for bias during the specification of A/B tests in Airbnb's Experimentation Reporting Framework [221]. Next, we observe research tools and prototypes (13 occurrences). Examples include a tool to perform online cloud experimentation [343], a research prototype for A/B testing implemented in NodeJS [308], a tool for A/B testing with decision assistants [208], and a tool that enables automatic execution of multiple A/B tests [335]. The remaining environments we identified were commercial A/B testing tools (10 occurrences), e.g., Optimizely [253], and Google Analytics [39]; commercial tools not related to A/B testing (7 occurrences), e.g., Crazy egg [39], a heat mapping tool used to design A/B variants, and using Yahoo Gemini (advertisement platform) to test different advertising strategies [240]; and a user survey (1 occurrence) to determine which A/B variants to test by conducting a preliminary survey.

4.5.3 Research opportunities and future research directions

From our study, we propose a number of potential future research directions in the field of A/B testing. Concretely, we provide three lines of research: research on further improving the general process of A/B testing, research on automating aspects of A/B testing, and research on the adoption of proposed statistical methods in A/B testing.

Improving the A/B testing process

One future direction relates to taking considerations when running many A/B tests at once [338]. Plenty of studies cover this topic by e.g., discussing lessons learned in unexpected A/B test results that were caused by other A/B tests that were running in parallel [113], or manually checking for possible effects of running A/B tests by analyzing the deployed A/B tests in the system [351]. Yet, we did not encounter a study that puts forward a systematic approach to tackle this problem.

Another avenue for future research is about improving the sensitivity in A/B tests by, e.g., combining different sensitivity improvement techniques as pointed out by Drutsa et al. [96], enabling proactive prediction of user behavior in A/B tests based on historical data [97], and a deeper study of A/B test estimators to achieve better sensitivity as mentioned by Poyarkov et al. [274].

The last avenue for future research in improving the A/B testing process relates to providing further guidelines and designing principles for choosing and engineering A/B metrics. We highlight two primary studies that mention open problems related to this opportunity: Kharitonov et al. [195] put forward learning sensitive combinations of A/B metrics as a general open problem, and Duan et al. [100] discuss investigating dynamics between surrogate metrics and the actual underlying metric.

Automation

In an effort to establish continuous experimentation, multiple studies put forward steps companies can take to develop an experimentation culture, e.g. [118, 388, 114]. In light of expanding this experimentation culture, (partial) automation of the A/B testing process is essential to enable and empower continuous experimentation [59, 156]. Initial research on automation of steps in the A/B testing has been conducted, for example, presented by Tamburrelli et al [335] and Mattos et al. [247], see Sections 4.5.1 and 4.4.5. Yet the present state of research on this topic suggests that further investigation and more in-depth solutions are necessary to fully exploit the automated design and execution of A/B tests. Additionally, a number of open problems still remain that could facilitate and enable automated experimentation, e.g., determining which A/B tests to prioritize at execution [156], and automatically generating insights related to the rationale and cause of experiment results to experiment developers to guide product development [388].

Adoption and tailoring statistical methods

Even though a number of primary studies discuss bootstrapping as a technique to evaluate the results of A/B tests [345, 2, 156], bootstrapping remains largely unexplored in A/B testing, despite the fact that this statistical method has the potential to improve the analysis of A/B test results [186, 19]. Moreover, bootstrapping can present an invaluable tool to provide statistical insights into the results of the tests which could e.g. not be obtained by a standard equality testing method [104]. However, one big downside of bootstrapping is that it is computationally expensive [233]. Alongside the adoption of known statistical methods, designing and tailoring new statistical methods to accommodate particular experimentation scenarios presents an interesting research direction. One example is mentioned by Kharitonov [196], who put forward designing a custom statistical test for non-binomial A/B metrics. Another example concerns taking into account "the effects from multiple treatments with various metrics of interest" to tailor the approach presented by Tu et al. [345] for optimal treatment assignments in A/B testing by leveraging causal effect estimations.

Besides a limited number of primary studies employing bootstrapping in the analysis of A/B tests, a significant number of studies mention statistically significant results or p-values in the analysis of conducted A/B tests without specifying the concrete statistical test used (37 occurrences). Moreover, a considerable number of studies do not report anything related to statistical analysis (47 occurrences). We argue that this information is important to report in research publications and urge authors to specify the concrete statistical methods used[14] to obtain the results in the studies.

4.5.4 Threats to validity

In this section, we list potential threats to the validity of the systematic literature review [11].

Internal validity

Internal validity refers to the extent to which a causal conclusion based on a study is warranted. One threat to the internal validity is a potential bias of researchers that perform the SLR, which may have an effect on the data collection and the insights

[14] Or alternatively an explicit mention of lack of statistical methods used.

derived in the study. In order to mitigate this threat, we involved multiple researchers in the study. Multiple researchers were responsible for selecting papers, extracting data, and analyzing results. In each step, cross-checking was applied to minimize bias. Extra researchers were involved if no consensus could be found. Additionally, we defined a rigid protocol for the systematic literature review.

External validity

External validity refers to the extent to which the findings of the study can be generalized to the general field of A/B testing. A threat to the external validity of this systematic literature review is that not all relevant works are covered. To mitigate this threat, we searched all main digital library sources that publish work in computer science. Secondly, we defined the search string by including all commonly used terms for A/B testing to ensure proper retrieval of relevant works. Lastly, we also applied snowballing on the selected papers from the automatic search query to uncover additional works that might have been missed.

Conclusion validity

Conclusion validity refers to the extent to which we obtained the right measure and whether we defined the right scope in relation to what is considered research in the field of A/B testing. One threat to the conclusion validity is the quality of the selected studies; studies of lower quality might produce insights that are not justified or applicable to the general field of A/B testing. To mitigate this threat, we excluded short papers, demo papers, and roadmap papers from the study. Furthermore, we evaluated a quality score for each selected paper. Papers with a quality score ≤ 4 were excluded from the study.

Reliability

Reliability refers to the extent to which this work is reproducible if the study would be conducted again. To mitigate this threat, we make all the collected and processed data available online. We also defined a specific search string, a list of online sources, and other details in the research protocol to ensure reproducibility. The bias of researchers also poses a threat here, influencing that similar results would be retrieved if the systematic literature review would be conducted again with a different set of reviewers.

4.6 Conclusion

A/B testing supports data-driven decisions about the adoption of features. It is widely used across different industries and key technology companies such as Google, Meta, and Microsoft. In this systematic literature review, we identified the subjects of A/B tests, how A/B tests are designed and executed, and the reported open research problems in the literature. We observed that algorithms, visual elements, and changes to a workflow or process are most commonly tested, with web, search engine, and e-commerce being the most popular application domains for A/B testing. Concerning the design of A/B tests, classic A/B tests with two variants are most commonly used, alongside engagement metrics such as conversion rate or number of impressions as metrics to gauge the potential of the A/B variants. Hypothesis tests for equality testing are broadly utilized to analyze A/B test results, and bootstrapping also garners interest in a few primary studies. We devised three roles stakeholders take on in the design of A/B tests: Concept designer, Experiment architect, and Setup technician. Regarding the execution of A/B tests, empirical evaluation is the leading evaluation method. Besides the main A/B metrics, data concerning the product or system, and user-centric data are collected the most to conduct a deeper analysis of the results of the A/B tests. A/B testing is most commonly used to determine and deploy the better-performing A/B variant, or to gradually roll out a feature. Lastly, we devised two roles stakeholders take on in the execution of A/B tests: Experiment contributor, and Experiment assessor.

We identified seven categories of open problems: improving proposed approaches, extending the evaluation of the proposed approach, providing a thorough analysis of the proposed approach, adding A/B testing process guidelines, automating the A/B testing process, enhancing scalability, and enhancing applicability. Leveraging these categories and observations made during the analysis, we provide three main lines of interesting research opportunities: developing more in-depth solutions to automate stages of the A/B testing process; presenting improvements to the A/B testing process by examining promising avenues for sensitivity improvement, systematic solutions to deal with interference of many A/B tests running at once, and providing guidelines and designing principles to choose and engineer A/B metrics; and lastly the adoption and tailoring of more sophisticated statistical methods such as bootstrapping to strengthen the analysis of A/B testing further.

Acknowledgement

We thank Michiel Provoost for his support to this study.

Chapter 5

Automating A/B Testing Pipelines with Population Split

Publication details. This chapter is based entirely on a conference article that is under submission [281].

Personal contributions. Conceptualization (80%), Methodology (80%), Software (100%), Validation (70%), Formal analysis and interpretation results (70%), Writing (70%), Visualization (100%).

Positioning. Traditionally, A/B tests are conducted sequentially, with each experiment targeting the entire population of the corresponding application. This approach can be time-consuming and costly, particularly when the experiments are not relevant to the entire population. To tackle these problems, we introduce a new self-adaptive approach called AutoPABS, short for Automated Pipelines of A/B tests using Self-adaptation, that (1) automates the execution of pipelines of A/B tests, and (2) supports a split of the population in the pipeline to divide the population into multiple A/B tests according to user-based criteria, leveraging machine learning. We started the evaluation with a small survey to probe the appraisal of the notation and infrastructure of AutoPABS. Then we performed a series of tests to measure the gains obtained by applying a population split in an automated A/B testing pipeline, using an extension of the SEAByTE artifact.

5.1 Introduction

A/B testing, also referred to as online controlled experimentation, continuous experimentation, bucket testing, or randomized experimentation, forms a crucial part of modern software businesses such as Google, Amazon, or Meta. A/B testing supports businesses to grow and innovate their customer-facing software applications [342, 358, 227, 338, 229, 387, 219]. The aim of A/B testing is to make data-driven decisions to improve the products offered to customers. In essence, A/B testing compares two different versions of a software product or service, variant A and variant B, by exposing them to end-users and evaluating the performance of each variant. Unlike traditional software testing methods, A/B testing takes place within live systems and provides real-world data that organizations can use to make well-informed decisions [197, 322, 85].

The A/B testing process comprises three phases: design, execution, and analysis [178]. The design of the A/B test consists of defining key parameters such as the hypothesis to compare the variants, experiment duration, the assignment of users to both variants and metrics to be collected are defined (designed phase). We refer to the group of users that take part in the A/B test as the A/B test population. The metrics, such as click-through rate, number of clicks, and number of sessions [26, 200], are used to quantify the performance of each variant during the experiment. Once the experiment is designed, both variants are deployed in the live system and the population is split between them (execution phase), as shown in Figure 5.1. The system tracks relevant data during the test, and after the experiment is complete, the hypothesis is tested using a statistical test, e.g. a Student's t-test or Welsh's t-test [238, 274, 156]. The test results provide valuable insights into the performance of each variant, and organizations can use this information to make decisions about which variant to use (analysis phase).

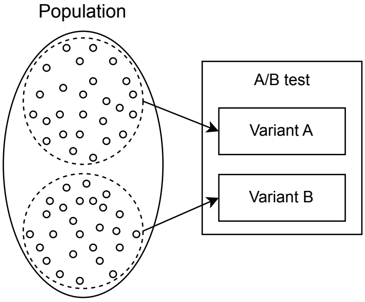


Figure 5.1: Example of 50/50 population assignment in the execution of an A/B test.

Traditionally, A/B tests are administered manually, and the results of the tests are analyzed sequentially [300, 334]. This process can be costly and time-consuming [202, 164, 297, 141]. Different researchers have therefore argued for further automating the A/B testing process [156, 59, 117, 247, 214, 141]. Automating A/B testing is the first challenge we aim to tackle in this chapter. Furthermore, conducting A/B tests on the entire population may not be optimal and even result in irrelevant outcomes due to the diverse characteristics of the population [388, 402]. Although previous work has looked into a segmented analysis of A/B tests [403], the execution of the A/B tests themselves do not target particular segments of the population. Yet, targeting specific segments of the population for A/B testing can increase the efficiency of the analysis of A/B test results. Targeting A/B tests to specific segments of their user base is the second challenge we aim to tackle in this chapter. These challenges are also confirmed by a recent systematic literature review we performed on A/B testing [286]. This brings us to the following problem statement we tackle in this chapter:

How can we automate A/B testing pipelines and run them efficiently?

To address this problem, we propose a self-adaptive solution called AutoPABS (Automated Pipelines of A/B tests using Self-adaptation) that handles the deployment, monitoring, analysis, and execution of pipelines of A/B tests automatically. AutoPABS interprets the outcome of A/B tests to initiate subsequent tests specified in the pipeline. To enhance efficiency, we introduce population splits to A/B testing pipelines to target A/B tests to specific segments of the population. We focus on segmenting a population based on the properties or behaviors of users, leveraging machine learning. Segmenting the population enables multiple parts of an A/B testing pipeline to be executed in parallel improving the efficiency of the test execution. We evaluate the usefulness of AutoPABS with a small survey and test the gains of a population split for an online web-store application.

In contrast to most research on self-adaptation that aim at novel approaches for engineering self-adaptive systems, AutoPABS takes a complementary angle and aims at supporting a key task of software engineers using self-adaptation, in particular enhancing the automation of executing A/B testing pipelines. This aligns with recent initiatives, such as Self-Adaptation 2.0 [42] that argues for an equal-to-equal relationship between self-adaptation and AI, benefiting one another, and self-adaptation that is applied to deal with degraded machine-learning components to maintain system utility [55].

This chapter presents the following three contributions:

- A specification and notation for A/B testing pipelines
- A self-adaptive architecture that enables automated execution of A/B testing pipelines
- A population split component that enables more efficient A/B pipeline execution

The chapter is structured as follows. Section 5.2 describes related work on automating A/B testing and the use of machine learning in A/B testing. In Section 5.3, we present AutoPABS, the new approach for automating pipelines of A/B tests using self-adaptation and we introduce population splits in pipelines. In Section 5.4, we present the evaluation of AutoPABS. Finally, we wrap up and look at future research directions in Section 5.5.

5.2 Related Work

We discuss a selection of related work for the two main lines of related research: the automation of A/B testing pipelines and the use of machine learning in A/B tests. Then, we position our work in the current landscape of research.

Automation of A/B testing pipelines. Automation of A/B testing has received limited attention in the literature. Tamburelli et al. [335] approach A/B testing as an optimization problem that is solved using automated search. Developers annotate program features and the framework automatically generates, selects, and enacts A/B variants. Mattos et al. [245] put forward an architecture framework to model automated experimentation in software systems that they briefly evaluate in a human-robot context. Fabijan et al. [114] describe an iterative software engineering process to accelerate the use of A/B testing from experience at Microsoft, Outreach, and Booking.com, lowering the human cost of A/B testing and accelerating innovation. Researchers have also studied A/B tests and automated deployment based on principles of workflow and task orchestration. Révész et al. [291] target long-term A/B tests and automated deployment in the context of CI/CD, leveraging container orchestration systems to realize the approach. In an alternative setting, a challenging issue concerning automating A/B tests involves the identification of machine learning models that achieve satisfactory results in a live context [344] (as opposed to offline evaluation based on historical data [37, 230, 158]). To that end, Dai et al. [77] present an approach that automatically selects machine learning models to A/B test in live systems, giving priority to promising models. Another perspective of automating A/B testing pipelines is the gradual roll-out of software releases. Schermann et al. [308] present a modeling approach that supports a gradual roll-out of live testing of a system by setting up multiple sequential A/B tests. The Follow-The-Best-Interval algorithm proposed by Munoz et al. [261] handles the roll-out process automatically. Gerostathopoulos et al. [145] present a tool for end-to-end optimization of a target system, providing a basis for a system to self-optimize through automated experimentation.

Related industrial efforts include Feature Flags and Argo Rollouts. Feature Flags [124] are if/else controls in a code base. This industrial approach enables faster and safer development, making it easy to manage features without pushing a change by separating deployment from release. Argo Rollouts [15] enables a user to run two versions of an application for a specific duration and perform an analysis of their application, for instance, start a baseline and canary deployment in parallel, and compare the metrics produced by the two.

Use of machine learning in A/B tests. Several researchers have used machine learning to improve the execution of A/B tests. One such case is learning sensitive metric combinations in A/B testing [195]. Other work looks at increasing the sensitivity in A/B testing, yielding more reliable and faster outcomes. Guo et al. [161] and Syrgkanis et al. [332] use linear regression predictions of experiment outcomes alongside variance estimators to improve variance reduction in A/B testing, leading to more precise inferences with less data. Poyarkov et al. [274] propose a similar approach using boosted decision tree regression. From a different angle, Li et al. [227] make use of diversified historical data and machine learning to make predictions on the A/B metrics of A/B tests, but without running the tests on live systems. Conversely, Zhao et al. [401] employ unsupervised learning techniques in the analysis phase of A/B testing to classify users based on their behavior and analyze test results accordingly.

Positioning and Challenges Tackled. The state-of-the-art in A/B testing points to the labor-intensiveness of setting up, analyzing, and conducting A/B tests. Hence an important challenge is further automation of A/B testing. In addition, current research highlights that A/B tests are costly to run in live software systems (A/B tests are typically run for a long time to obtain ample observations). This underpins the challenge of running A/B tests more efficiently, for instance by using machine learning to improve the sensitivity of the A/B tests. Our work aims at contributing to these two challenges, on the one hand by exploiting self-adaptation as a means to automate the execution of A/B testing pipelines and on the other hand through support for splitting populations to focus testing leveraging machine learning.

5.3 Approach

We now present AutoPABS, the new approach to automate pipelines of A/B tests with support for population splits. We start with outlining the requirements for a solution. Then, we explain how we automate the execution of pipelines of A/B tests using self-adaptation. Next, we zoom in on splitting a population in a pipeline leveraging machine learning. Lastly, we present a concrete implementation of AutoPABS.

5.3.1 Requirements

The requirements for a solution to automate the execution of pipelines of A/B tests with support for population split are:

R1 To provide a specification for modeling pipelines of A/B tests;

R2 To design a conceptual architecture that automates the execution of the modeled pipelines of A/B tests;

R3 To provide a specification for modeling a population split in a pipeline of A/B tests;

R4 To design an extended conceptual architecture to support population splits when executing a pipeline of A/B tests.

As an additional requirement, **R5**, an infrastructure is required that implements the extended conceptual architecture.

5.3.2 Self-adaptation to Automate A/B Testing Pipelines

AutoPABS leverages the principles of self-adaptation [64, 223, 222, 361] to automate the execution of pipelines of A/B tests. AutoPABS adds a feedback loop [193, 376] (managing system) on top of a running system (managed system) that is responsible to deploy and execute a pipeline of A/B tests. AutoPABS assumes that the managed system is ”A/B testing-enabled” meaning, i.e., it is endowed with capabilities to deploy, monitor, and run A/B tests during operation.

We explain self-adaptation in AutoPABS in two steps. First, we present a specification to model pipelines of A/B tests (tackling requirement R1). Second, we present the conceptual architecture of AutoPABS focusing on the automated execution of pipelines of A/B tests (tackling requirement R2).

Specification to model pipelines of A/B tests. To support modeling an A/B testing pipeline, we put forward a simple specification. Figure 5.2 shows a visual notation of the different elements for a basic example of an A/B testing pipeline. Subsequently, we explain the specification of an A/B test, transition rules, and an A/B testing pipeline.

Specification A/B test

$$\textit{AB-test} = < \textit{Exp-length, AB-assignment, Hypothesis, \{AB-metrics\}, Stat-test} >$$

The experiment length of an A/B test (*Exp-length*) denotes the number of observations or the duration of the test required to complete the experiment. The A/B assignment (*AB-*

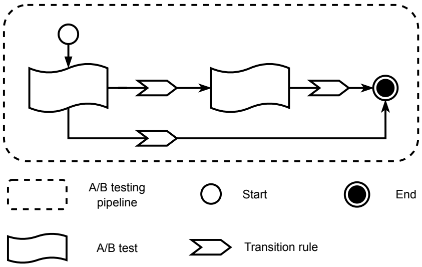


Figure 5.2: Visual notation of an A/B testing pipeline

assignment) specifies the proportions of the population that use variant A and variant B, respectively. The hypothesis (*Hypothesis*) is the supposition that is put forward before the A/B test is conducted. The A/B metrics (*{AB-metrics}*) are quantifiable measures used to validate the hypothesis. The statistical test (*Stat-test*) is used to test the hypothesis once the A/B test finishes (due to the number of observations or duration).

Specification transition rule

Trans-rule = < *Assoc-AB-test, Cond-stat, Subseq-AB-test* >

A transition rule applies to an A/B test, i.e., the associated A/B test (*Assoc-AB-test*) or the *End* element that ends the pipeline (see below). The conditional statement (*Cond-stat*) is a boolean expression on the outcome of the associated A/B test. The subsequent A/B test (*Subseq-AB-test*) is the next A/B test in the pipeline that should be started if the conditional statement is satisfied. Algorithm 1 describes the semantics of a transition rule given an A/B test and the result of the test.

Algorithm 1 Test the application of a transition rule.

```
procedure RULE-APPLIES(trans-rule, result, AB-test)
    return
        AB-test = trans-rule.Assoc-AB-test and
        trans-rule.Cond-stat.test(result) = true
end procedure
```

Specification of A/B testing pipeline

AB-test-pl = < *{AB-test}, {Trans-rule}, Start, End* >

An A/B testing pipeline consists of a set of A/B tests (*{AB-test}*) and a set of transition rules (*{Trans-rule}*). The start element (*Start*) points to the first A/B test in the pipeline, and the end element (*End*) marks the end of the pipeline execution. The end element is of the type A/B test. The set of transition rules defines the execution flow of the pipeline, based on the results of the executed A/B tests. It is the responsibility of the designer to create consistent transition rules that ensure a proper execution of the pipeline. Algorithm 2 describes the semantics of the automatic execution of A/B testing pipelines.

Algorithm 2 Automatically execute A/B testing pipelines.

```
 1: procedure ExecutePipeline(pipeline)
 2:     test $\leftarrow$ pipeline.Start
 3:     while test $\neq$ End do
 4:         res $\leftarrow$ Deploy(test)
 5:         next $\leftarrow$ End
 6:         for rule in pipeline.Trans-rule do
 7:             if Rule-Applies(rule, res, test) then
 8:                 next $\leftarrow$ rule.Subseq-AB-test
 9:                 break
10:             end if
11:         end for
12:         test $\leftarrow$ next
13:     end while
14: end procedure
```

The execution of the A/B testing pipelines starts with the initial A/B test on line 2. Until the End of the pipeline is not encountered (line 3), the current A/B test is deployed and the result of the A/B test is collected (line 4). The result of the A/B test is used to test the condition of the transition rules and to identify the next A/B test (lines 6-9). If the next A/B test is End, the execution of the A/B testing pipeline stops (as explained, we assume a consistent set of designed rules).

Conceptual architecture. We present now the conceptual architecture of AutoPABS. We start with the viewpoint of setting up and initiating a pipeline. Then we look at the viewpoint of the execution of a pipeline of A/B tests.

Architecture: Setting up and initiating a pipeline Figure 5.3 shows the architecture of AutoPABS with a focus on setting up and initiating a pipeline of A/B tests. The process is initiated by an operator. After deploying the A/B testing pipeline specification at the pipeline workflow logic (1), the operator triggers the managing system to initiate the pipeline (2). The start component then initializes the current A/B test with the first A/B test of the pipeline (3). Next, start requests the planner

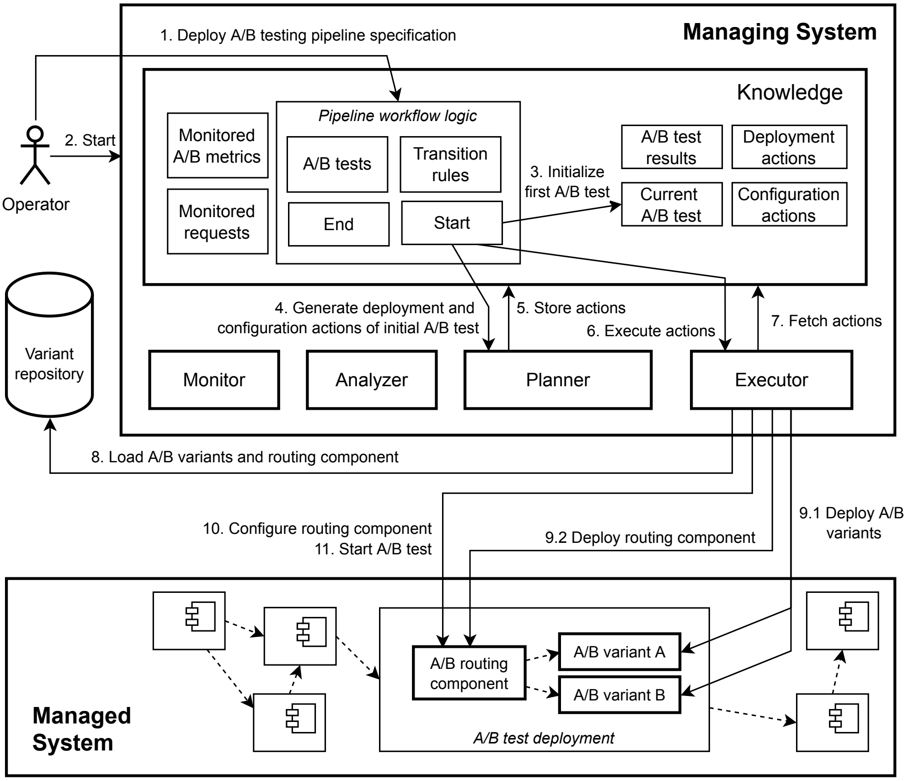


Figure 5.3: Architecture from the viewpoint of setting up and initiating an A/B testing pipeline

to generate the deployment and configuration actions to deploy the A/B variants and the routing component for the first A/B test (4). The planner stores the actions in the knowledge repository (5). Start then triggers the executor to execute these actions (6). To that end, the executor fetches the actions (7) and loads the A/B variants and the routing component from the variant repository (8). Then the executor deploys the A/B variants (9.1) and the A/B routing component (9.2). Finally, the executor uses the configuration actions to configure the A/B routing component (10). This completes the setup and initialization of the A/B testing pipeline. The executor can then start the first A/B test of the pipeline (11).

Architecture: Executing a pipeline Figure 5.4 shows the architecture of AutoPABS with a focus on the execution of an A/B testing pipeline. We assume that the A/B test on the left-hand side is currently in execution and that this test

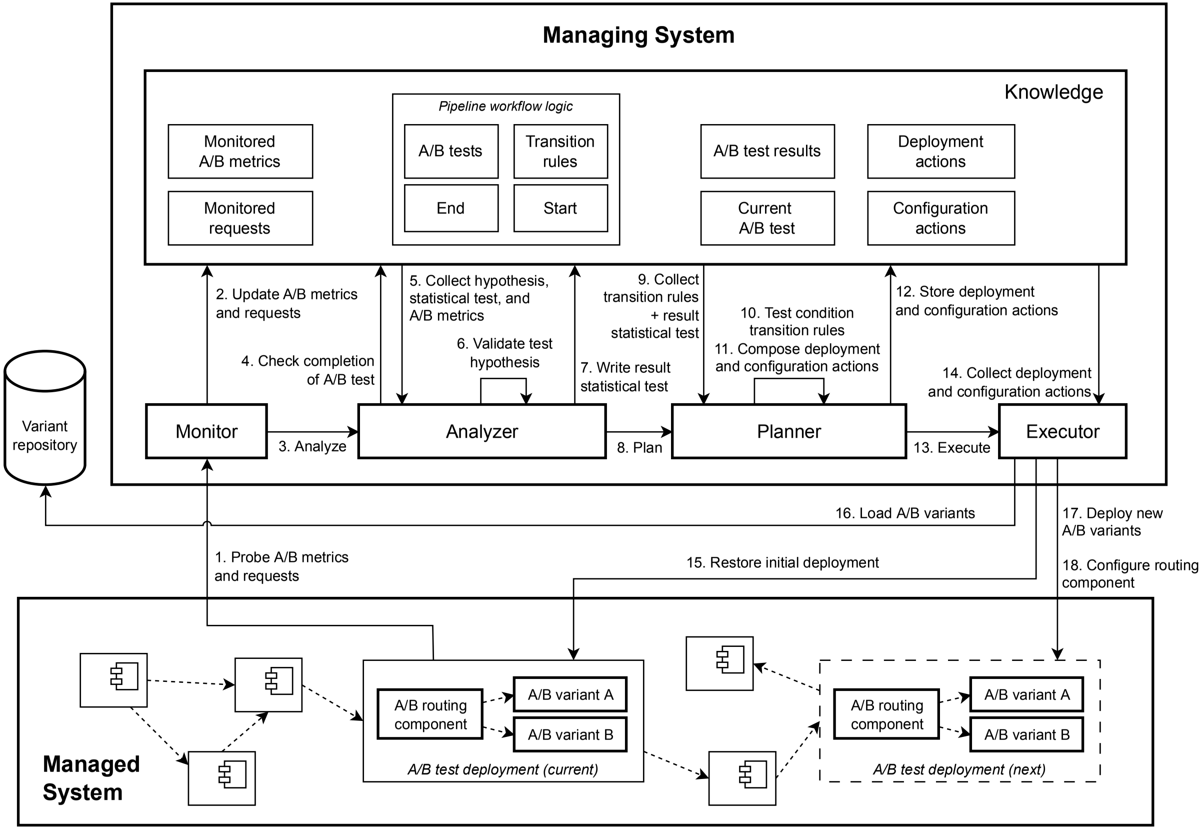


Figure 5.4: Architecture from the viewpoint of executing an A/B testing pipeline

uses the number of requests as the length of the experiment, while the A/B test on the right-hand side is the next A/B test. The monitor of the managing system starts with probing the status of the A/B metrics of the A/B test currently running and the number of requests invoked on the A/B variants (1); it then uses this data to update the knowledge repository of the managing system (2). Next, the monitor triggers the analyzer (3). The analyzer checks whether the number of invoked requests is sufficient to end the current A/B test (4). If this is the case, the process completes (not shown in Figure 5.4). Otherwise, the analyzer fetches the hypothesis, the statistical test, and the A/B metrics (5). The analyzer then tests the hypothesis (6), writes the result of the test to the knowledge repository (7), and triggers the planner (8). The planner collects the transition rules of the pipeline and the result of the statistical test (9) and tests the conditions of the transition rules (10). The planner then composes the deployment and configuration actions (11), stores the actions in the knowledge repository (12), and triggers the executor (13). The executor collects the deployment and configuration actions (14). It then restores the initial deployment of the components involved in the current A/B test (15). Finally, if the next step in the pipeline is the execution of a new test, the A/B variants are loaded (16) and deployed (17) and the routing component is configured for the new A/B test (18). Otherwise, if the next step in the pipeline is the end of the execution, the stakeholders are informed that the execution of the pipeline has been completed and that the results of the A/B tests are available (not shown in Figure 5.4).

5.3.3 Self-adaptation and Machine Learning to Split Populations

AutoPABS supports a split of the population in an A/B testing pipeline according to predefined criteria. In this chapter, we focus on segmenting a population based on properties or behaviors of users. This segmentation can then be used to target tailored A/B tests based on the appropriate property or type of user. AutoPABS leverages self-adaptation and machine learning techniques to support population splits. Segmenting the population offers an important benefit in terms of the efficiency of executing A/B testing pipelines: multiple parts of an A/B testing pipeline can be executed in parallel if the population segments assigned to each part of the pipeline are mutually exclusive, i.e., users are guaranteed to only be eligible for a single A/B testing pipeline.

We follow a similar structure as the section outlining the use of self-adaptation to automate the execution of pipelines to explain population splits in AutoPABS. We present a specification to model a population split (tackling requirement R3). Then, we present the conceptual architecture of AutoPABS focusing on population splits (tackling requirement R4).

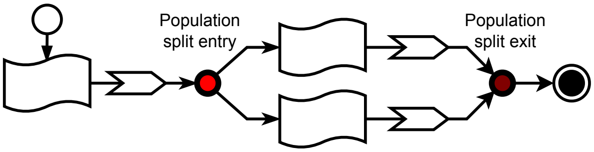


Figure 5.5: Visual notation population split in a simple pipeline

Specification to model pipelines of A/B tests. To support modeling a population split, we present a simple specification. Figure 5.5 shows a visual notation of a population split used in a basic example of an A/B testing pipeline.

We first explain the specification of a population split. Afterward, we extend the specification of an A/B testing pipeline with population splits.

Specification population split

Pop-split = < *Pop-split-entry, Pop-split-exit* >

A population split (*Pop-split*) consists of an entry (*Pop-split-entry*) and an exit (*Pop-split-exit*). We specify both parts now in detail, starting with the entry.

Pop-split-entry = < *Split-prop, {Sub-pipeline}, {Cond-stat}* >

Sub-pipeline = < *Subpl-ID, Start, {AB-test}, {Trans-rule}* >

The split property (*Split-prop*) defines the attribute on which the population is segmented. An example is the likelihood that a user makes a purchase on a website. In this example, machine learning could be used to predict the likelihood of making purchases based on the behavior of the user on the website. The population is then segmented and assigned to a list of sub-pipelines (*{Sub-pipeline}*) based on the split property. A sub-pipeline contains a unique identifier (*Subpl-ID*). Additionally, it consists of a set of A/B tests (*{AB-test}*) with a starting AB test (*Start*) as the first test of the sub-pipeline, and a set of transition rules ({Trans-rule}). Since these sub-pipelines can be executed in parallel, they should not interfere to ensure the satisfaction of the SUTVA [388, 160]. (i.e., not involve shared components). Ensuring this constraint is the responsibility of the designer of the pipeline. The assignment of population segments to tests depends on the satisfaction of the specified set of conditional statements *{Cond-stat}*, one per subsequent A/B test.

The exit of a population split is defined as:

Pop-split-exit = < *{Assoc-trans-rule}, Subseq-AB-test* >

An exit has an associated set of transition rules (*Assoc-trans-rule*) that correspond to the completion of the different sub-pipelines determined by the population split. After the exit of the population split, the A/B testing pipeline continues execution with its remaining part, i.e., the execution of the next A/B test or the execution ends (*Subseq-AB-test*).

Updated specification A/B testing pipeline Lastly, we update the specification for an A/B testing pipelines that accommodates for population splits:

AB-test-pl = < {AB-test}, {Trans-rule}, {Pop-split}, Start, End >

An A/B testing pipeline comprises next to a set of A/B tests and transition rules also a set of population splits (*{Pop-split}*) that enable the enclosed sub-pipelines to run in parallel.

Algorithm 3 describes the semantics of the application of a population split in A/B testing pipelines. We distinguish between the execution of a population split entry and a population split exit. The execution of a population split entry starts with instantiating a new knowledge component for each sub-pipeline (line 4). In each knowledge instance, the population split adds a specific routing configuration according to the population split property and the conditional statement of the sub-pipeline (line 5). Lastly, the population split entry starts the parallel execution of the sub-pipelines (line 7). The execution of a population split exit removes the knowledge instances for each sub-pipeline from the knowledge repository (line 14). Afterward, the A/B testing pipeline continues with the next activated A/B test in the pipeline (line 16).

Architecture with population split. Figure 5.6 shows the architecture of AutoPABS with a focus on population splits. We assume that the A/B test on the left in the managed system (denoted by *A/B test deployment (current)*) is in operation. Steps 1 through 8 remain identical to the steps outlined in Figure 5.4. Once the planner is triggered, it collects the transition rules, results of the statistical test, and population splits from the knowledge (9). The planner then tests the conditions of the transition rules (10) and, if one of the transition rules is satisfied, composes the deployment and configuration actions (11). If the transition rule results in a regular A/B test, the flow as described in Figure 5.4 continues. However, if a rule of a population split is satisfied, the planner prepares the knowledge and adds two knowledge instances; one for each sub-pipeline in the population split (12). Then, the planner stores the deployment and configuration actions in the knowledge (13), and triggers the executor (14). The executor collects the deployment and configuration actions from the knowledge (15). It then restores the initial deployment of the components involved with the A/B test (16). The executor then fetches the A/B variants for the sub-pipelines and the population split component (17). Next, it deploys and configures the population split component for the sub-pipelines (18). Finally, the executor deploys the new A/B variants for both

Algorithm 3 Apply a population split.

```
procedure ExecuteSplitEntry(split)
    entry ← split.Pop-split-entry
    for sub-pl, cond in (entry.Sub-pipeline,
                         entry.Cond-stat) do
        k ← Knowledge.addInstance(sub-pl.Subpl-ID)
        k.configureRouting(cond, entry.Split-prop)
    end for
    ExecutePipelines(entry.Sub-pipeline)
end procedure

procedure ExecuteSplitExit(split)
    pipelines ← split.Pop-split-entry.Sub-pipeline
    ids ← pipelines.Subpl-ID
    for id in ids do
        Knowledge.removeInstance(id)
    end for
    Deploy(split.Pop-split-exit.Subseq-AB-test)
end procedure
```

sub-pipelines (19) and configures both routing components (20). The sub-pipelines can then start executing in parallel.

5.3.4 Concrete Realization of the Conceptual Architecture

We implemented the conceptual architecture leveraging the SEAByTE [283] artifact that provides basic support for the automatic execution of pipelines of A/B tests applied to the domain of microservice-based systems (tackling R5). To implement the conceptual architecture of AutoPABS, we extended the blueprints of experiments, transition rules, and pipelines in SEAByTE and added a blueprint for a population split. Then we extended the implementation of the managing system and we added a population split component to SEAByTE. We focus here on the realization of the population split. For further details, we refer to the SEAByTE website.[1]

Blueprint population split with machine learning Listing 5.1 shows an example of the blueprint of a population split for the microservice-based application of SEAByTE.

[1] https://people.cs.kuleuven.be/danny.weyns/software/SEAByTE/

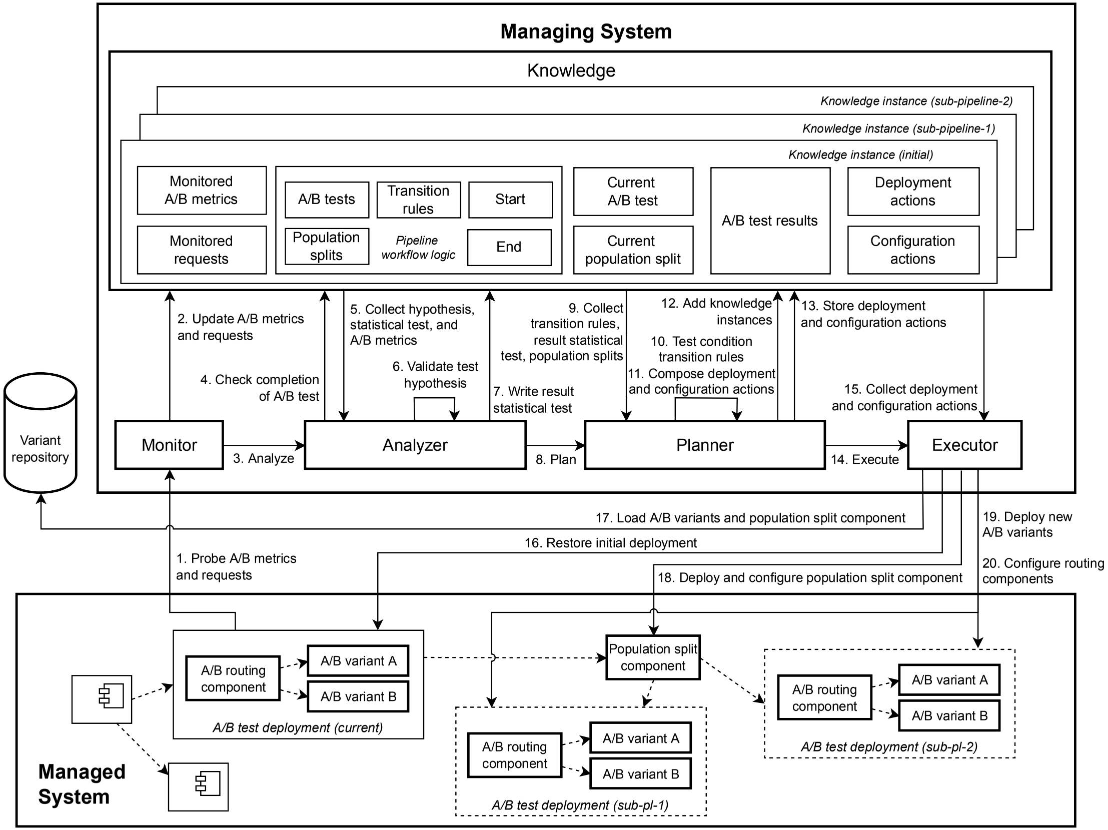

Managing System
Knowledge
Knowledge instance (sub-pipeline-2)
Knowledge instance (sub-pipeline-1)
Knowledge instance (initial)
Monitored A/B metrics
Monitored requests
A/B tests
Transition rules
Start
Population splits
Pipeline workflow logic
End
Current A/B test
Current population split
A/B test results
Deployment actions
Configuration actions
2. Update A/B metrics and requests
4. Check completion of A/B test
5. Collect hypothesis, statistical test, and A/B metrics
6. Validate test hypothesis
7. Write result statistical test
9. Collect transition rules, result statistical test, population splits
12. Add knowledge instances
10. Test condition transition rules
11. Compose deployment and configuration actions
13. Store deployment and configuration actions
15. Collect deployment and configuration actions
Variant repository
Monitor
3. Analyze
Analyzer
8. Plan
Planner
14. Execute
Executor
17. Load A/B variants and population split component
16. Restore initial deployment
1. Probe A/B metrics and requests
18. Deploy and configure population split component
19. Deploy new A/B variants
20. Configure routing components
A/B routing component
A/B variant A
A/B variant B
A/B test deployment (current)
Population split component
A/B routing component
A/B variant A
A/B variant B
A/B test deployment (sub-pl-1)
A/B routing component
A/B variant A
A/B variant B
A/B test deployment (sub-pl-2)
Managed System

Figure 5.6: Architecture from the viewpoint of executing an A/B testing pipeline with population split

```
{
  "name": "Population-split-purchases-prediction",
  "splitProperty": "purchase-likelihood",
  "pipelines": ["Review-pipeline", "Recommendation-pipeline"],
  "conditionalStatements": [{"==", 0}, {"==", 1}],
  "nextComponent": "end",
  "splitComponent": {
    "serviceName": "purchase-prediction-component",
    "imageName": "ml-purchase-filter"
  }
}
```

Listing 5.1: Example blueprint of a population split.

The elements in the blueprint of population splits are:

- name: The name of the population split component.
- splitProperty: The property on which the population will be segmented.
- pipelines: The name of the pipelines to be started, in *{Sub-pipeline}* of a population split entry. In the example, we consider two sub-pipelines.
- conditionalStatements: The conditions that determine which segments of the population will take part in the designated A/B testing pipelines, corresponding to *{Cond-stat}* in the population split entry. In the example, users classified with a purchase likelihood of 0 will take part in the review pipeline, while users with a likelihood of 1 will take part in the recommendation pipeline.
- nextComponent: The component that follows after completing all pipelines in the population split.
- splitComponent: The population split component that is responsible for exposing an API that classifies users on the provided split property. The population split component is deployed in docker with the given service name from the provided image name.

Realization of the population split component Figure 5.7 shows the population split component supported by the enhanced version of SEAByTE during deployment. Before deployment in the live system (not shown in Figure 5.7), a classification machine learning model is loaded into the population split component. Prior to this, the model is trained using historically labeled user data. In our example, the feedback loop was responsible for keeping track of historical data in the application, and using this data to train the machine-learning model. At runtime, user requests are collected by the population split API (1). The API invokes a query with the split property to the population divider (2). The population divider then uses the trained classification machine learning model to predict the classification of the user (3). Next, the population

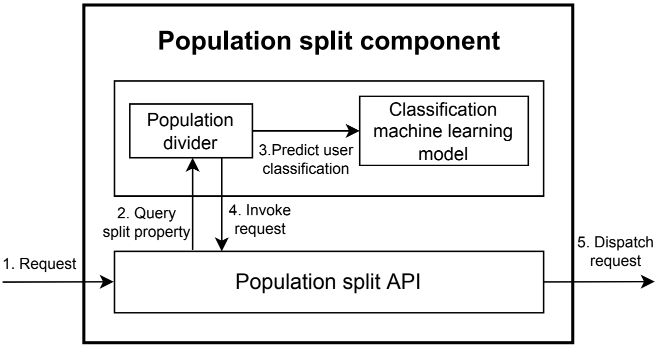


Figure 5.7: Architecture of the population split component

divider invokes the request for the predicted class using the population split API (4) that then dispatches the request to the sub-pipeline of that class (5).

5.4 Evaluation

We start with presenting the results of a short survey with experts on the usefulness of the notation and infrastructure of AutoPABS. Then we present the results of a series of tests that we performed to measure the gain obtained when using an automated A/B testing pipeline with a population split compared to a sequential pipeline. We also report the runtime overhead caused by a population split. For the tests, we use a scenario that we implemented in SEAByTE. We conclude the section with a discussion and analysis of threats to validity.

5.4.1 Evaluation questions

The evaluation aims at answering the following four evaluation questions:

EQ1: How do people knowledgeable in the topic appraise the usefulness of the notation of AutoPABS to model pipelines of A/B tests with population splits?

EQ2: How do people knowledgeable in the topic appraise the usefulness of the infrastructure of AutoPABS to execute pipelines of A/B tests with population splits?

EQ3: What is the reduction in the number of requests that we can obtain to get statistically significant results of A/B tests in pipelines with population splits compared to sequential pipelines?

Table 5.1: Metrics to answer the evaluation questions

Question	Metric	Description
EQ3	Reduction in requests	Difference in number of requests to obtain statistically significant results (from where $p \leq 0.05$) of an automatically executed pipeline of A/B tests with and without population split.
EQ4	Overhead	The time it takes (i) to train the machine learning model of the population split component, (ii) to deploy the population split component, (iii) to classify a request for a population split.

EQ4: How much overhead do population splits introduce before and during executing A/B testing pipelines?

5.4.2 Evaluation metrics

To answer EQ1 and EQ2 we use a single questionnaire where experts could express their appraisal for AutoPABS on a five-point Likert scale. To answer EQ3 and EQ4 we run experiments on a concrete system using the metrics of Table 5.1. The results report the median from 15 runs. All evaluation materials with results are available on the SEAByTE website.

5.4.3 Evaluation Instruments and Settings

Population and questionnaire for answering EQ1 and EQ2. We invited 32 experts to participate in the questionnaire and received 19 valid answers. Thirteen answers (68.4%) were from academics with practical experience (on average 2.29 years of experience in the software industry) and 6 answers (31.6%) were from practitioners (on average 9.58 years of experience in the software industry). The online questionnaire started with a brief introduction of AutoPABS. Then we asked the participants to answer the following questions:

Q1 How useful is automating A/B testing pipelines?

Q2 How useful is a population split in A/B testing?

Q3 How useful is the notation to specify A/B testing pipelines?

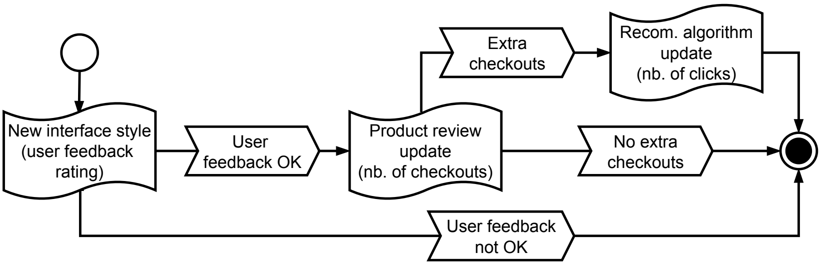

New interface style (user feedback rating)
User feedback OK
Product review update (nb. of checkouts)
Extra checkouts
Recom. algorithm update (nb. of clicks)
No extra checkouts
User feedback not OK

Figure 5.8: Pipeline for evaluating sequential runs of A/B tests

Q4 How useful is an implementation that supports running A/B testing pipelines automatically?

In addition for validity of the answers, we asked participants how familiar they are with self-adaptation and A/B testing.

Scenarios for answering EQ3 and EQ4. For the approach without population split we used the pipeline shown in Figure 5.8. The pipeline starts with an A/B test on a new style of a user interface of a web-store application. If the new style is favored by users, a new A/B test is launched that tests if users are more likely to purchase products when product reviews are presented at checkout, which may be an incentive for users to buy if they are hesitant to purchase at checkout. If this leads to more checkouts, a new A/B test is launched that evaluates a new version of the recommendation algorithm with the aim of serving better-targeted recommendations. The hypothesis is that the new algorithm is more effective at generating recommendations that result in more purchases.

For AutoPABS with a population split we used the pipeline shown in Figure 5.9. After successful completion of the A/B test of the new interface style (first A/B test of the pipeline), users are split between two pipelines according to their *likelihood of purchasing* something in the web store. To that end, the population split uses a classifier to make predictions about the likelihood of a user making a purchase. Users that are predicted to make a purchase take part in the recommendation A/B test and the others take part in the review A/B test. Since users belong to a single class, the review A/B test and the recommendation A/B test can run in parallel.

Data sets. We use publicly available datasets to (1) predict the likelihood of a user making a purchase in the web-store, and (2) to model user behavior in user profiles.

Customer propensity to purchase. The first dataset[2] consists of $455,401$ labeled data samples, each with a user identifier, 23 features describing actions taken by the user

[2] https://www.kaggle.com/datasets/benpowis/customer-propensity-to-purchase-data

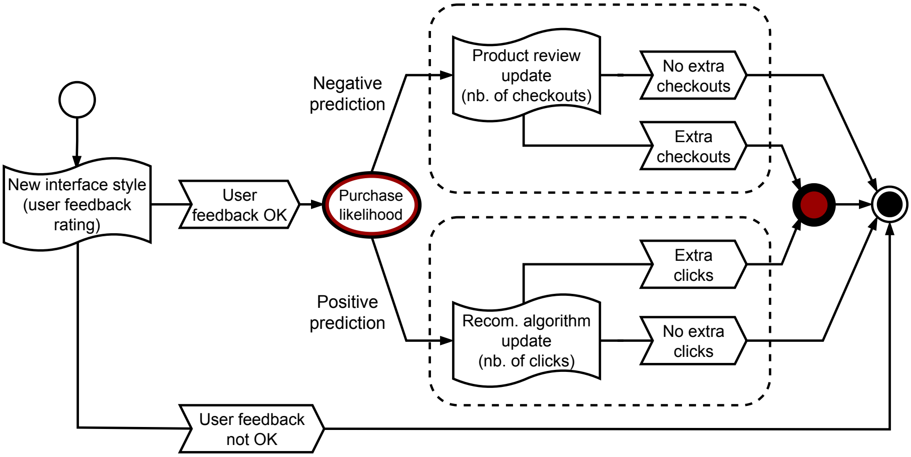


Figure 5.9: Pipeline for evaluating AutoPABS with population split

(e.g. requested information about a product) or characteristics of the user (e.g. the device used to visit the web-store), and a label about whether the user made a purchase in the web-store. Of all data samples, 4.2% are labeled positively, i.e. a customer that made a purchase. We used 25% of the data samples to train the machine learning model; the remaining 75% was used to model the user behavior in the web store.

User profiles. The second dataset[3] contains $10,000$ data samples of an A/B test about the revenue obtained from users. In variant A 1.605% of users make a purchase, while in variant B 1.435% of users make a purchase. The third dataset[4] contains $120,000$ data samples with clicks of users. Of all samples, 14.70% of users produce clicks in variant A, while 16.17% of users produce clicks in variant B. We designed the user profile using these data sets.

Test environment. We implemented the pipelines in the extended version of SEAByTE. The tests were run on a machine with a 2 x Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz processor and 16GB of RAM.

For the population split component, we used a Stochastic Gradient Descent (SGD) classifier. The classifier is trained by estimating the gradient of the loss for the training samples, and iteratively updating its model to minimize this loss. In our evaluation, we employed an implementation of the classifier from the scikit-learn library [269]. The test setup and a replication package are available at the SEAByTE website.

For the evaluation of the approaches for sequential- and parallel A/B testing pipelines

[3] https://www.kaggle.com/datasets/sergylog/ab-test-data

[4] https://www.kaggle.com/datasets/sergylog/ab-test-aggregated-data

we used custom-developed solutions for A/B test execution (incl. user assignment, A/B metric tracking, and hypothesis testing). To the best of our knowledge, existing A/B testing tools such as Split.io, VWO, or Convert[5] do not offer support for implementing a population split component as described in our approach, hence restricting us from evaluating the parallel A/B pipeline in these tools. To ensure consistency in the execution of both pipelines, we also chose to execute the sequential A/B testing pipeline using our custom-developed solution, exploiting the same code for running, monitoring, and analyzing the A/B tests.

5.4.4 Evaluation Results

Survey Results (EQ1 and EQ2) From the 19 valid answers[6], the average score for familiarity with self-adaptation was 4.11 and for familiarity with A/B testing was 3.21 on a Likert scale: 0 not familiar ... 5 an expert. These results show that the participants have the required knowledge to provide valid answers.

We obtained an average score of 4.16 [± 0.69] for the usefulness of automating A/B testing pipelines (Q1), while the score for the usefulness of population split was 4.21 [± 0.85] (Q2) both on a Likert scale: 0 not useful ... 5 highly important. This underpins the importance of the research problem.

EQ1: Usefulness notation For the usefulness of the notation of AutoPABS that supports modeling A/B testing pipelines with population splits (Q3), we obtained a score of 3.72 [± 0.75]. This result shows that the participants appraise the usefulness of the notation provided by AutoPABS.

EQ2: Usefulness infrastructure For the usefulness of AutoPABS's infrastructure to run A/B testing pipelines with population splits (Q2) we obtained a score of 4.21 [± 0.71]. These results show that the participants appraise the importance of the infrastructure provided by AutoPABS.

Results of the Tests (EQ3 and EQ4) We start with the results for the reduction in number of requests to get statistically significant results of A/B tests in pipelines with population splits compared to sequential pipelines. Then we look at the results for overhead caused by population splits.

[5] https://www.split.io/, https://vwo.com/, https://www.convert.com/

[6] We removed four additional answers from participants that expressed that they have no basic knowledge of either self-adaptation or A/B testing.

Table 5.2: Reduction number of required requests (EQ3)

Pipeline	**A/B test**	**Number of requests (until $p \leq 0.05$)**	**Total requests required**
Sequential	S_1 Recommendation update	112,000	112,000
	S_2 Review update	27,000	27,000
	Total (SUM S_1 + S_2)		**139,000**
Parallel	S_1 Recommendation update	1,000	24,038
	S_2 Review update	26,000	27,128
	Total (MAX P_1, P_2)		**27,128**

EQ3: Reduction in the required number of requests with population splits Since both A/B testing pipelines used in the evaluation (sequential Figure 5.8, and parallel Figure 5.9) start with a common A/B test that targets the whole population, the results of this A/B test are the same for both pipelines. Hence, we focus on the results of the two other A/B tests: the adjusted product review update A/B test and the adjusted recommendation algorithm A/B test. Table 5.2 summarizes the results (median values of number of requests over 15 runs).

For the review update A/B test, we observe a small difference in favor of the test with the population split. The review update A/B test in the sequential pipeline obtains a statistically significant result at $27,000$ requests, versus $26,000$ requests for the pipeline with population split. The results show that the population split component assigned 95.84% of the requests to the review update A/B test. The reduction in the number of requests to finish the A/B test with population split is 3.70%.

For the recommendation update A/B test, we observe a large improvement in favor of the population split. Sequential execution obtains a statistically significant result after $112,000$ requests. Parallel execution immediately reaches statistical significance after $1,000$ requests. The population split component assigned 4.16% of the requests to the recommendation update A/B test. The reduction in the number of requests for the recommendation update A/B test is 99.11%. This very high number shows that the machine learning component is able to separate the two classes of users very well. The requests invoked on the recommendation update A/B test ensure that the test quickly obtains a statistically significant result. Figure 5.10 illustrates the progress of the p-values for the experiment of the recommendation update A/B test.

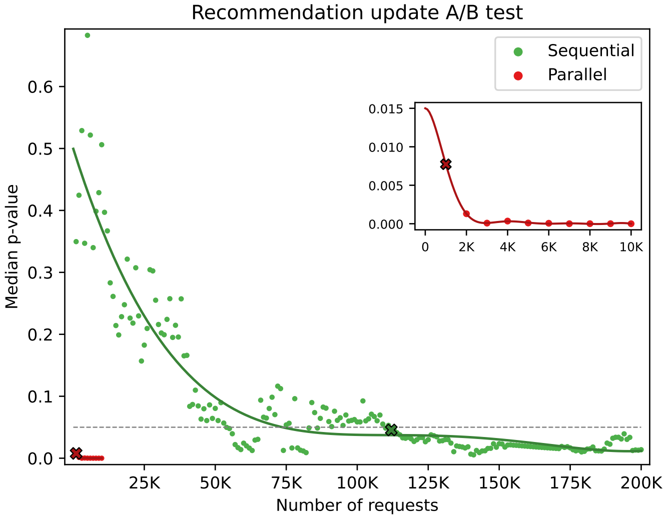


Figure 5.10: Median p-values in the recommendation update A/B test for the sequential and parallel pipelines (detail top right)[7]

Lastly, we look at the total number of requests required to complete the two tests with sequential and parallel execution. The execution with the sequential pipeline requires a total of $139,000$ requests to finish the execution of the two A/B tests with statistically significant results, i.e., the sum of $27,000$ and $112,000$ for the review update and recommendation A/B test, respectively. The execution of the pipeline with the population split finished after 27,128 requests in total, i.e., the total number of requests required to obtain statistically relevant results (26,000 and 1,000). Figure 5.11 illustrates the progress of the p-values over the requests to complete the tests. The overall gain in required requests with population split is 80.48%. We conclude that splitting a population based on the specific behavior of users realizes a significant improvement in required requests to complete the tests with significant results.

EQ4: Overhead introduced by population splits Table 5.3 provides an overview of the overhead introduced by a population split during preparation and operation.

Preparing the population split component consists of two steps: training the learning model and deploying the component. To train the machine learning model, historically labeled data is used. In the evaluation setting, this data was derived from the data sets we used. Training the machine learning model took on average 324 ms. The deployment

[7] The regression lines denote a polynomial fit over the data.

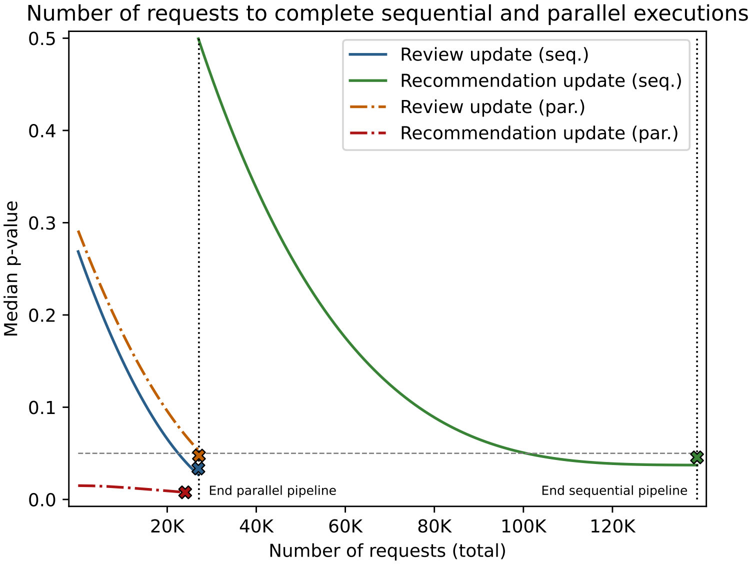


Figure 5.11: Progress of sequential and parallel execution for both the review update and recommendation update A/B tests[7]

Table 5.3: Overhead population split component (EQ4)

Overhead type	**Timing**	**Overhead (average, in msec)**
Training classifier	Offline	324
Component deployment	Deployment	6433
Prediction user class	Runtime	0.3

time of the population split component heavily depends on its implementation. In the case of SEAByTE, we use Docker to create and start a container that contains the population split component as a micro-service. The creation and startup took on average 1433 ms. In addition, Docker waits 5 sec (fixed) to check that the container is healthy. This resulted in a total average deployment time of 6433 msec. This overhead is not relevant compared to the time it takes to run A/B tests in practice.

During operation, the population split component classifies the population (e.g., purchasing or non-purchasing) using the trained classifier model. Predicting the class of a user took less than a millisecond (0.3 msec). This time is also negligible compared to the time it takes to run A/B tests in practice.

5.4.5 Discussion and Threats to Validity

Discussion We start with critical reflections on the tests.

- The significant increase in efficiency with a population split results from i) the ability to run the review and recommendation updates in parallel, and ii) the population split allows for targeted A/B testing to relevant segments of the population.
- The performance and accuracy of the machine learning model affect the result of the method: a model that makes bad predictions will divide a population wrongly thus affecting the results of the A/B tests. The model used in the evaluation performed very well, demonstrating the benefits of population splits for targeted A/B tests.
- The introduction of a population split introduces additional latency to the processing of requests. This extra latency can be detrimental to the user experience as noted by practitioners at Booking.com [25]. Hence, the designer needs to ensure that the time the machine learning model takes to produce predictions is acceptable to the users.
- Besides the evaluation scenario of this chapter, the population split component could also be used to detect undesirable outcomes of A/B tests early in specific population segments. In case an A/B test on a population segment produces regressive results, practitioners can specify that the A/B pipeline should shut down prematurely. Otherwise, A/B testing can continue on the other population segments, if desired.
- Creating a labeled dataset to train the classifier can also carry a substantial cost. We leave delving into this topic more comprehensively for future work.
- In the evaluation, we split the population into binary segments. However, AutoPABS does not impose this limitation. Future work could explore dividing

the population into more than two segments, or explore the use of unsupervised learning to split the population to avoid the need for labeled datasets.

Threats to validity We discuss construct and external validity threats of both the survey and the tests.

Construct validity The questionnaire probed the usefulness of automating A/B testing pipelines and population split in general and the support offered by AutoPABS in particular. Since we used closed questions, the participants were not able to provide nuances in their answers. Moreover, we provided only a brief introduction to AutoPABS, so the participants may not fully grasp the usefulness of the notation and infrastructure. We acknowledge that the validity of the small survey may be limited. However, we believe that the results provide a first good indication. To obtain deeper insight, additional studies are needed where participants effectively use the notation and infrastructure.

To measure the reduction in requests in the tests, we defined statistical significance from the point where $p \leq 0.05$ of the experiments of A/B tests. We used the median values over 15 runs to account for stochasticity in the data. The number 15 was empirically determined and may differ for other settings.

Internal validity To evaluate AutoPABS, we used publicly available datasets to model the behavior of users in the web store. However, limited information is available about the origin of the datasets (as mentioned by others [343]), raising a threat to the internal validity of the results. To fully mitigate this threat, an industrial case study should be conducted.

External validity The questionnaire only involved 19 participants with mixed knowledge and experience in self-adaptation and A/B testing. A more extensive survey and more participants would enhance the generalization of the results.

Since we only evaluated the approach for one concrete scenario in the context of a web store, we cannot make general claims about the applicability of the approach in different contexts. We anticipate that the technology and domain used for the evaluation are particularly relevant for contemporary software systems. In addition, we used external data sets to avoid bias. Lastly, we also provide a replication package [282] that is available for other researchers to replicate the results.

To answer EQ4, we measured (1) the time it took to train the machine learning model used in the population split component and (2) the time it took to deploy the component and use the component at runtime. We acknowledge that these measurements depend on the algorithms and technology used.

5.5 Conclusion and Future Work

Leveraging self-adaptation and machine learning, we presented AutoPABS, a new approach to automating the execution of pipelines of A/B tests with support for splitting populations. We specified the elements of AutoPABS and based on that presented a conceptual architecture. We instantiated this architecture extending the SEAByTE artifact. A small survey underpins the relevance of the approach and its usefulness. Test results on a realistic micro-service application show that AutoPABS accelerates the identification of statistically significant results of the A/B tests in the required number of requests with 80.48% compared to traditional sequential tests on the general population. In future work, we plan to study the identification of user groups without having access to labeled data leveraging unsupervised learning. This opens possibilities of automatically setting up A/B tests by experimenting with the target group of the A/B tests, without explicitly specifying the complete A/B tests in the pipeline. We also plan to investigate ways of incorporating and potentially altering the A/A testing process in the approach. Lastly, we aim to provide a full-fledged tool for specifying A/B testing pipelines with AutoPABS along with a framework to execute the pipelines that can be tailored to the domain and needs at hand.

Chapter 6

SEAByTE: A Self-adaptive Micro-service System Artifact for Automating A/B Testing

Publication details. This chapter is based entirely on a conference artifact publication in the International Symposium on Software Engineering for Self-Adaptive Systems (SEAMS) [283].

Personal contributions. Conceptualization (80%), Methodology (70%), Software (100%), Validation (70%), Formal analysis and interpretation results (70%), Writing (50%), Visualization (80%).

Positioning. Micro-services are a common architectural approach to software development today. An indispensable tool for evolving micro-service systems is A/B testing. In A/B testing, two variants, A and B, are applied in an experimental setting. By measuring the outcome of an evaluation criterion, developers can make evidence-based decisions to guide the evolution of their software. In this chapter, we contribute a novel artifact that aims at enhancing the automation of an experimentation pipeline of a micro-service system relying on the principles of self-adaptation. Concretely, we propose SEAByTE, an experimental framework for testing novel self-adaptation solutions to enhance the automation of continuous A/B testing of a micro-service based system. We illustrate the use of the SEAByTE artifact with a concrete example.

6.1 Introduction

Micro-services are nowadays a commonly used architectural approach to software development [133]. A micro-service architecture comprises small independent services that communicate over well-defined APIs. These services are usually owned by small, self-contained teams. Since each service performs a single function and runs independently, services can be easily updated, deployed, and scaled to meet changing demands. As such, a micro-service architecture naturally supports continuous deployment (CD) [185]. CD is based on the principles of agile development [90] and DevOps [254] and leverages continuous integration (CI) [252] that automates tasks such as compiling code, running tests, and building deployment packages. Among the benefits of CI/CD are rapid innovation, shorter time-to-market, increased customer satisfaction, continuous feedback, and improved developer productivity.

One of the key concerns of CI/CD is increasing the agility of development teams in terms of testing, updating, maintaining and deploying software. Understanding user engagement and satisfaction of product features or variants plays an important role in this. A/B testing, also called bucket testing or controlled experimentation [202, 297], offers a solution to this concern: comparing two variants, A and B, and test if the statistical distribution of a property of A is different from that of B [203]. By focusing on different properties in an experimental setting and systematically measuring the outcome, developers can make evidence-based decisions to guide software evolution. This is especially useful for (micro-)service based software with large user volumes [297]. Conducting experiments in iterations is known as continuous experimentation (CE) [122]. Recent studies highlight the need for enhanced automation of continuous experimentation as one of the key challenges in this area [296, 297].

In this chapter, we contribute a novel artifact that aims at enhancing the automation of continuous experimentation of a micro-service system using the principles of self-adaptation. In particular, we propose SEAByTE, an experimental framework that can be used for testing novel self-adaptation solutions to enhance the automation of continuous A/B testing of a micro-service based system.

The remainder of this chapter is structured as follows. In Section 6.2, we summarize the principles of micro-services and A/B testing and position SEAByTE in the landscape of self-adaptive system artifacts. Section 6.3 presents the artifact with scenarios. Section 6.4 illustrates the application of the artifact for one of the scenarios. Section 6.5 discusses the applicability of the artifact, Section 6.6 discusses future research directions, and Section 6.7 wraps up.

6.2 Background and Positioning of the Artifact

We briefly introduce the basics of micro-services and A/B testing, and then highlight how SEAByTE complements the existing artifacts for engineering self-adaptive systems.

6.2.1 Micro-services

A micro-service is a small independent piece of software that performs a single function (i.e., a business capability) and communicates with other micro-services via well-defined interfaces [95]. Each service in a micro-services architecture can be developed, deployed, operated, and scaled independently, without affecting other services. Services do not need to share any code with other services, hence they promote decentralized governance. When a change of a certain part of a micro-service based application is needed, only the related service(s) can be modified and redeployed without the need to modify and redeploy the entire application.

Key players that migrated towards a micro-service architecture include Amazon, Netflix, eBay, and Twitter [34, 33]. Yet, micro-service architecture also comes with challenges, such as determining the right size of micro-services, front-end integration, and failure management (need for self-healing) [181].

6.2.2 A/B Testing

A/B testing is a systematic approach to compare the use of two versions of a system and determining which of the two variants is preferred according to some criterion. More specifically, an A/B test consists of a randomized controlled experiment where experimental units (e.g., users) are assigned to one of two variants, called A and B, that are expected to influence some metric of interest. This metric, the overall evaluation criterion, provides a quantitative measure of the experiment's objective (in the form of a hypothesis). To compare the two variants, statistical hypothesis testing is commonly used. The choice of the tests (and their power) depends on the (assumed) distribution of the data. For instance, for Gaussian distributed data the unpaired t-test can be used. If the distribution is unknown, the Mann-Whitney U test can be used. A key aspect is the randomization applied to experimental units to map them to variants. Proper randomization is important to ensure that the populations are assigned to the different variants are statistically similar such that causal effects can be determined with high probability. Prerequisites for A/B testing are: (1) experimental units can be assigned to different variants with no (or little) interference; (2) there are enough experimental units (e.g., users), usually thousands, (3) the key metrics are agreed upon and can be practically evaluated, (4) experimental units can be easily assigned to variants (e.g., server-side software is much easier to change than client-side). Continuous experimentation refers to pipelines of experiments. Often, the analysis results may trigger the need for additional (analysis) and follow-up experiments. Setting up such pipelines is a challenge for current experimentation platforms, see e.g., [164].

A/B testing has been extensively used, ranging from user interface element testing, testing product pricing, evaluating personalized recommendations, testing product features, and more generally evaluating the impact of changes made to software products and services. Online controlled experiments are today common practice and are heavily used in practice [164].

Multi-variant testing is similar to A/B testing, but the tests then use more than two versions at the same time. Yet, the current focus of SEAByTE is on two-variant testing (A/B).

6.2.3 Positioning of the Artifact

At the time of writing, the community of engineering self-adaptive systems has produced 27 artifacts. According to the SEAMS artifacts website[1,2] these artifacts provide "examples, challenge problems, and solutions that the community can use to motivate research, exhibit and evaluate solutions and techniques, and compare results."

Of the 27 artifacts, only three have a specific focus on service-based systems: Znn.com, TAS, and SWIM. Znn.com [66] is centered on a web server system that provides a simplified news site. The testing environment simulates the slash-dot effect which are periods of abnormally high traffic that overload the system. TAS [367] is an exemplar of a service-based system (SBS). SBSs are widely used in e-commerce, online banking, e-health, and many other applications. In these systems, services offered by third-party providers are dynamically composed into workflows delivering complex functionality. SBSs increasingly rely on self-adaptation to cope with the uncertainties associated with third-party services, as the loose coupling of services makes online reconfiguration feasible. SWIM [259] is an exemplar for the evaluation and comparison of self-adaptation approaches for Web applications. The exemplar simulates a web application (simulating a 60-server cluster subject to millions of requests) that can be used as a target system with an external adaptation manager interacting with it through its TCP-based interface.

While the existing artifacts produced by the community aim at supporting research on novel approaches for *engineering self-adaptive systems*, SEAByTE takes a different angle and provides an artifact that aims at supporting research on novel approaches

[1] https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/

[2] ZENODO: https://zenodo.org/communities/seams/?page=1&size=20

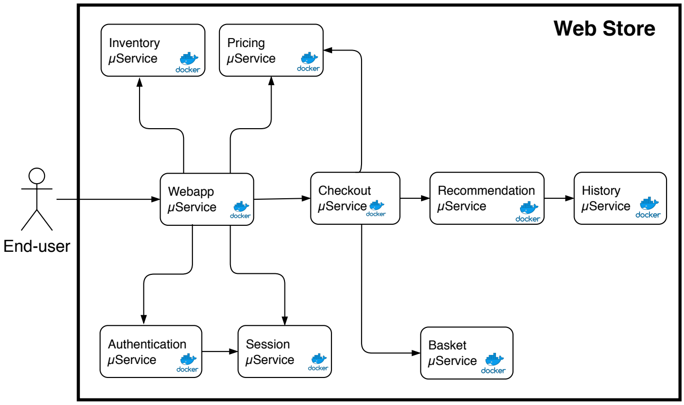


Figure 6.1: Architecture of the Web Store Application.

for *a key task of software engineers using self-adaptation*, in particular enhancing the automation of evolution with A/B testing. This aligns with recent initiatives of other researchers such as Self-Adaptation 2.0 [42] that argues for an equal-to-equal relationship between self-adaptation and AI, benefiting one another. Another example is discussed in [55] where self-adaptation is applied to deal with degraded machine-learning components to maintain system utility.

6.3 SEAByTE

SEAByTE provides an Internet Web Store composed of multiple micro-services. Figure 6.1 shows the architecture of the Web Store of SEAByTE. The Web Store serves end-users that perform purchases via the website of the store. When an end-user invokes a request, the Webapp service will authenticate the user, start a session, retrieve the price of the listed product, and update the inventory. Then the checkout service will be invoked that gives an overview of the products present in the basket of the user, provides recommendations for the user, and finally adds the purchase to the history of the user. This closes the session. Practically, the web-services are implemented using the Java Spring framework, with each service running in a separate Docker container. The implementation of the Web Store artifact, an installation and usage guide, and a concrete example are available at the SEAByTE website [282].

6.3.1 Experimental Pipeline

SEAByTE aims to enhance the automation of the evolution of the micro-service system using a series of A/B tests. Central to this automation is an experimental pipeline that comprises two basic elements: experiments and transition rules. The artifact provides templates to define both experimental pipelines and transition rules.

Experiments

An experiment comprises:

- *ID*: identifier experiment
- *variantA*: the first variant
- *variantB*: the second variant
- *userProfile*: the profile specifying end-user behaviors
- *ABAssignment*: mapping of end-users to variants
- *samples*: the number of samples used for the test
- *metrics*: the set of metrics
- *statisticalTest*
  - *hypothesis*: the hypothesis of the experiment
  - *pValue*: the p-value for the test[3]
  - *type*: the type of statistical test of the experiment
  - *result*: the result variable of the experiment

Listing 6.1 shows an example of an experiment with ID “Upgrade v1.0.0 - v1.1.0” for the artifact.

```
{
  "Upgrade v1.0.0 - v1.1.0": {
    "variantA": "ws-recommendation-service:1.0.0",
    "variantB": "ws-recommendation-service:1.1.0",
    "userProfile": "Standard",
    "ABAssignment": {
      "weightA": 50,
      "weightB": 50
    },
```

[3] The specified p-value decides, after collecting the specified number of samples for both variants, if the test can be rejected based on the observed p-value. The result of the test will either be ’reject’ or ’inconclusive’.

```
    "samples": 20000,
    "metrics": ["ResponseTime_A", "ResponseTime_B"],
    "statisticalTest": {
        "hypothesis": "ResponseTime_A == ResponseTime_B",
        "pValue": 0.025,
        "type": "welsh’s t-test",
        "resultingVariable": "result-wt-test"
    }
  }
}
```

Listing 6.1: Example specification of experiment in SEAByTE.

This experiment compares the performance of two variants of the Web Store that implement different versions of the recommendation service, see Figure 6.3. Version 1.0.0 of the recommendation service uses the history to provide recommendations to the user, while version 1.1.0 exploits also information about the current purchase to provide recommendations. The experiment uses a user profile that generates user requests. The requests are randomly assigned to the two versions, 50% each. The experiment tracks the requests with their response time. The statistical test compares the mean values of the response times of the invocations of both versions using a Welsh’s t-test [360]. The artifact uses the Apache Commons Mathematics Library for statistical testing [69]. The result of the test (accept or reject the hypothesis) is recorded in the resulting variable.

Transition Rules

A transition rule comprises:

- *ID*: identifier of the rule
- *fromExperiment*: identifier current experiment
- *toExperiment*: identifier of the next experiment
- *conditions*: the conditions to make a transition from the current experiment to the next experiment (that depends on the result of the current experiment).

to-experiment = *"end"* is a reserved value that indicates the end of an experimental pipeline and returns the control to the operator. Empty *conditions* indicate that the transition from the current to the next experiment is taken unconditionally.

Listing 6.2 shows an example of the specification of a transition rule for the artifact.

```
{
  "Performance OK": {
    "fromExperiment": "Upgrade v1.0.0 - v1.1.0",
    "toExperiment": "Clicks v1.0.0 - v1.1.0",
    "conditions": [{
       "leftOperand": "result-wt-test",
       "operator": "!=",
       "rightOperand": "reject"
    }]
  }
}
```

Listing 6.2: Example specification of a transition rule.

This transition rule is applied once the Upgrade v1.0.0 - v1.1.0 experiment is finished. It checks whether the result of the test was accepted, and if so, it starts the Clicks v1.0.0 - v1.1.0 experiment.

Experimental Pipeline

Finally, an experimental pipeline comprises the following:

- setup: identifier of the setup for the pipeline[4]
- start: the identifier of the experiment that starts the pipeline
- experiments: the set of experiment identifiers
- rules: a set of transition rule identifiers

Figure 6.2 shows a graphical representation of an example of an experimental pipeline, and Listing 6.3 shows how this pipeline is specified for the artifact.

The experiment starts with preparing the Web Store according to the specified setup. Then, the Upgrade v1.0.0 - v1.1.0 experiment that checks the performance overhead in response time that is generated by the upgrade of the Web Store to version v1.1.0 is executed. If the performance is acceptable (i.e., version 1.1.0 of the Web Store generates no significant overhead in response time compared to version 1.0.0), the transition via the rule “Performance OK” is taken and the pipeline starts the Clicks v1.0.0 - v1.1.0 experiment. Otherwise, the transition via the rule “Performance Overhead” is taken and the execution of the pipeline ends. The Clicks experiment checks whether users perform a significant number of extra clicks on recommendations for version 1.1.0 compared to version 1.0.0. If this is not the case, the transition “No Extra Clicks” is taken and the execution of the pipeline ends. Otherwise, the Purchases v1.0.0 - v1.1.0 experiment is started via the “Extra Clicks” rule. This last experiment compares the purchase behavior of the user with both variants and reports the result to the operator, completing

[4] A setup comprises the necessary steps to be taken to get the managed system ready for conducting experiments (further explained in Section 6.3.2).

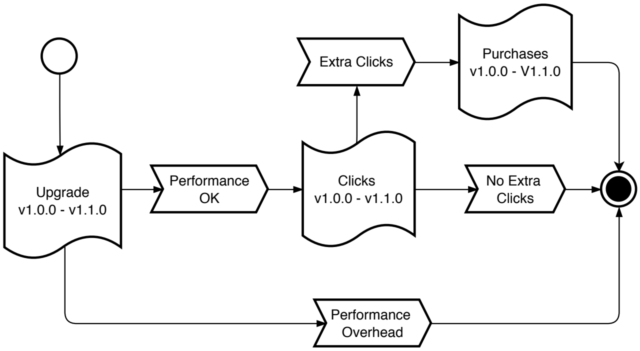


Figure 6.2: Example of an experimental pipeline.

the execution of the experimental pipeline. Based on the results the stakeholders can then decide whether or not to upgrade the Web Store, or set up additional experiments if needed.

```
{
  "setup": "Recommendation_upgrade",
  "start": "Upgrade v1.0.0 - v1.1.0",
  "experiments": [
    "Upgrade v1.0.0 - v1.1.0",
    "Clicks v1.0.0 - v1.1.0",
    "Purchases v1.0.0 - v1.1.0"
  ],
  "rules": [
    "Performance OK",
    "Performance Overhead",
    "Extra Clicks",
    "No Extra Clicks"
  ]
}
```

Listing 6.3: Example specification of experimental pipeline.

6.3.2 Architecture SEAByTE

SEAByTE adds a MAPE-K based feedback loop [193, 376] on top of the Web Store application that supports the automatic execution of an experimental pipeline. Figure 6.3 shows the architecture of SEAByTE illustrated for a scenario with two variants of the Recommendation service, A and B. We explain the main components now.

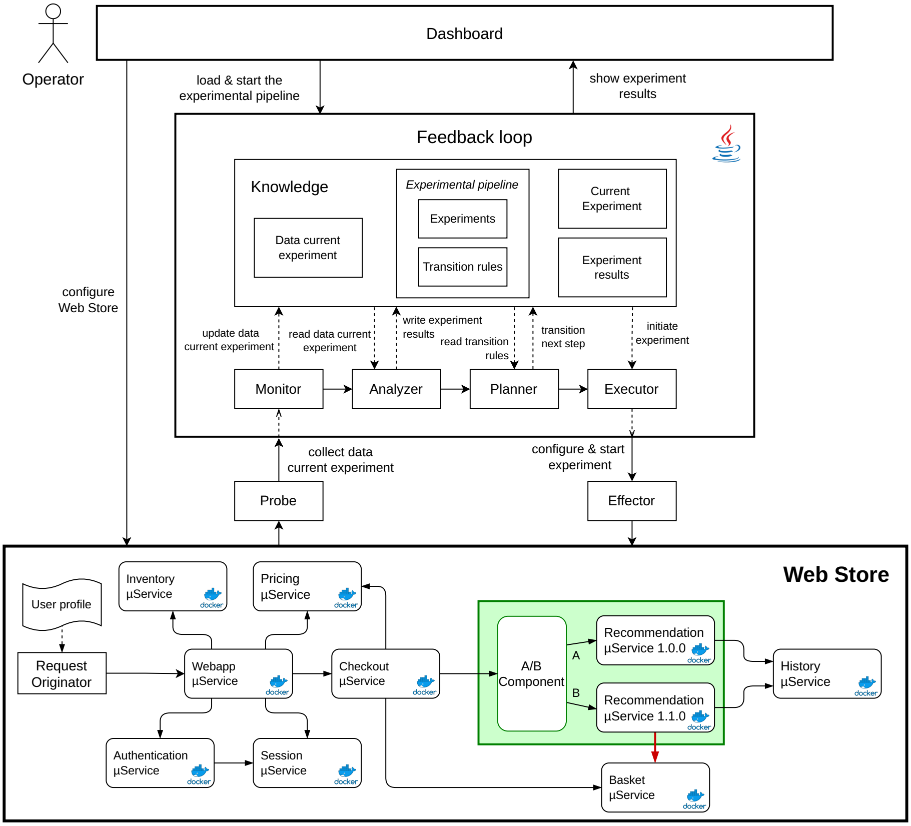


Figure 6.3: Architecture of SEAByTE.

User Profile

A user profile represents end-users of the Web Store; it defines the behavior of the users. Behavior refers to the actions taken by users and includes both the invocation of requests and feedback provided by users based on the completed requests. SEAByTE provides a template to specify user profiles. Listing 6.4 shows an example instance of the profile template.

The example shows the profile for a standard user that specifies different aspects of the behavior of the users when visiting the website of the store: purchasing goods, browsing the store, exploring recommendations, and their effect. A profile can include different types of users. User profiles can be derived from different sources such as domain knowledge, historical data, or based on tests. The request originator component exploits the user profile to generate the requests for the web store according to the specified parameters in the profile.

```
{
  "user-profile-regular" : {
    "Standard": {
      "count": 1000,
      "mean-seconds-between-request": 15,
      "probability-purchase": 0.25,
      "recommendation-click-probability": 0.2,
      "recommendation-purchase-probability": 0.05,
      "bonus-recommendation-click-B": 0.1,
      "bonus-recommendation-purchase-B": 0.05
    },
    "frivolous": {
      ...
    }
  }
}
```

Listing 6.4: Example specification of a user profile.

A/B Component

The A/B component determines the test setup with A and B variants and manages the routing of invocations to the two variants. The A/B test setting will be configured before the execution of the experimental pipeline starts (via the dashboard, see below). The concrete routing to the A and B variants will be configured for each concrete experiment (the responsibility of the effector, see below).

Probe and Effector

The probe is responsible for collecting data of experiments, while the effector is responsible for setting up the web store and managing the AB component. Listing 6.5 shows a basic API of the probe and effector.

```
Probe
  + List<URLRequest> getRequestHistory(String ABName, String variant)

Effector
  + void clearABComponentHistory(String ABName)
  + void setABRouting(String ABName, int a, int b)
  + void deploySetup(String setupName)
  + void removeSetup(String setupName)
```

Listing 6.5: The provided API for the probe and effector.

The probe provides a method to collect the history of the requests of the active experiment of the service invocations per variant. A request contains information about the response time, the requested URL, and the client ID. The effector enables setting up an experiment, including configuring the A and B setting and the routing for the variants, and clearing the history of the AB component.

Feedback Loop

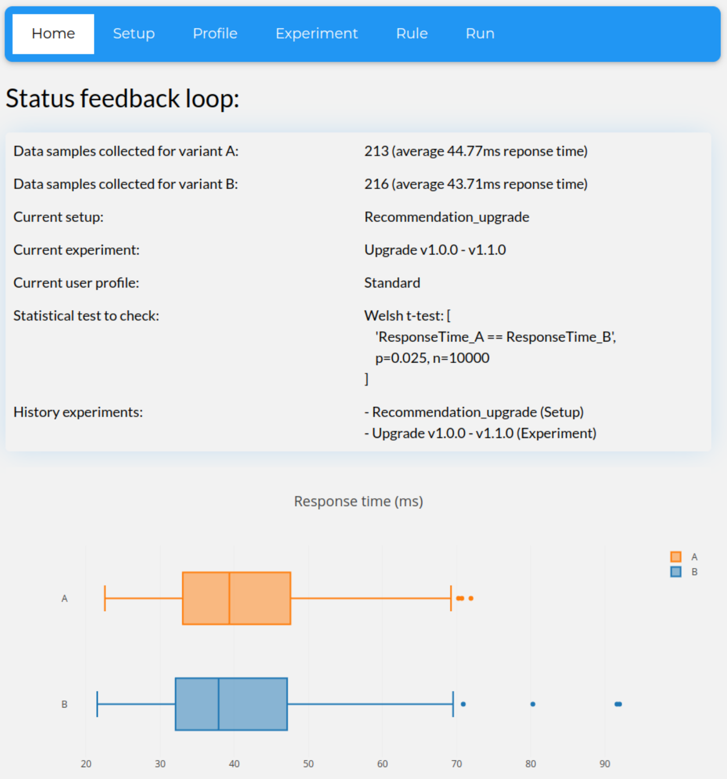
The feedback loop is responsible for executing an experimental pipeline, see Figure 6.3. The knowledge comprises a specification of the pipeline with the experiments and transition rules, the configuration of the current experiment, a repository to store data of the current experiment, and the experiment results. The monitor collects the data of the running experiment and updates the knowledge repository accordingly. When the experiment is completed, the analyzer applies the statistical test and writes the experiment result to the knowledge. The planner then applies the transition rules to the result of the last experiment and determines the next step in the experimental pipeline. Based on that, the executor initiates the next experiment, or alternatively, the experimental pipeline ends. The artifact provides abstract implementations of the feedback loop elements with an example.

Dashboard

The dashboard gives the operator access to the test environment, enabling: (1) configuration of the Web Store to run an experimental pipeline, (2) loading and starting an experimental pipeline, (3) monitoring the execution of an experimental pipeline, and (4) showing the results. Figure 6.4 shows an excerpt of the dashboard to monitor the progress of the execution of an experimental pipeline. The box plot at the bottom shows a live representation of the response times for versions A and B.

6.3.3 Test Scenarios

Table 6.1 shows the test scenarios supported by SEAByTE. Each scenario introduces increasingly difficult challenges. The challenges focus on three key aspects of A/B testing: system upgrades (S1 and S2), price setting (S3), and segmentation (S4 and S5). The artifact provides a full specification for concrete instances of scenarios S1 and S2, and a basic implementation supporting the other scenarios.



Status feedback loop:

Data samples collected for variant A:	213 (average 44.77ms reponse time)
Data samples collected for variant B:	216 (average 43.71ms reponse time)
Current setup:	Recommendation_upgrade
Current experiment:	Upgrade v1.0.0 - v1.1.0
Current user profile:	Standard
Statistical test to check:	Welsh t-test: ['ResponseTime_A == ResponseTime_B', p=0.025, n=10000]
History experiments:	- Recommendation_upgrade (Setup) - Upgrade v1.0.0 - v1.1.0 (Experiment)

Figure 6.4: Excerpt of the dashboard of SEAByTE.

6.4 Experimentation with the Artifact

6.4.1 Workflow to use the artifact

Using the artifact comprises the following steps:

- Specify the experimental pipeline with the experiments and transition rules
- Configure the Web Store with the variants to be tested
- Configure the feedback loop to prepare the execution of the experimental pipeline
- Load the experimental pipeline via the dashboard
- Deploy the Web Store configuration and the feedback loop via the dashboard and start the experimental pipeline

- Collect the experiment results and make a decision

6.4.2 Results

We applied the steps above for the configuration with two variants of the recommendation service shown in Figure 6.3. We applied the experimental pipeline shown in Figure 6.2 using the specifications of the different elements illustrated above.

We conducted the "Upgrade v1.0.0 - v1.1.0" experiment that compared the distributions of the response time of the two variants for 20k samples. After the 20k samples were collected, we observed that the null hypothesis (response time of v1.1.0 similar to response time v1.0.0) could not be rejected. The pipeline thus continues with the next experiment: "Clicks v1.0.0 - v1.1.0". For more details and additional results, we refer to the project website [282].

6.5 On the Applicability of SEAByTE

SEAByTE targets the automation of continuous experimentation of micro-service systems. The artifact aims for a high level of reality of a practical micro-services system, yet, several aspects of real-world systems are emulated, including user behavior and system load. The artifact supports A/B testing for both functional and non-functional aspects. It also supports human involvement in the form of human feedback, which is a crucial aspect of A/B testing in practice.

SEAByTE comes with five scenarios; the code and specifications of two scenarios are shipped with the artifact. For the other scenarios, the artifact provides a basic implementation. Yet, the artifact does not exclude research on new scenarios.

It is important to balance the design of an experimental pipeline. Automatically running multiple experiments may result in a negative experience for some users [200]. Furthermore, it may accumulate errors in statistical tests, which need to be considered when defining p-values for subsequent tests.

6.6 Future research directions

The current version of SEAByTE supports the basics for setting up experimental pipelines for A/B testing using a feedback loop. We highlight several opportunities for future research beyond these basics. A first opportunity is to incorporate user feedback into the experimental pipeline, akin to one of the test scenarios presented in Table 6.1. A second opportunity is to identify classes of users to target A/B tests for specific subgroups. Here self-adaptation can be combined with machine learning to identify such classes and determine which class users belong to [284, 151]. Incorporating user feedback in this setting can also play a crucial role in developing a suitable adaptation strategy. A third opportunity for future research could be the automatic identification of A/B experiments that can be set up and run in the application. This can range from trying out different GUI layouts to testing different algorithms (similar to the recommendation algorithm tested in Section 6.3). A fourth and last opportunity is supporting multiple experiments simultaneously. Since running multiple A/B experiments can be a risk (experiments may affect each other), care has to be taken. Self-adaptation is a perfect candidate to deal with such concerns. Self-adaptation can also play a crucial role in dealing with a large space of potential experiments that need to be run. Self-adaptation can further help in identifying the experiments that are most likely to have a high impact on the overall business goals of the system, and determining in which order to run the experiments.

6.7 Conclusions

We presented SEAByTE, a novel artifact that applies self-adaptation [361] to enhance the automation of A/B testing to support the evolution of micro-service systems. In contrast to existing artifacts that target novel approaches to engineer self-adaptive systems, SEAByTE exploits self-adaptation as a means to solve a key task of software engineers of service-based systems: automating continuous experimentation. We hope that the research community will use the SEAByTE to evaluate research advances in the application of self-adaptation to support software engineers of micro-service systems. The artifact is available via the project website [282].

Table 6.1: Test scenarios for SEAByTE

ID	Name	Description	Metric
S1	Basic service upgrade	Experimental pipeline with two versions of a single micro-service	Response time of service invocations
S2	Advanced service upgrade	Experimental pipeline with two versions of a single micro-service	Response time of service invocations and user behavior
S3	Setting product price	Experimental pipeline that determines the right price for a product with the aim to maximize the revenue	Appreciation of the price of purchases by the users
S4	Basic segmentation	Experimental pipeline that determines a segmentation strategy for two variants based on the age of end-users	Preferences of variants using user feedback
S5	Advanced segmentation	Experimental pipeline that determines a segmentation strategy for two variants based on the age and geographic location of end-users	Preferences of variants using user feedback

Chapter 7

Conclusion

This chapter concludes this dissertation. We provide a summary of the context of the presented research presented, a summary of the contributions, and we discuss potential avenues for future work. Lastly, we conclude with two personal reflections.

In this dissertation, we investigated how self-adaptation supported by machine learning can be used to engineer modern software systems. With the increasing complexity of modern software systems and the environments they operate in, engineering such software systems presents numerous challenges. The field of self-adaptation presents a consolidated approach to address these challenges. The research field of self-adaptation has contributed numerous approaches to support the engineering of resilient software systems that can effectively deal with uncertainties during operation. However, several challenges remain to engineer self-adaptive systems that can deal with the increasing complexity of modern software systems and their environments in an efficient manner.

In light of these challenges, we explored two complementary perspectives. Each perspective was explored in a corresponding research track. In the first research track, we addressed the challenge of efficiently dealing with large adaptation spaces in self-adaptive systems. Although several approaches had been proposed to deal with large adaptation spaces, the approaches lacked the desired level of reusability. The approaches were either entangled with the adaptation logic, or tailored to specific types of systems. To address this challenge, we explored how the application of machine learning can be used to provide a more efficient self-adaptation process, ultimately leading to the ability to offer better adaptations to the managed system.

In the second research track, we addressed the challenge of automatically and efficiently executing A/B testing pipelines. In practice, A/B testing is a time-consuming process and manual process. Although some approaches had been proposed to automate

A/B testing, the automated execution of A/B testing remained an open challenge. Additionally, improving the efficiency of A/B testing remains a perpetual challenge. To address these challenges, we explored how machine learning and self-adaptation can be applied to automate the execution of A/B testing pipelines and to improve the efficiency of A/B testing.

7.1 Summary of contributions

Research track 1

Contribution 1: SLR Machine Learning in Self-Adaptive Systems

The first contribution in this thesis was a systematic literature review on the use of machine learning to realize self-adaptive systems, covered in Chapter 2. The literature review explored the types of machine learning utilized to support self-adaptation alongside the concrete problems they solve, contributing a clear overview of which machine learning techniques are used to solve different types of problems. Additionally, the literature review identified a number of open challenges and opportunities, which resulted in the definition of research question 1 of this dissertation:

> *How can machine learning be used to reduce large adaptation spaces of self-adaptive systems with different types of adaptation goals to perform more efficient analysis without compromising the goals?*

Contribution 2: ML2ASR+

In the second contribution, we presented ML2ASR+, a novel approach to handle large adaptation spaces of self-adaptive systems with different types of adaptation goals, covered in Chapter 3. The approach contributes a self-adaptive architecture with a formal foundation, introducing a *Machine Learning Module* in the architecture to select a subset of relevant adaptation options. The approach was evaluated in two domains: (1) the Internet-of-Things, and (2) Service-Based-Systems. With this contribution, we can formulate an answer to research question 1:

> We presented ML2ASR+, a novel approach to reduce large adaptation spaces of self-adaptive systems with different types of adaptation goals. ML2ASR+ leverages supervised machine learning techniques to reduce large adaptation spaces of self-adaptive systems with different types of adaptation goals. Moreover, ML2ASR+ efficiently achieves adaptation

> space reduction with negligible impact on system qualities, and without requiring input from end users at runtime.

Research track 2

Contribution 3: SLR A/B Testing

The third contribution in this thesis was a systematic literature review on A/B testing, covered in Chapter 4. The literature review presented an overview of several engineering aspects related to designing and executing A/B tests, contributing an overview of various A/B testing aspects such as the application domains, types of A/B tests, and roles stakeholders undertake in A/B testing. Additionally, the literature review identified several open challenges and future research directions, which resulted in the definition of research question 2 of this dissertation:

> *How can we automate A/B testing pipelines, and how can machine learning be used to run A/B testing pipelines more efficiently?*

Contribution 4: AutoPABS

In the fourth contribution, we presented AutoPABS, a novel approach for automatically and efficiently executing A/B testing pipelines, covered in Chapter 5. The approach contributes a notation of A/B testing pipelines and population splits, and an architectural solution to realize the automatic execution of A/B testing pipelines. A small survey supported the relevance and usefulness of the approach, and the approach was evaluated in the e-commerce domain. With this contribution, we can formulate an answer to research question 2:

> We presented AutoPABS, an approach that leverages self-adaptation principles and machine learning to (1) automate the execution of A/B testing pipelines, and (2) split populations in A/B testing to improve efficient execution of the A/B tests. AutoPABS sets up A/B tests in the managed system, monitors the deployed A/B tests, collects data from the A/B tests, and analyzes the results when the A/B tests are completed. Moreover, AutoPABS introduces a population split component to A/B testing pipelines, enabling splitting up populations to specific A/B tests to improve the efficiency of A/B testing. To realize the splitting, AutoPABS leverages supervised machine learning techniques.

Contribution 5: SEAByTE

The fifth contribution presented in this thesis is a technical contribution of an artifact named SEAByTE, presented in Chapter 6. SEAByTE comprises two parts: (1) a micro-service based web application of an online web-store and (2) a barebones self-adaptive system supporting the execution of A/B testing pipelines supported by a dashboard for operators. The artifact puts realism at the forefront by simulating real network traffic supported by user profiles. We used SEAByTE to evaluate AutoPABS in Contribution 4, in support to answer research question 2.

7.2 Threats to validity

We highlight the key threats to validity for each research track. For an elaborate discussion on the threats to the validity of the individual contributions, we refer back to the respective chapters of these contributions.

Researh track 1 Contribution 2 presented ML2ASR+ for adaptation space reduction in self-adaptive systems. In the approach we used classical supervised machine learning techniques to support the adaptation process. We applied and evaluated ML2ASR+ to a limited set of scenarios with particular characteristics and types of uncertainties, presenting a threat to the external validity of the approach. To mitigate this threat to some extent, we evaluated the approach in two domains with different challenges regarding adaptation space reduction for different combinations of adaptation goals. However, to fully confirm the generalizability of the approach further evaluation in different domains is essential.

Research track 2 Contribution 4 presented AutoPABS for automating efficient execution of A/B testing pipelines. The questionnaire we used to probe for the usefulness of automating A/B testing pipelines and population splits used closed questions, meaning that the participants were not able to provide nuanced answers. Moreover, we provided only a brief introduction to AutoPABS, presenting potential threats to construct validity. To mitigate this threat additional larger studies are required where participants can fully exploit the notation and infrastructure.

Additionally, we only evaluated AutoPABS in one scenario in the context of an online web-store. This presents a threat to external validity, since we cannot make general claims about the applicability of the approach for different contexts. We anticipated that the technology and domain used for the evaluation are particularly relevant for contemporary software systems. However, a more extensive evaluation of the approach in different domains is crucial to claim generalizability of AutoPABS.

7.3 Future work

Lastly, we present three lines of future work that we derived from the contributions in this dissertation.

7.3.1 Goal evolution in self-adaptive systems supported by machine learning

The first avenue for future work lies in the use of machine learning to support goal evolution in self-adaptive systems [64, 223]. Several approaches have provided accommodations for evolving adaptation goals, e.g., MORPH [38], a reference architecture for self-adaptation that explicitly models goal evolution as one of the key components in the architecture; SimCA* [316], a control-based approach that supports basic goal evolution during a Goal Update Phase; ActivFORMS [373], an approach that supports making changes at runtime to the feedback loop, including making changes to adaptation goals of the self-adaptive system; and Carwehl et al. [54], presenting an approach for runtime verification of self-adaptive systems that can deal with changing requirements. However, these approaches do not provide accommodations for goal evolution in the presence of machine learning.

Steps to enable support for goal evolution in the presence of machine learning include: (1) retraining machine learning models to accommodate for updated or new adaptation goals, (2) providing considerations for updating data stored in the Knowledge component of the MAPE loop, and (3) adjusting MAPE components (which leverage machine learning) to deal with evolving goals. Furthermore, goal evolution in the presence of machine learning also has implications for the guarantees that the self-adaptive system can provide. For example, the guarantees provided by the self-adaptive system might change when the goals evolve and the machine learning models are not updated satisfactorily. This presents another interesting avenue for future work.

7.3.2 Role of A/B testing in self-adaptive systems

A second track of future work we present lies in the additional use of A/B testing in self-adaptive systems. In this dissertation, we focused on the use of self-adaptation to realize automatic execution of A/B testing pipelines, and the use of machine learning to improve the efficiency of A/B testing. In these scenarios, self-adaptation is used to realize A/B testing. Conversely, an aspect we did not explore yet is the use of A/B testing[1] as an additional mechanism to realize self-adaptation in a data-driven manner. Limited work has explored the role of A/B testing within self-adaptation. One study exploits self-adaptation principles to realize DevOps process adoption within the context of digital twins for cyber-physical systems (CPS in short) [93]. The approach presented by the authors enables A/B testing in the CPS. The study does not explore the use of A/B testing to realize self-adaptation. We envision that A/B testing can be used to evaluate adaptation plans and actions. Furthermore, we see potential in the use of A/B testing to support lifelong learning [150], where A/B testing could be used as an additional measure to deal with concept drift, catastrophic forgetting, and other challenges that arise in lifelong learning.

On a similar note, several challenges remain to conduct A/B testing in the embedded systems domain. Self-adaptation has been used to alleviate runtime concerns (i.e., dealing with uncertainties), e.g., in the Internet-of-Things [176, 24], yet A/B testing has not been exploited to its full capacity in this domain. As presented by Mattos et al. [246], numerous open challenges hinder widespread adoption in the embedded systems domain, presenting a promising direction for future work.

7.3.3 Holistic approach to automating A/B testing pipelines

A third avenue for future work lies in the development of a holistic approach to automate A/B testing pipelines. In this dissertation, we presented AutoPABS, an approach to automate and efficiently execute A/B testing pipelines. The approach presented an initial step towards automating A/B testing pipelines. However, several challenges remain to fully automate A/B testing pipelines. For example, the approach does not incorporate known pitfalls and challenges in A/B testing, e.g., diagnosing sample ratio mismatches (SRM) [119], and Simpson's paradox [73]. Additionally, we did not deeply explore the role of humans during the experiment design and execution of A/B testing pipelines in the approach. A deeper understanding of these roles is crucial to fully automate A/B testing pipelines.

On top of this, opportunities for future work in improving AutoPABS lie in exploring the use of unsupervised learning to split populations in A/B testing pipelines. As opposed to splitting the population based on a specified split property such as the likelihood of purchasing products in a web store, unsupervised learning can be used to split populations based on unknown properties. Another opportunity for improving AutoPABS lies in devising a method to incorporate A/A testing in the presence of population splitting. A/A testing is a technique to control for bias in A/B testing before

[1] Interleaving experiments also present as an alternative approach to A/B testing in particular scenarios where conventional A/B testing cannot be effectively utilized.

running the actual A/B test [168, 60]. However, challenges arise when the population for A/B testing is split based on properties at runtime, rendering A/A testing infeasible. Additional research is required to solve this challenge.

7.4 Concluding reflections

Arriving at the end of this dissertation, I would like to reflect back on the journey that ultimately led to this dissertation. I have learned a significant amount during these 4 years, both in terms of research and personal development. Hoping that the previous 200+ pages of this dissertation have sufficiently covered the research part, I would like to focus on the personal development side in this last section.

If I would have to highlight one thing that I have learned during these 4 years, it would be the importance and value of feedback. It is the key ingredient to improve yourself and your work. Feedback can be used to fully develop and fledge out ideas, to identify issues and shortcomings early on, and to improve your skills. This also applies particularly to feedback from others: each individual has their own perspective and expertise, judging things from a different angle.

A second point that I would like to address is feelings of self-doubt and skepticism. For this, I put forward the following controversial statement: self-doubt and skepticism can be useful emotions. The caveat in the previous statement of course being: *in moderation*. A positive side of self-doubt is the resulting push to be more thorough in your efforts, aiming to deliver high-quality work. Too much self-doubt on the other hand feels more like a roadblock than anything else. Personally I have found that dealing with feelings of self-doubt and skepticism is a matter of putting things into perspective. For example: you were accepted as a PhD researcher, so you must be doing something right. Another example: accepted research has been thoroughly reviewed by fellow researchers, which deemed the work as a valuable contribution to the community. Awareness of these feelings and learning to put things into perspective have helped me immensely in my PhD journey.

I sincerely hope these reflections can be of use to others as well.

Appendix A

Appendix

A.1 Systematic Literature Review on Machine Learning in Self-Adaptation

A.1.1 List of Primary Studies

Title	Venue	Year
Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems	TAAS	2012
MARC: A Resource Consumption Modeling Service for Self-Aware Autonomous Agents	TAAS	2017
Fault Monitoring with Sequential Matrix Factorization	TAAS	2015
Generating Adaptation Rules of Software Systems: A Method Based on Genetic Algorithm	ICMLC	2018
To Adapt or Not to Adapt?: Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance	ICPE	2018
Adaptive model learning for continual verification of non-functional properties	ICPE	2014
SATISFy: Towards a Self-Learning Analyzer for Time Series Forecasting in Self-Improving Systems	FAS*W	2018
A Concept for Proactive Knowledge Construction in Self-Learning Autonomous Systems	FAS*W	2018
Meta-Learning for Realizing Self-x Management of Future Networks	IEEE Access	2017
Framework for Building Self-Adaptive Component Applications Based on Reinforcement Learning	SCC	2018

Introducing Deep Learning Self-Adaptive Misuse Network Intrusion Detection Systems	IEEE Access	2019
A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks	IEEE Access	2018
Self-Adaptation Applied to MQTT via a Generic Autonomic Management Framework	ICIT	2019
Instance-Based Learning for Hybrid Planning	FAS*W	2017
An algorithm for online planning to improve availability and performance of self-adaptive websites	CFIS	2017
Towards History-Aware Self-Adaptation with Explanation Capabilities	FAS*W	2019
Protecting Cyber Physical Systems Using a Learned MAPE-K Model	IEEE Access	2019
Reinforcement Learning-Based Predictive Control for Autonomous Electrified Vehicles	IV	2018
Machine Learning Meets Quantitative Planning: Enabling Self-Adaptation in Autonomous Robots	SEAMS	2019
A Reinforcement Learning-Based Framework for the Generation and Evolution of Adaptation Rules	ICAC	2017
A Learning Approach to Enhance Assurances for Real-Time Self-Adaptive Systems	SEAMS	2018
Adding Self-Improvement to an Autonomic Traffic Management System	ICAC	2017
A self-adaptation framework for dealing with the complexities of software changes	ICSESS	2017
Losing Control: The Case for Emergent Software Systems Using Autonomous Assembly, Perception, and Learning	SASO	2016
Two-Level Autonomous Optimizations Based on ML for Cardiac FEM Simulations	ICAC	2018
Adaptive runtime response time control in PLC-based real-time systems using reinforcement learning	SEAMS	2018
Adaptivity at every layer: a modular approach for evolving societies of learning autonomous systems	SEAMS	2008
Autonomic Software Product Lines (ASPL)	ECSA	2010
Learning to sample: exploiting similarities across environments to learn performance models for configurable systems	FSE	2018
Learning revised models for planning in adaptive systems	ICSE	2013
Knowledge Base K Models to Support Trade-Offs for Self-Adaptation using Markov Processes	SASO	2019
Learning a Dynamic Re-combination Strategy of Forecast Techniques at Runtime	ICAC	2015
Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning	SEAMS	2019
A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems	TSE	2013
Transfer Learning for Improving Model Predictions in Highly Configurable Software	SEAMS	2017

An Analysis of Decision-Making Techniques in Dynamic, Self-Adaptive Systems	SASO	2014
All Versus One: An Empirical Comparison on Retrained and Incremental Machine Learning for Modeling Performance of Adaptable Software	SEAMS	2019
Training Prediction Models for Rule-Based Self-Adaptive Systems	ICAC	2018
Learning non-deterministic impact models for adaptation	SEAMS	2018
FUSION: a framework for engineering self-tuning self-adaptive software systems	FSE	2010
RaM: Causally-Connected and Requirements-Aware Runtime Models using Bayesian Learning	MODELS	2019
Adaptive Execution of Continuous and Data-intensive Workflows with Machine Learning	Middleware	2018
Fuzzy Self-Learning Controllers for Elasticity Management in Dynamic Cloud Architectures	QoSA	2016
Handling Uncertainty in Self-Adaptive Software Using Self-Learning Fuzzy Neural Network	COMPSAC	2016
Supporting the Self-Learning of Systems at the Network Edge with Microservices	SSI	2019
Multi-Model Deep Learning for Cloud Resources Prediction to Support Proactive Workflow Adaptation	IEEE Cloud Summit	2019
A self-learning strategy for artificial cognitive control systems	INDIN	2015
SLOPE: A Self Learning Optimization and Prediction Ensembler for Task Scheduling	WiMob	2018
Self-Learning Production Systems (SLPS) — Energy management application for machine tools	ISIE	2013
Machine Learning for Achieving Self-* Properties and Seamless Execution of Applications in the Cloud	NCCA	2015
Driving skill analysis using machine learning The full curve and curve segmented cases	ITST	2012
Self-Learning approach to support lifecycle optimization of Manufacturing processes	IECON	2013
Managing Uncertainty in Autonomic Cloud Elasticity Controllers	IEEE Cloud Computing	2016
Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services	TSE	2017
Elasticat: A load rebalancing framework for cloud-based key-value stores	HiPC	2012
Energy Efficiency in Machine Tools - A Self-Learning Approach	SMC	2013
Model-based reinforcement learning approach for planning in self-adaptive software system	IMCOM	2015
Self-evolvable knowledge-enhanced IoT data mobility for smart environment	IML	2017
TSLAM: A Trust-enabled Self-Learning Agent Model for Service Matching in the Cloud Market	TAAS	2019
SLICE: self-learnable IoT common software engine	IOT	2018

An adaptive prediction approach based on workload pattern discrimination in the cloud	JNCA	2017
A Reconfiguration Algorithm for Power-Aware Parallel Applications	TACO	2016
Effective Decision Making in Self-adaptive Systems Using Cost-Benefit Analysis at Runtime and Online Learning of Adaptation Spaces	ENASE	2019
Self-Learning Production Systems: Adapter Reference Architecture	FAIM	2013
Decentralized Planning for Self-Adaptation in Multi-cloud Environment	ESOCC	2015
IDES: Self-adaptive Software with Online Policy Evolution Extended from Rainbow	Computer and Information Science (Book)	2012
Rationalism with a dose of empiricism: combining goal reasoning and case-based reasoning for self-adaptive software systems	Requirements Engineering Journal	2015
Adaptive process control based on a self-learning mechanism in autonomous manufacturing systems	JAMT	2012
A Q-Leaning-Based On-Line Planning Approach to Autonomous Architecture Discovery for Self-managed Software	OTM	2008
A three-phase decision making approach for self-adaptive systems using web services	CASM	2018
Architectural Homeostasis in Self-Adaptive Software-Intensive Cyber-Physical Systems	ECSA	2016
Towards Self-adaptation Planning for Complex Service-Based Systems	ICSOC	2013
Risk management for self-adapting self-organizing emergent multi-agent systems performing dynamic task fulfillment	AAMAS	2014
Synthesis and Verification of Self-aware Computing Systems	Self-Aware Computing Systems (Book)	2017
PRESC 22 : efficient self-reconfiguration of cache strategies for elastic caching platforms	Computing	2013
Avionics Self-adaptive Software: Towards Formal Verification and Validation	ICDCIT	2019
Self-* programming: run-time parallel control search for reflection box	Evolving Systems	2013
A Model-Based Approach to Dynamic Self-assessment for Automated Performance and Safety Awareness of Cyber-Physical Systems	IMBSA	2017
Automatic Adaptation of SOA Systems Supported by Machine Learning	DoCEIS	2013
VM Reservation Plan Adaptation Using Machine Learning in Cloud Computing	Journal of Grid Computing	2019

Machine learning-based auto-scaling for containerized applications	Neural Computing and Applications (Journal)	2019
A Model for Using Machine Learning in Smart Environments	GPC	2011
A cognitive/intelligent resource provisioning for cloud computing services: opportunities and challenges	Soft Computing	2019
An autonomic resource provisioning framework for efficient data collection in cloudlet-enabled wireless body area networks: a fuzzy-based proactive approach	Soft Computing	2019
An autonomic approach for resource provisioning of cloud services	Cluster Computing	2016
Toward Proactive Learning of Multi-layerd Cloud Service Based Application	CLOSER	2016
Utilizing Twitter Data for Identifying and Resolving Runtime Business Process Disruptions	OTM	2018
A Self Healing Microservices Architecture: A Case Study in Docker Swarm Cluster	AINA	2019
Performance Comparison of Deep VM Workload Prediction Approaches for Cloud	ICCAN	2018
Towards Automated Analysis and Optimization of Multimedia Streaming Services Using Clustering and Semantic Techniques	MACE	2010
Building Autonomic Elements from Video-Streaming Servers	JNSM	2019
PSO-based novel resource scheduling technique to improve QoS parameters in cloud computing	Neural Computing and Applications (Journal)	2019
On the use of hybrid reinforcement learning for autonomic resource allocation	Cluster Computing	2007
IO dependent SSD cache allocation for elastic Hadoop applications	SCIS (Jorunal)	2018
Architecting Dependable Systems with Proactive Fault Management	Architecting Dependable Systems VII (Book)	2010
An autonomic provisioning framework for outsourcing data center based on virtual appliances	Cluster Computing	2008
An Auto-Scaling Cloud Controller Using Fuzzy Q-Learning - Implementation in OpenStack	ESOCC	2016
Mobile Apps with Dynamic Bindings Between the Fog and the Cloud	ICSOC	2019
A Self-Learning Scheduling in Cloud Software Defined Block Storage	CLOUD	2017
Self-Adaptive Learning PSO-Based Deadline Constrained Task Scheduling for Hybrid IaaS Cloud	TASE	2014
Dynamic adaptation of policies using machine learning	CCGRID	2016

A self-learning approach for validation of runtime adaptation in service-oriented systems	SOCA	2017
Self-adaptation for Mobile Robot Algorithms Using Organic Computing Principles	ARCS	2013
Using a recurrent artificial neural network for dynamic self-adaptation of cluster-based web-server systems	APIN	2017
A Control-Theoretic Approach to Self-adaptive Systems and an Application to Cloud-Based Software	LASER	2013
A Nash equilibrium based decision-making method for internet of things	JAIHC	2019
Building Automated Data Driven Systems for IT Service Management	JNSM	2017
Catch-up TV forecasting: enabling next-generation over-the-top multimedia TV services	MTAP	2017
Dynamic QoS Management and Optimization in Service-Based Systems	TSE	2011

A.2 Reducing Large Adaptation Spaces

A.2.1 Auxiliary Formal Definitions

Model training (split train-test)

To enable the evaluation of learning models, one option is to split the data set into 2 parts: a training data set and a testing data set. We define a *split* function as follows (tr short for training and te short for testing):

> $Map : \Lambda_i \rightarrow \Phi_i$ is a function that maps a system state, represented by a feature vector, to the qualities of the system, represented by a quality vector.
>
> $Split : \Lambda \times \Phi \times W \rightarrow \Lambda \times \Phi \times \Lambda \times \Phi$
>
> $Split(\Lambda_o, \Phi_o, w) = < \Lambda_{tr}, \Phi_{tr}, \Lambda_{te}, \Phi_{te} >$ with
> $\quad \Lambda_{tr} \cup \Lambda_{te} = \Lambda_o$ and $\Phi_{tr} \cup \Phi_{te} = \Phi_o$ and
> $\quad |\Lambda_{tr}| = w * |\Lambda_o|$ and $|\Phi_{tr}| = w * |\Phi_o|$ and
> $\quad \forall \lambda_i \in \Lambda_{tr}$: $Map(\lambda_i) = \phi_i$ with $\phi_i \in \Phi_{tr}$ and
> $\quad \forall \lambda_j \in \Lambda_{te}$: $Map(\lambda_j) = \phi_j$ with $\phi_j \in \Phi_{te}$

The training data set is used to train the machine learning models, while the testing data set is used to test and validate the trained machine learning models. This testing is conducted by comparing the predictions made to the actual quality values in the form of machine learning evaluation metrics.

Exploration

We formally define the selection of explored adaptation options as follows:

$$\begin{aligned}
&DetermineExploration : \Pi \times E \rightarrow \Pi \\
&DetermineExploration(\{\pi_1, ..., \pi_n\}, e) = \Pi_e \text{ with} \\
&\qquad \Pi_e\{\, \pi_i \in \{\pi_1, ..., \pi_n\} \mid \pi_i \notin \Pi_{filtered}\,\} \quad \text{and} \\
&\qquad |\Pi_e| = e * |\{\pi_1, ..., \pi_n\}|
\end{aligned}$$

The set $\Pi_{filtered}$ refers to the set of adaptation options that were predicted by the machine learning models to satisfy the adaptation goals. Hence, we explore adaptation options outside the set of adaptation options that were already selected for verification. It is important to note that adaptation options that are predicted to meet all system goals should be given priority in case of insufficient time to verify all the included adaptation options. The logic is that the explored adaptation options should not hinder the verification of adaptation options with greater promise.

Filter

We formally define filtering as follows:

$G = \{g\}$: The granularity that defines an upper bound on the size of the filtered adaptation space.

$\mathbb{G}_s \in \mathbb{G}$: The specific set of adaptation goals of the system.

$Filter : \Pi \times \mathbb{Z} \times G \times \mathbb{G} \rightarrow \Pi$

$Filter(\Pi_i, \{\Omega_1, ..., \Omega_n\}, g, \mathbb{G}_s) = \Pi_j \;\text{ where }\; \Pi_j \subseteq \Pi_i \;\text{ and }\; |\Pi_j| \leqslant g$

Filtering takes a set of adaptation options, a set of quality predictions, a granularity value that puts a bound on adaptation space reduction, and an adaptation goal. The result is a reduced set of adaptation options.

The criteria for filtering adaptation options vary depending on the type of quality goals that are evaluated. In particular, filtering handles three types of operations, one for each type of adaptation goal. The first type of filter operation filters adaptation options that do not comply with a threshold goal in the system. Formally, the filter operation for a threshold goal $\mathcal{T} \in \mathbb{T}$ with a threshold value $\bar{x}$ for any quality value q is defined as follows:

$$f_{\mathcal{T}_{<\bar{x}}} = \{\pi_1, \pi_2, ..., \pi_n\} \mapsto \{\pi_{f_1}, \pi_{f_2}, ..., \pi_{f_m}\} \text{ where}$$
$$\mathcal{T}_{<\bar{x}}(q_k) = True,\ k \in \{f_1, f_2, ..., f_m\}$$

$$f_{\mathcal{T}_{>\bar{x}}} = \{\pi_1, \pi_2, ..., \pi_n\} \mapsto \{\pi_{f_1}, \pi_{f_2}, ..., \pi_{f_m}\} \text{ where}$$
$$\mathcal{T}_{>\bar{x}}(q_k) = True,\ k \in \{f_1, f_2, ..., f_m\}$$

The second type of filter operation relates to setpoint goals in the system. We define the filter operation that filters adaptation options according to setpoint goal $\mathcal{S}$ with target μ and error margin ϵ for any quality value q as follows:

$$f_{\mathcal{S}_{\mu,\epsilon}} = \{\pi_1, \pi_2, ..., \pi_n\} \mapsto \{\pi_{f_1}, \pi_{f_2}, ..., \pi_{f_m}\} \text{ where}$$
$$m \leq g \text{ and } \sum_{i=0}^{m} \left|q_{f_i} - \mu\right| = min(\{\sum_{k \in K} |q_k - \mu| \mid K \subseteq \Pi \text{ and } |K| = m\})$$

Lastly, the filter deals with up to one optimization goal. We define the filter operation for an optimization goal $\mathcal{O}$ for quality values q as follows:

$$f_{\mathcal{O}_{min}} = \{\pi_1, \pi_2, ..., \pi_n\} \mapsto \{\pi_{f_1}, \pi_{f_2}, ..., \pi_{f_m}\} \text{ where}$$
$$m \leq g \text{ and } \sum_{i=0}^{m} q_{f_i} = min(\{\sum_{k \in K} q_k\} \mid K \subseteq \Pi \text{ and } |K| = m)$$

$$f_{\mathcal{O}_{max}} = \{\pi_1, \pi_2, ..., \pi_n\} \mapsto \{\pi_{f_1}, \pi_{f_2}, ..., \pi_{f_m}\} \text{ where}$$
$$m \leq g \text{ and } \sum_{i=0}^{m} q_{f_i} = max(\{\sum_{k \in K} q_k\} \mid K \subseteq \Pi \text{ and } |K| = m)$$

In our research, we use filters that combine the different filter operations in a predefined order. In particular, the filter first filters adaptation options that violate threshold goals. Next, it filters adaptation options that violate the setpoint goals of the system. Finally, it filters the options based on a single optimization goal. We restrict filtering to a single optimization goal to avoid conflicting scenarios when multiple optimization goals are specified in the system. Equation A.1 specifies how we define the main filter operation:

$$\mathcal{F} = f_{\mathcal{O}} \circ ... \circ f_{\mathcal{S}_2} \circ f_{\mathcal{S}_1} \circ ... \circ f_{\mathcal{T}_2} \circ f_{\mathcal{T}_1} \qquad \text{(A.1)}$$

In case any of the types of adaptation goals are not applicable, that type is ignored by the filter.

Table A.2: Design stage model selection for the DeltaIoT application in the two system scenarios: scenario 1 (top), and scenario 2 (bottom). MCC: Matthews Correlation Coefficient; MSE: Mean Squared Error; MAE: Median Absolute Error; ME: Maximum Error

Scenario 1	$\mathcal{T}^{\text{packet loss}}_{<10\%}$, $\mathcal{T}^{\text{latency}}_{<5\%}$
ML algorithm	SGD Classifier
Loss function	Log
Penalty function	l1
Scaler type	MinMax Scaler
Exploration rate	5%
Warm-up count	45
Metrics	*F1*: 0.818 [0.022, 0.818] *MCC*: 0.715 [-0.004, 0.716]

Scenario 2	$\mathcal{T}^{\text{packet loss}}_{<10\%}$, $\mathcal{T}^{\text{latency}}_{<5\%}$	$\mathcal{O}^{\text{energy consumption}}_{min}$
ML algorithm	SGD Classifier	Passive Aggressive Regressor
Loss function	Log	Squared Epsilon Insensitive
Penalty function	l1	N/A
Scaler type	MinMax Scaler	None
Exploration rate	5%	5%
Warm-up count	45	45
Metrics	*F1*: 0.818 [0.022, 0.818] *MCC*: 0.715 [-0.004, 0.716]	*R2*: 0.833 [-1.091, 0.854] *MSE*: 0.004 [0.004, $8.8e^{24}$] *MAE*: 0.043 [0.040, $4e^{12}$] *ME*: 0.269 [0.241, $9e^{12}$]

A.2.2 Additional Machine Learning Material

Table A.2, Table A.4, and Table A.3 summarize the scalers and models selected for evaluation scenarios in both applications. The numbers between square brackets indicate the boundaries of the evaluation metric values for the alternative options that were not selected. For both applications and scenarios, the *warm-up count* is selected from 30, 45, and 60, and the *exploration rate* is selected from 5% and 10%.

Table A.3: Design stage model selection for the Service-Based System application in the second system scenario.

Scenario 2	$\mathcal{T}^{\text{failure rate}}_{<10\%}$	$\mathcal{S}^{\text{response time}}_{10ms,\epsilon=0.25ms}$	$\mathcal{O}^{\text{cost}}_{min}$
ML algorithm	SGD Classifier	Passive Aggressive Regressor	Passive Aggressive Regressor
Loss function	Hinge	Squared Epsilon Insensitive	Epsilon Insensitive
Penalty function	Elasticnet	N/A	N/A
Scaler type	Standard Scaler	None	None
Exploration rate	5%	5%	5%
Warm-up count	60	60	60
Metrics		*R2*: 0.860 [-5.0e^{25}, 0.868]	*R2*: 0.906 [-7.8e^{23}, 0.908]
	F1: 0.933 [0.293, 0.934]	*MSE*: 0.035 [0.033, 1.4e^{25}]	*MSE*: 1.753 [1.706, 1.4e^{25}]
	MCC: 0.866 [-0.028, 0.867]	*MAE*: 0.123 [0.119, 2.8e^{12}]	*MAE*: 0.901 [0.859, 2.7e^{12}]
		ME: 0.976 [0.975, 1.5e^{13}]	*ME*: 5.981 [5.981, 1.3e^{13}]

Table A.4: Design stage model selection for the Service-Based System application in the first system scenario.

Scenario 1	$\mathcal{T}^{\text{failure rate}}_{<10\%}$, $\mathcal{T}^{\text{response time}}_{<10ms}$	$\mathcal{O}^{\text{cost}}_{min}$
ML algorithm	SGD Classifier	Passive Aggressive Regressor
Loss function	Hinge	Squared Epsilon Insensitive
Penalty function	l1	N/A
Scaler type	None	None
Exploration rate	5%	5%
Warm-up count	60	60
Metrics		*R2*: 0.906 [-7.8e^{23}, 0.908]
	F1: 0.895 [0.000, 0.895]	*MSE*: 1.753 [1.706, 1.4e^{25}]
	MCC: 0.812 [-0.112, 0.812]	*MAE*: 0.901 [0.859, 2.7e^{12}]
		ME: 5.981 [5.981, 1.3e^{13}]

A.3 A/B Testing: A Systematic Literature Review

A.3.1 List of Primary Studies

Table A.5: List of primary studies.

ID	Ref.	Title
1	[1]	A Nonparametric Sequential Test for Online Randomized Experiments
2	[307]	Detecting Network Effects: Randomizing Over Randomized Experiments
3	[204]	Unexpected Results in Online Controlled Experiments
4	[59]	How A/B Tests Could Go Wrong: Automatic Diagnosis of Invalid Online Experiments
5	[274]	Boosted Decision Tree Regression Adjustment for Variance Reduction in Online Controlled Experiments
6	[199]	Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained
7	[200]	Online Controlled Experiments at Large Scale
8	[382]	Non-Stationary A/B Tests
9	[160]	Network A/B Testing: From Sampling to Estimation
10	[385]	False Discovery Rate Controlled Heterogeneous Treatment Effect Detection for Online Controlled Experiments
11	[113]	Experimentation Pitfalls to Avoid in A/B Testing for Online Personalization
12	[83]	Statistical Inference in Two-Stage Online Controlled Experiments with Treatment Selection and Validation
13	[41]	Consistent Transformation of Ratio Metrics for Efficient Online Controlled Experiments

14	[357]	CONQ: CONtinuous Quantile Treatment Effects for Large-Scale Online Controlled Experiments
15	[119]	Diagnosing Sample Ratio Mismatch in Online Controlled Experiments: A Taxonomy and Rules of Thumb for Practitioners
16	[321]	IPEAD A/B Test Execution Framework
17	[187]	Peeking at A/B Tests: Why It Matters, and What to Do about It
18	[195]	Learning Sensitive Combinations of A/B Test Metrics
19	[99]	Practical Aspects of Sensitivity in Online Experimentation with User Engagement Metrics
20	[387]	Evaluating Mobile Apps with A/B and Quasi A/B Tests
21	[84]	On Post-Selection Inference in A/B Testing
22	[100]	Online Experimentation with Surrogate Metrics: Guidelines and a Case Study
23	[96]	Future User Engagement Prediction and Its Application to Improve the Sensitivity of Online Experiments
24	[118]	The Evolution of Continuous Experimentation in Software Product Development: From Data to a Data-Driven Organization at Scale
25	[386]	How to Measure Your App: A Couple of Pitfalls and Remedies in Measuring App Performance in Online Controlled Experiments
26	[196]	Sequential Testing for Early Stopping of Online Experiments
27	[89]	Shrinkage Estimators in Online Experiments
28	[342]	A Counterfactual Framework for Seller-Side A/B Testing on Marketplaces
29	[97]	Periodicity in User Engagement with a Search Engine and Its Application to Online Controlled Experiments
30	[228]	Evolving Software to be ML-Driven Utilizing Real-World A/B Testing: Experiences, Insights, Challenges
31	[98]	Using the Delay in a Treatment Effect to Improve Sensitivity and Preserve Directionality of Engagement Metrics in A/B Experiments
32	[168]	A Cluster-Based Nearest Neighbor Matching Algorithm for Enhanced A/A Validation in Online Experimentation
33	[232]	Variance-Weighted Estimators to Improve Sensitivity in Online Experiments
34	[92]	A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments
35	[221]	Winner's Curse: Bias Estimation for Total Effects of Features in Online Controlled Experiments
36	[388]	From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks
37	[188]	A Sequential Test for Selecting the Better Variant: Online A/B Testing, Adaptive Allocation, and Continuous Monitoring
38	[326]	Unbiased Experiments in Congested Networks
39	[345]	Personalized Treatment Selection Using Causal Heterogeneity
40	[238]	Beyond Success Rate: Utility as a Search Quality Metric for Online Experiments
41	[396]	Algorithms and System Architecture for Immediate Personalized News Recommendations
42	[229]	Experimentation in the Operating System: The Windows Experimentation Platform

43	[350]	AB4Web: An On-Line A/B Tester for Comparing User Interface Design Alternatives
44	[262]	Real-World Product Deployment of Adaptive Push Notification Scheduling on Smartphones
45	[207]	Mining the Stars: Learning Quality Ratings with User-Facing Explanations for Vacation Rentals
46	[93]	Towards Digital Twin-Enabled DevOps for CPS Providing Architecture-Based Service Adaptation & Verification at Runtime
47	[359]	Adaptive Experimentation with Delayed Binary Feedback
48	[230]	Unifying Offline Causal Inference and Online Bandit Learning for Data Driven Decision
49	[10]	Beyond Data: From User Information to Business Value through Personalized Recommendations and Consumer Science
50	[225]	Learning to Bundle Proactively for On-Demand Meal Delivery
51	[21]	Measuring Dynamic Effects of Display Advertising in the Absence of User Tracking Information
52	[20]	Marketing Campaign Evaluation in Targeted Display Advertising
53	[249]	Whole Page Optimization: How Page Elements Interact with the Position Auction
54	[293]	The MOOClet Framework: Unifying Experimentation, Dynamic Improvement, and Personalization in Online Courses
55	[389]	Split-Treatment Analysis to Rank Heterogeneous Causal Effects for Prospective Interventions
56	[218]	Promoting Positive Post-Click Experience for In-Stream Yahoo Gemini Users
57	[299]	Predicting Counterfactuals from Large Historical Data and Small Randomized Trials
58	[330]	Multi-Source Pointer Network for Product Title Summarization
59	[76]	Beyond Relevance Ranking: A General Graph Matching Framework for Utility-Oriented Learning to Rank
60	[158]	Offline Evaluation to Make Decisions About PlaylistRecommendation Algorithms
61	[162]	Trustworthy Experimentation Under Telemetry Loss
62	[155]	The Netflix Recommender System: Algorithms, Business Value, and Innovation
63	[308]	Bifrost: Supporting Continuous Deployment with Automated Enactment of Multi-Phase Live Testing Strategies
64	[137]	CompactETA: A Fast Inference System for Travel Time Prediction
65	[19]	Design and Analysis of Benchmarking Experiments for Distributed Internet Services
66	[109]	Learning to Rank in the Position Based Model with Bandit Feedback
67	[35]	VisRel: Media Search at Scale
68	[394]	Behavioral Consequences of Reminder Emails on Students' Academic Performance: A Real-World Deployment
69	[231]	Content Recommendation by Noise Contrastive Transfer Learning of Feature Representation
70	[123]	External Evaluation of Ranking Models under Extreme Position-Bias
71	[346]	Tackling Cannibalization Problems for Online Advertisement

72	[334]	Filling Context-Ad Vocabulary Gaps with Click Logs
73	[135]	Practical Lessons from Developing a Large-Scale Recommender System at Zalando
74	[381]	How Airbnb Tells You Will Enjoy Sunset Sailing in Barcelona? Recommendation in a Two-Sided Travel Marketplace
75	[390]	Modeling Professional Similarity by Mining Professional Career Trajectories
76	[58]	Social Incentive Optimization in Online Social Networks
77	[319]	Ad Close Mitigation for Improved User Experience in Native Advertisements
78	[272]	Off-Line vs. On-Line Evaluation of Recommender Systems in Small E-Commerce
79	[2]	LASER: A Scalable Response Prediction Platform for Online Advertising
80	[6]	The Role of Relevance in Sponsored Search
81	[300]	Contextual Bandit Applications in a Customer Support Bot
82	[383]	Safe Velocity: A Practical Guide to Software Deployment at Scale using Controlled Rollout
83	[333]	When Relevance is Not Enough: Promoting Diversity and Freshness in Personalized Question Recommendation
84	[226]	Interference, Bias, and Variance in Two-Sided Marketplace Experimentation: Guidance for Platforms
85	[248]	Automotive A/B testing: Challenges and Lessons Learned from Practice
86	[36]	A/B Testing at SweetIM: The Importance of Proper Statistical Analysis
87	[81]	A Framework Model to Support A/B Tests at the Class and Component Level
88	[186]	Statistical Reasoning of Zero-Inflated Right-Skewed User-Generated Big Data A/B Testing
89	[236]	Size matters? Or not: A/B testing with limited sample in automotive embedded software
90	[351]	Scalable Data Reporting Platform for A/B Tests
91	[30]	Applying Bayesian parameter estimation to A/B tests in e-business applications examining the impact of green marketing signals in sponsored search advertising
92	[8]	Experiment-driven improvements in Human-in-the-loop Machine Learning Annotation via significance-based A/B testing
93	[165]	The Anatomy of a Large-Scale Experimentation Platform
94	[12]	Demystifying dark matter for online experimentation
95	[156]	Controlled experiments for decision-making in e-Commerce search
96	[215]	Evaluating usability of a web application: A comparative analysis of open-source tools
97	[60]	Faster online experimentation by eliminating traditional A/A validation
98	[91]	Pitfalls of long-term online controlled experiments
99	[327]	SoftSKU: Optimizing Server Architectures for Microservice Diversity @Scale
100	[117]	The Benefits of Controlled Experimentation at Scale
101	[317]	Context Adaptation for Smart Recommender Systems
102	[393]	Whales, Dolphins, or Minnows? Towards the Player Clustering in Free Online Games Based on Purchasing Behavior via Data Mining Technique
103	[233]	Enterprise-Level Controlled Experiments at Scale: Challenges and Solutions
104	[31]	Should companies bid on their own brand in sponsored search?

105	[167]	A Probabilistic, Mechanism-Indepedent Outlier Detection Method for Online Experimentation
106	[403]	Inform Product Change through Experimentation with Data-Driven Behavioral Segmentation
107	[247]	Your System Gets Better Every Day You Use It: Towards Automated Continuous Experimentation
108	[57]	Fashion Recommendation Systems, Models and Methods: A Review
109	[9]	Subject Line Personalization Techniques and Their Influence in the E-Mail Marketing Open Rate
110	[240]	Impact of promotional social media content on click-through rate - Evidence from a FMCG company
111	[7]	Related Entity Expansion and Ranking Using Knowledge Graph
112	[43]	LinkLouvain: Link-Aware A/B Testing and Its Application on Online Marketing Campaign
113	[253]	Ascend by Evolv: Artificial intelligence-based massively multivariate conversion rate optimization
114	[398]	A new framework for online testing of heterogeneous treatment effect
115	[343]	JACKPOT: Online experimentation of cloud microservices
116	[163]	Digital Marketing Effectiveness Using Incrementality
117	[306]	Business process improvement with the AB-BPM methodology
118	[103]	A genetic algorithm for finding a small and diverse set of recent news stories on a given subject: How we generate aaai's ai-alert
119	[213]	Measuring the value of recommendation links on product demand
120	[116]	Experimentation growth: Evolving trustworthy A/B testing capabilities in online software companies
121	[311]	Online Evaluation of Bid Prediction Models in a Large-Scale Computational Advertising Platform: Decision Making and Insights
122	[305]	AB-BPM: Performance-driven instance routing for business process improvement
123	[102]	Have It Both Ways—From A/B Testing to A&B Testing with Exceptional Model Mining
124	[245]	More for Less: Automated Experimentation in Software-Intensive Systems
125	[68]	Regression Tree for Bandits Models in A/B Testing
126	[271]	When the Crowd is Not Enough: Improving User Experience with Social Media through Automatic Quality Analysis
127	[39]	Pixel efficiency analysis: A quantitative web analytics approach
128	[208]	A/B Testing in E-commerce Sales Processes
129	[264]	A Method for the Construction of User Targeting Knowledge for B2B Industry Website
130	[279]	Validating Mobile Designs with Agile Testing in China: Based on Baidu Map for Mobile
131	[356]	User Latent Preference Model for Better Downside Management in Recommender Systems
132	[335]	Towards Automated A/B Testing
133	[201]	Seven rules of thumb for web site experimenters

134	[224]	Enabling A/B Testing of Native Mobile Applications by Remote User Interface Exchange
135	[338]	Overlapping Experiment Infrastructure: More, Better, Faster Experimentation
136	[138]	Optimizing price levels in e-commerce applications: An empirical study
137	[52]	Facilitating Controlled Tests of Website Design Changes: A Systematic Approach
138	[3]	Soft Frequency Capping for Improved Ad Click Prediction in Yahoo Gemini Native
139	[82]	Objective Bayesian Two Sample Hypothesis Testing for Online Controlled Experiments
140	[126]	Test & Roll: Profit-Maximizing A/B Tests
141	[397]	Improving Library User Experience with A/B Testing: Principles and Process

Bibliography

[1] Vineet Abhishek and Shie Mannor. “A Nonparametric Sequential Test for Online Randomized Experiments”. In: *Proceedings of the 26th International Conference on World Wide Web Companion*. WWW ’17 Companion. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 610–616. ISBN: 9781450349147. DOI: 10.1145/3041021.3054196 (pp. 161, 162, 231).

[2] Deepak Agarwal, Bo Long, Jonathan Traupman, Doris Xin, and Liang Zhang. “LASER: A Scalable Response Prediction Platform for Online Advertising”. In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM ’14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 173–182. ISBN: 9781450323512. DOI: 10.1145/2556195.2556252 (pp. 161, 165, 234).

[3] Michal Aharon, Yohay Kaplan, Rina Levy, Oren Somekh, Ayelet Blanc, Neetai Eshel, Avi Shahar, Assaf Singer, and Alex Zlotnik. “Soft Frequency Capping for Improved Ad Click Prediction in Yahoo Gemini Native”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM ’19. Beijing, China: Association for Computing Machinery, 2019, pp. 2793–2801. ISBN: 9781450369763. DOI: 10.1145/3357384.3357801 (pp. 161, 236).

[4] Michal Aharon, Oren Somekh, Avi Shahar, Assaf Singer, Baruch Trayvas, Hadas Vogel, and Dobri Dobrev. “Carousel Ads Optimization in Yahoo Gemini Native”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1993–2001. ISBN: 9781450362016. DOI: 10.1145/3292500.3330740 (p. 127).

[5] Mir Shahnawaz Ahmad and Shahid Mehraj Shah. “Supervised Machine Learning Approaches for Attack Detection in the IoT Network”. In: *Internet of Things and Its Applications*. Springer, 2022, pp. 247–260. DOI: 10.1007/978-981-16-7637-6_22 (p. 83).

[6] Luca Aiello, Ioannis Arapakis, Ricardo Baeza-Yates, Xiao Bai, Nicola Barbieri, Amin Mantrach, and Fabrizio Silvestri. "The Role of Relevance in Sponsored Search". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. Indianapolis, Indiana, USA: Association for Computing Machinery, 2016, pp. 185–194. ISBN: 9781450340731. DOI: 10.1145/2983323.2983840 (pp. 146, 161, 234).

[7] Ryuya Akase, Hiroto Kawabata, Akiomi Nishida, Yuki Tanaka, and Tamaki Kaminaga. "Related Entity Expansion and Ranking Using Knowledge Graph". In: *Complex, Intelligent and Software Intensive Systems*. Ed. by Leonard Barolli, Kangbin Yim, and Tomoya Enokido. Cham: Springer International Publishing, 2021, pp. 172–184. ISBN: 978-3-030-79725-6 (pp. 137, 152, 161, 235).

[8] Rafael Alfaro-Flores, José Salas-Bonilla, Loic Juillard, and Juan Esquivel-Rodríguez. "Experiment-driven improvements in Human-in-the-loop Machine Learning Annotation via significance-based A/B testing". In: *2021 XLVII Latin American Computing Conference (CLEI)*. 2021, pp. 1–9. DOI: 10.1109/CLEI53233.2021.9639977 (pp. 161, 234).

[9] Joana Almeida and Beatriz Casais. "Subject Line Personalization Techniques and Their Influence in the E-Mail Marketing Open Rate". In: *Information Systems and Technologies*. Ed. by Alvaro Rocha, Hojjat Adeli, Gintautas Dzemyda, and Fernando Moreira. Cham: Springer International Publishing, 2022, pp. 532–540. ISBN: 978-3-031-04829-6 (pp. 152, 161, 235).

[10] Xavier Amatriain. "Beyond Data: From User Information to Business Value through Personalized Recommendations and Consumer Science". In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. CIKM '13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 2201–2208. ISBN: 9781450322638. DOI: 10.1145/2505515.2514701 (pp. 137, 161, 233).

[11] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies". In: *Information and Software Technology* 106 (2019), pp. 201–230. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2018.10.006 (p. 165).

[12] Nirupama Appiktala, Miao Chen, Michael Natkovich, and Joshua Walters. "Demystifying dark matter for online experimentation". In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 1620–1626. DOI: 10.1109/BigData.2017.8258096 (pp. 161, 234).

[13] Hamid Arabnejad, Pooyan Jamshidi, Giovani Estrada, Nabil El Ioini, and Claus Pahl. "An Auto-Scaling Cloud Controller Using Fuzzy Q-Learning - Implementation in OpenStack". In: *Service-Oriented and Cloud Computing*. Ed. by Marco Aiello, Einar Broch Johnsen, Schahram Dustdar, and Ilche

Georgievski. Cham: Springer International Publishing, 2016, pp. 152–167. ISBN: 978-3-319-44482-6. DOI: 10.1007/978-3-319-44482-6_10 (p. 38).

[14] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation". In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2015, pp. 13–23. DOI: 10.1109/SEAMS.2015.10 (p. 2).

[15] Argo-Rollouts. https://argoproj.github.io/argo-rollouts/. 2023 (p. 173).

[16] Florian Auer and Michael Felderer. "Current State of Research on Continuous Experimentation: A Systematic Mapping Study". In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2018, pp. 335–344. DOI: 10.1109/SEAA.2018.00062 (pp. 124, 128).

[17] Florian Auer, Rasmus Ros, Lukas Kaltenbrunner, Per Runeson, and Michael Felderer. "Controlled experimentation in continuous experimentation: Knowledge and challenges". In: *Information and Software Technology* 134 (2021), p. 106551. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2021.106551 (pp. 124, 128).

[18] Alia Ayoub and Amal Elgammal. "Utilizing Twitter Data for Identifying and Resolving Runtime Business Process Disruptions". In: *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*. Ed. by Hervé Panetto, Christophe Debruyne, Henderik A. Proper, Claudio Agostino Ardagna, Dumitru Roman, and Robert Meersman. Cham: Springer International Publishing, 2018, pp. 189–206. ISBN: 978-3-030-02610-3. DOI: 10.1007/978-3-030-02610-3_11 (pp. 73, 85).

[19] Eytan Bakshy and Eitan Frachtenberg. "Design and Analysis of Benchmarking Experiments for Distributed Internet Services". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 108–118. ISBN: 9781450334693. DOI: 10.1145/2736277.2741082 (pp. 127, 161, 165, 233).

[20] Joel Barajas, Jaimie Kwon, Ram Akella, Aaron Flores, Marius Holtan, and Victor Andrei. "Marketing Campaign Evaluation in Targeted Display Advertising". In: *Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy*. ADKDD '12. Beijing, China: Association for Computing Machinery, 2012. ISBN: 9781450315456. DOI: 10.1145/2351356.2351361 (pp. 144, 157, 161, 233).

[21] Joel Barajas, Jaimie Kwon, Ram Akella, Aaron Flores, Marius Holtan, and Victor Andrei. "Measuring Dynamic Effects of Display Advertising in the Absence of User Tracking Information". In: *Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet*

Economy. ADKDD '12. Beijing, China: Association for Computing Machinery, 2012. ISBN: 9781450315456. DOI: 10.1145/2351356.2351364 (pp. 139, 161, 233).

[22] Luciano Baresi, Giovanni Quattrocchi, and Nicholas Rasi. "Federated Machine Learning as a Self-Adaptive Problem". In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2021, pp. 41–47. DOI: 10.1109/SEAMS51251.2021.00016 (p. 95).

[23] Victor R. Basili, Gianluigi Caldiera, and Dieter H. Rombach. "The Goal Question Metric Approach". In: vol. I. John Wiley & Sons, 1994 (p. 129).

[24] Jacob Beal, Mirko Viroli, Danilo Pianini, and Ferruccio Damiani. "Self-Adaptation to Device Distribution in the Internet of Things". In: *ACM Trans. Auton. Adapt. Syst.* 12.3 (Sept. 2017). ISSN: 1556-4665. DOI: 10.1145/3105758 (p. 218).

[25] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. "150 Successful Machine Learning Models: 6 Lessons Learned at Booking.Com". In: *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1743–1751. ISBN: 9781450362016. DOI: 10.1145/3292500.3330744 (pp. 5, 193).

[26] Narayan Bhamidipati, Ravi Kant, and Shaunak Mishra. "A Large Scale Prediction Engine for App Install Clicks and Conversions". In: *ACM on Conference on Information and Knowledge Management*. 2017. ISBN: 9781450349185. DOI: 10.1145/3132847.3132868 (p. 170).

[27] Kay Bierzynski, Pavel Lutskov, and Uwe Assmann. "Supporting the Self-Learning of Systems at the Network Edge with Microservices". In: *Smart Systems Integration; 13th International Conference and Exhibition on Integration Issues of Miniaturized Systems*. 2019, pp. 1–8 (p. 35).

[28] Christopher Bishop. *Pattern recognition and machine learning*. springer, 2006 (pp. 25, 38).

[29] Gordon Blair, Nelly Bencomo, and Robert France. "Models@ run.time". In: *Computer* 42 (Nov. 2009), pp. 22–27. DOI: 10.1109/MC.2009.326 (pp. 3, 7, 22, 52).

[30] Tobias Blask. "Applying Bayesian parameter estimation to A/B tests in e-business applications examining the impact of green marketing signals in sponsored search advertising". In: *2013 International Conference on e-Business (ICE-B)*. 2013, pp. 1–8 (pp. 140, 161, 234).

[31] Tobias Blask, Burkhardt Funk, and Reinhard Schulte. "Should companies bid on their own brand in sponsored search?" In: *Proceedings of the International Conference on e-Business*. 2011, pp. 1–8 (pp. 140, 152, 161, 234).

[32] Thomas Bodenheimer, Kate Lorig, Halsted Holman, and Kevin Grumbach. “Patient Self-Management of Chronic Disease in Primary Care”. In: *JAMA : the journal of the American Medical Association* 288 (Dec. 2002), pp. 2469–75. DOI: 10.1001/jama.288.19.2469 (p. 101).

[33] Justus Bogner, Jonas Fritzsch, Stefan Wagner, and Alfred Zimmermann. “Industry practices and challenges for the evolvability assurance of microservices”. In: *Empirical Software Engineering* 26.104 (July 2021), pp. 24–35. ISSN: 1573-7616. DOI: 10.1007/s10664-021-09999-9 (p. 198).

[34] Justus Bogner, Jonas Fritzsch, Stefan Wagner, and Alfred Zimmermann. “Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality”. In: *Companion Int. Conference on Software Architecture Companion*. 2019. DOI: 10.1109/ICSA-C.2019.00041 (p. 198).

[35] Fedor Borisyuk, Siddarth Malreddy, Jun Mei, Yiqun Liu, Xiaoyi Liu, Piyush Maheshwari, Anthony Bell, and Kaushik Rangadurai. “VisRel: Media Search at Scale”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD ’21. Virtual Event, Singapore: Association for Computing Machinery, 2021, pp. 2584–2592. ISBN: 9781450383325. DOI: 10.1145/3447548.3467081 (pp. 143, 161, 233).

[36] Slava Borodovsky and Saharon Rosset. “A/B Testing at SweetIM: The Importance of Proper Statistical Analysis”. In: *2011 IEEE 11th International Conference on Data Mining Workshops*. 2011, pp. 733–740. DOI: 10.1109/ICDMW.2011.19 (pp. 146, 161, 234).

[37] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. “Counterfactual Reasoning and Learning Systems: The Example of Computational Advertising”. In: *Journal of Machine Learning Research* 14.101 (2013), pp. 3207–3260. URL: https://jmlr.org/papers/v14/bottou13a.html (p. 172).

[38] Victor Braberman, Nicolas D’Ippolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. “MORPH: A Reference Architecture for Configuration and Behaviour Self-Adaptation”. In: *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*. CTSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, pp. 9–16. ISBN: 9781450338141. DOI: 10.1145/2804337.2804339 (p. 217).

[39] Alex Brown, Binky Lush, and Bernard J. Jansen. “Pixel efficiency analysis: A quantitative web analytics approach”. In: *Proceedings of the Association for Information Science and Technology* 53.1 (2016), pp. 1–10. DOI: 10.1002/pra2.2016.14505301040 (pp. 139, 154, 161, 163, 235).

[40] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. "Coordinated Self-Configuration of Virtual Machines and Appliances Using a Model-Free Learning Approach". In: *IEEE Transactions on Parallel and Distributed Systems* 24.4 (2013), pp. 681–690. DOI: 10.1109/TPDS.2012.174 (p. 55).

[41] Roman Budylin, Alexey Drutsa, Ilya Katsev, and Valeriya Tsoy. "Consistent Transformation of Ratio Metrics for Efficient Online Controlled Experiments". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM '18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 55–63. ISBN: 9781450355810. DOI: 10.1145/3159652.3159699 (pp. 161, 231).

[42] Tomáš Bureš. "Self-Adaptation 2.0". In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 2021, pp. 262–263. DOI: 10.1109/SEAMS51251.2021.00046 (pp. 171, 200).

[43] Tianchi Cai, Daxi Cheng, Chen Liang, Ziqi Liu, Lihong Gu, Huizhi Xie, Zhiqiang Zhang, Xiaodong Zeng, and Jinjie Gu. "LinkLouvain: Link-Aware A/B Testing and Its Application on Online Marketing Campaign". In: *Database Systems for Advanced Applications*. Ed. by Christian S. Jensen, Ee-Peng Lim, De-Nian Yang, Wang-Chien Lee, Vincent S. Tseng, Vana Kalogeraki, Jen-Wei Huang, and Chih-Ya Shen. Cham: Springer International Publishing, 2021, pp. 499–510. ISBN: 978-3-030-73200-4 (pp. 137, 161, 162, 235).

[44] Radu Calinescu, Marco Autili, Javier Cámara, Antinisca Di Marco, Simos Gerasimou, Paola Inverardi, Alexander Perucci, Nils Jansen, Joost-Pieter Katoen, Marta Kwiatkowska, Ole J. Mengshoel, Romina Spalazzese, and Massimo Tivoli. "Synthesis and Verification of Self-aware Computing Systems". In: *Self-Aware Computing Systems*. Ed. by Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu. Cham: Springer International Publishing, 2017, pp. 337–373. ISBN: 978-3-319-47474-8. DOI: 10.1007/978-3-319-47474-8_11 (p. 43).

[45] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. "Dynamic QoS Management and Optimization in Service-Based Systems". In: *IEEE Transactions on Software Engineering* 37.3 (May 2011), pp. 387–409. ISSN: 0098-5589. DOI: 10.1109/TSE.2010.92 (pp. 4, 24, 30, 39, 53).

[46] Radu Calinescu, Shinji Kikuchi, and Kenneth Johnson. "Compositional Reverification of Probabilistic Safety Properties for Large-Scale Complex IT Systems". In: *Large-Scale Complex IT Systems. Development, Operation and Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 303–329. ISBN: 978-3-642-34059-8. DOI: 10.1007/978-3-642-34059-8_16 (p. 56).

[47] Radu Calinescu, Raffaela Mirandola, Diego Perez-Palacin, and Danny Weyns. "Understanding Uncertainty in Self-adaptive Systems". In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. 2020, pp. 242–251. DOI: 10.1109/ACSOS49614.2020.00047 (pp. 37, 48).

[48] Radu Calinescu, Yasmin Rafiq, Kenneth Johnson, and Mehmet Emin Bakır. "Adaptive Model Learning for Continual Verification of Non-Functional Properties". In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*. ICPE '14. Dublin, Ireland: Association for Computing Machinery, 2014, pp. 87–98. ISBN: 9781450327336. DOI: 10.1145/2568088.2568094 (pp. 31, 43).

[49] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. "Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases". In: *IEEE Transactions on Software Engineering* 44.11 (2018), pp. 1039–1069. DOI: 10.1109/TSE.2017.2738640 (p. 4).

[50] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, and R. Ventura. "Incorporating architecture-based self-adaptation into an adaptive industrial software system". In: *Journal of Systems and Software* 122 (2016), pp. 507–523. ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.09.021 (p. 22).

[51] Javier Cámara, David Garlan, Gabriel A. Moreno, and Bradley Schmerl. "Analyzing Self-Adaptation Via Model Checking of Stochastic Games". In: *Software Engineering for Self-Adaptive Systems III. Assurances*. Ed. by Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese. Cham: Springer International Publishing, 2017, pp. 154–187. ISBN: 978-3-319-74183-3. DOI: 10.1007/978-3-319-74183-3_6 (pp. 24, 53).

[52] Javier Cámara and Alfred Kobsa. "Facilitating Controlled Tests of Website Design Changes: A Systematic Approach". In: *Web Engineering*. Ed. by Martin Gaedke, Michael Grossniklaus, and Oscar Díaz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 370–378. ISBN: 978-3-642-02818-2 (pp. 161, 236).

[53] Javier Cámara, Henry Muccini, and Karthik Vaidhyanathan. "Quantitative Verification-Aided Machine Learning: A Tandem Approach for Architecting Self-Adaptive IoT Systems". In: *Proceedings International Conference on Software Architecture, ICSA*. 2020, pp. 11–22. DOI: 10.1109/ICSA47634.2020.00010 (pp. 11, 54).

[54] Marc Carwehl, Thomas Vogel, Genaina Rodrigues, and Lars Grunske. "Runtime Verification of Self-Adaptive Systems with Changing Requirements". In: *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Los Alamitos, CA, USA: IEEE Computer

Society, May 2023, pp. 104–114. DOI: 10.1109/SEAMS59076.2023.00024 (p. 217).

[55] Maria Casimiro, Paolo Romano, David Garlan, Gabriel A. Moreno, Eunsuk Kang, and Mark Klein. "Self-Adaptation for Machine Learning Based Systems". In: *European Conference on Software Architecture*. Vol. 2978. CEUR Workshop Proceedings. 2021. URL: https://ceur-ws.org/Vol-2978/saml-paper6.pdf (pp. 171, 200).

[56] Gert Cauwenberghs and Tomaso Poggio. "Incremental and Decremental Support Vector Machine Learning". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00. Denver, CO: MIT Press, 2000, pp. 388–394 (p. 80).

[57] Samit Chakraborty, Md. Saiful Hoque, Naimur Rahman Jeem, Manik Chandra Biswas, Deepayan Bardhan, and Edgar Lobaton. "Fashion Recommendation Systems, Models and Methods: A Review". In: *Informatics* 8.3 (2021). ISSN: 2227-9709. DOI: 10.3390/informatics8030049 (pp. 137, 161, 235).

[58] Guangde Chen, Bee-Chung Chen, and Deepak Agarwal. "Social Incentive Optimization in Online Social Networks". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM '17. Cambridge, United Kingdom: Association for Computing Machinery, 2017, pp. 547–556. ISBN: 9781450346757. DOI: 10.1145/3018661.3018700 (pp. 161, 234).

[59] Nanyu Chen, Min Liu, and Ya Xu. "How A/B Tests Could Go Wrong: Automatic Diagnosis of Invalid Online Experiments". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. WSDM '19. Melbourne VIC, Australia: Association for Computing Machinery, 2019, pp. 501–509. ISBN: 9781450359405. DOI: 10.1145/3289600.3291000 (pp. 11, 139, 152, 153, 161, 162, 164, 171, 231).

[60] Russell Chen, Miao Chen, Mahendrasinh Ramsinh Jadav, Joonsuk Bae, and Don Matheson. "Faster online experimentation by eliminating traditional A/A validation". In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 1635–1641. DOI: 10.1109/BigData.2017.8258098 (pp. 161, 162, 219, 234).

[61] Tao Chen and Rami Bahsoon. "Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services". In: *IEEE Transactions on Software Engineering* 43.5 (2017), pp. 453–475. DOI: 10.1109/TSE.2016.2608826 (pp. 38, 43).

[62] Tao Chen and Rami Bahsoon. "Self-adaptive and online QoS modeling for cloud-based software services". In: *IEEE Transactions on Software Engineering* 43.5 (2016), pp. 453–475. DOI: 10.1109/TSE.2016.2608826 (p. 54).

[63] Tao Chen, Rami Bahsoon, Shuo Wang, and Xin Yao. “To Adapt or Not to Adapt? Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance”. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE ’18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 48–55. ISBN: 9781450350952. DOI: 10.1145/3184407.3184413 (pp. 43, 44).

[64] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. “Software Engineering for Self-Adaptive Systems: A Research Roadmap”. In: *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26. ISBN: 978-3-642-02161-9. DOI: 10.1007/978-3-642-02161-9_1 (pp. 1, 2, 22, 52, 174, 217).

[65] Betty H.C. Cheng, Andres Ramirez, and Philip K. McKinley. “Harnessing evolutionary computation to enable dynamically adaptive systems to manage uncertainty”. In: *2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*. 2013, pp. 1–6. DOI: 10.1109/CMSBSE.2013.6604427 (pp. 22, 53).

[66] Shang-Wen Cheng. “Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation”. In: *PhD Thesis Carnegie Mellon University, CMU-ISR-08-113*. 2008 (p. 199).

[67] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. “Evaluating the effectiveness of the Rainbow self-adaptive system”. In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. 2009, pp. 132–141. DOI: 10.1109/SEAMS.2009.5069082 (p. 6).

[68] Emmanuelle Claeys, Pierre Gançarski, Myriam Maumy-Bertrand, and Hubert Wassner. “Regression Tree for Bandits Models in A/B Testing”. In: *Advances in Intelligent Data Analysis XVI*. Ed. by Niall Adams, Allan Tucker, and David Weston. Cham: Springer International Publishing, 2017, pp. 52–62. ISBN: 978-3-319-68765-0 (pp. 161, 235).

[69] Commons-Math. *Apache Commons Mathematics Library*. *https://commons.apache.org/proper/commons-math/* (p. 202).

[70] Rafael Costa, Elie Cheniaux, Pedro Rosaes, Marcele Carvalho, Rafael Freire, Márcio Versiani, Bernard Range, and Antonio Nardi. “The effectiveness of cognitive behavioral group therapy in treating bipolar disorder: A randomized controlled study”. In: *Revista brasileira de psiquiatria (São Paulo, Brazil :*

1999) 33 (June 2011), pp. 144–9. DOI: 10.1590/S1516-44462011000200009 (p. 127).

[71] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. “Online Passive-Aggressive Algorithms”. In: *Journal of Machine Learning Research* 7 (Mar. 2006), pp. 551–585 (p. 83).

[72] John Creswell and Timothy Guetterman. *Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research, 6th Edition*. New York, NY, USA: Pearson, Feb. 2018. ISBN: 9780134519395 (pp. 125, 127).

[73] Thomas Crook, Brian Frasca, Ron Kohavi, and Roger Longbotham. “Seven Pitfalls to Avoid When Running Controlled Experiments on the Web”. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’09. Paris, France: Association for Computing Machinery, 2009, pp. 1105–1114. ISBN: 9781605584959. DOI: 10.1145/1557019.1557139 (p. 218).

[74] Laizhong Cui, Shu Yang, Fei Chen, Zhong Ming, Nan Lu, and Jing Qin. “A survey on application of machine learning for Internet of Things”. In: *International Journal of Machine Learning and Cybernetics* 9.8 (Aug. 2018), pp. 1399–1417. ISSN: 1868-808X. DOI: 10.1007/s13042-018-0834-5 (p. 28).

[75] Mirko D’Angelo, Simos Gerasimou, Sona Ghahremani, Johannes Grohmann, Ingrid Nunes, Evangelos Pournaras, and Sven Tomforde. “On Learning in Collective Self-Adaptive Systems: State of Practice and a 3D Framework”. In: *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. May 2019, pp. 13–24. DOI: 10.1109/SEAMS.2019.00012 (p. 28).

[76] Xinyi Dai, Yunjia Xi, Weinan Zhang, Qing Liu, Ruiming Tang, Xiuqiang He, Jiawei Hou, Jun Wang, and Yong Yu. “Beyond Relevance Ranking: A General Graph Matching Framework for Utility-Oriented Learning to Rank”. In: *ACM Trans. Inf. Syst.* 40.2 (Nov. 2021). ISSN: 1046-8188. DOI: 10.1145/3464303 (pp. 137, 146, 154, 161, 233).

[77] Zhenwen Dai, Praveen Ravichandran, Ghazal Fazelnia, Ben Carterette, and Mounia Lalmas-Roelleke. “Model selection for production system via automated online experiments”. In: vol. 2020-December. 2020. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85108405793%5C&partnerID=40%5C&md5=2a08fa5f662c442b711e0f999cf23618 (pp. 11, 172).

[78] Maya Daneva, Daniela Damian, Alessandro Marchetto, and Oscar Pastor. “Empirical research methodologies and studies in Requirements Engineering: How far did we come?” In: *Journal of Systems and Software* 95 (2014), pp. 1–9. ISSN: 0164-1212. DOI: 10.1016/j.jss.2014.06.035 (p. 127).

[79] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. "Uppaal SMC tutorial". In: *International Journal on Software Tools for Technology Transfer* 17.4 (2015), pp. 397–415. ISSN: 1433-2787. DOI: 10.1007/s10009-014-0361-y (pp. 91, 106).

[80] Daniele De Sensi, Massimo Torquati, and Marco Danelutto. "A Reconfiguration Algorithm for Power-Aware Parallel Applications". In: *ACM Transactions on Architecture and Code Optimization* 13.4 (Dec. 2016). ISSN: 1544-3566. DOI: 10.1145/3004054 (pp. 39, 44).

[81] Wagner S. De Souza, Fernando O. Pereira, Vanessa G. Albuquerque, Jorge Melegati, and Eduardo Guerra. "A Framework Model to Support A/B Tests at the Class and Component Level". In: *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2022, pp. 860–865. DOI: 10.1109/COMPSAC54236.2022.00136 (pp. 161, 162, 234).

[82] Alex Deng. "Objective Bayesian Two Sample Hypothesis Testing for Online Controlled Experiments". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15 Companion. Florence, Italy: Association for Computing Machinery, 2015, pp. 923–928. ISBN: 9781450334730. DOI: 10.1145/2740908.2742563 (pp. 161, 236).

[83] Alex Deng, Tianxi Li, and Yu Guo. "Statistical Inference in Two-Stage Online Controlled Experiments with Treatment Selection and Validation". In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. Seoul, Korea: Association for Computing Machinery, 2014, pp. 609–618. ISBN: 9781450327442. DOI: 10.1145/2566486.2568028 (pp. 161, 231).

[84] Alex Deng, Yicheng Li, Jiannan Lu, and Vivek Ramamurthy. "On Post-Selection Inference in A/B Testing". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD '21. Virtual Event, Singapore: Association for Computing Machinery, 2021, pp. 2743–2752. ISBN: 9781450383325. DOI: 10.1145/3447548.3467129 (pp. 161, 232).

[85] Alex Deng and Xiaolin Shi. "Data-Driven Metric Development for Online Controlled Experiments: Seven Lessons Learned". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 77–86. ISBN: 9781450342322. DOI: 10.1145/2939672.2939700 (p. 170).

[86] Niranjana Deshpande and Naveen Sharma. "Composition Algorithm Adaptation in Service Oriented Systems". In: *Software Architecture*. Ed. by Henry Muccini, Paris Avgeriou, Barbora Buhnova, Javier Camara, Mauro Caporuscio, Mirco Franzago, Anne Koziolek, Patrizia Scandurra, Catia Trubiani, Danny Weyns, and Uwe Zdun. Springer International Publishing, 2020, pp. 170–179. ISBN: 978-3-030-59155-7. DOI: 10.1007/978-3-030-59155-7_13 (p. 83).

[87] Pierangelo Di Sanzo, Alessandro Pellegrini, and Dimiter R. Avresky. “Machine Learning for Achieving Self-* Properties and Seamless Execution of Applications in the Cloud”. In: *Proceedings of the 2015 IEEE 4th Symposium on Network Cloud Computing and Applications*. NCCA ’15. USA: IEEE Computer Society, 2015, pp. 51–58. ISBN: 9781467377416. DOI: 10.1109/NCCA.2015.18 (p. 56).

[88] Alhassan Boner Diallo, Hiroyuki Nakagawa, and Tatsuhiro Tsuchiya. “Adaptation Space Reduction Using an Explainable Framework”. In: *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2021, pp. 1653–1660. DOI: 10.1109/COMPSAC51774.2021.00247 (pp. 11, 53, 55, 83).

[89] Drew Dimmery, Eytan Bakshy, and Jasjeet Sekhon. “Shrinkage Estimators in Online Experiments”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2914–2922. ISBN: 9781450362016. DOI: 10.1145/3292500.3330771 (pp. 139, 143, 161, 232).

[90] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. “A decade of agile methodologies: Towards explaining agile software development”. In: *Journal of Systems and Software* 85.6 (2012). Special Issue: Agile Development, pp. 1213–1221. ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.02.033 (p. 197).

[91] Pavel Dmitriev, Brian Frasca, Somit Gupta, Ron Kohavi, and Garnet Vaz. “Pitfalls of long-term online controlled experiments”. In: *2016 IEEE International Conference on Big Data (Big Data)*. 2016, pp. 1367–1376. DOI: 10.1109/BigData.2016.7840744 (pp. 137, 161, 162, 234).

[92] Pavel Dmitriev, Somit Gupta, Dong Woo Kim, and Garnet Vaz. “A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1427–1436. ISBN: 9781450348874. DOI: 10.1145/3097983.3098024 (pp. 161, 162, 232).

[93] Jürgen Dobaj, Andreas Riel, Thomas Krug, Matthias Seidl, Georg Macher, and Markus Egretzberger. “Towards Digital Twin-Enabled DevOps for CPS Providing Architecture-Based Service Adaptation & Verification at Runtime”. In: *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 132–143. ISBN: 9781450393058. DOI: 10.1145/3524844.3528057 (pp. 154, 161, 162, 218, 233).

[94] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. "A Survey of Autonomic Communications". In: *ACM Trans. Auton. Adapt. Syst.* 1.2 (Dec. 2006), pp. 223–259. ISSN: 1556-4665. DOI: 10.1145/1186778.1186782 (p. 2).

[95] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. "Microservices: Yesterday, Today, and Tomorrow". In: *Present and Ulterior Software Engineering*. Ed. by Manuel Mazzara and Bertrand Meyer. Cham: Springer International Publishing, 2017, pp. 195–216. ISBN: 978-3-319-67425-4. DOI: 10.1007/978-3-319-67425-4_12 (p. 198).

[96] Alexey Drutsa, Gleb Gusev, and Pavel Serdyukov. "Future User Engagement Prediction and Its Application to Improve the Sensitivity of Online Experiments". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 256–266. ISBN: 9781450334693. DOI: 10.1145/2736277.2741116 (pp. 161, 162, 164, 232).

[97] Alexey Drutsa, Gleb Gusev, and Pavel Serdyukov. "Periodicity in User Engagement with a Search Engine and Its Application to Online Controlled Experiments". In: *ACM Trans. Web* 11.2 (Apr. 2017). ISSN: 1559-1131. DOI: 10.1145/2856822 (pp. 12, 137, 152, 157, 161, 162, 164, 232).

[98] Alexey Drutsa, Gleb Gusev, and Pavel Serdyukov. "Using the Delay in a Treatment Effect to Improve Sensitivity and Preserve Directionality of Engagement Metrics in A/B Experiments". In: *Proceedings of the 26th International Conference on World Wide Web*. WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 1301–1310. ISBN: 9781450349130. DOI: 10.1145/3038912.3052664 (pp. 12, 137, 143, 161, 232).

[99] Alexey Drutsa, Anna Ufliand, and Gleb Gusev. "Practical Aspects of Sensitivity in Online Experimentation with User Engagement Metrics". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. Melbourne, Australia: Association for Computing Machinery, 2015, pp. 763–772. ISBN: 9781450337946. DOI: 10.1145/2806416.2806496 (pp. 125, 131, 161, 232).

[100] Weitao Duan, Shan Ba, and Chunzhe Zhang. "Online Experimentation with Surrogate Metrics: Guidelines and a Case Study". In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 193–201. ISBN: 9781450382977. DOI: 10.1145/3437963.3441737 (pp. 124, 144, 161, 162, 164, 232).

[101] Francisco Duarte, Richard Gil, Paolo Romano, Antónia Lopes, and Luís Rodrigues. “Learning Non-Deterministic Impact Models for Adaptation”. In: *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 196–205. ISBN: 9781450357159. DOI: 10.1145/3194133.3194138 (pp. 5, 37).

[102] Wouter Duivesteijn, Tara Farzami, Thijs Putman, Evertjan Peer, Hilde J. P. Weerts, Jasper N. Adegeest, Gerson Foks, and Mykola Pechenizkiy. “Have It Both Ways—From A/B Testing to A&B Testing with Exceptional Model Mining”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Yasemin Altun, Kamalika Das, Taneli Mielikäinen, Donato Malerba, Jerzy Stefanowski, Jesse Read, Marinka Žitnik, Michelangelo Ceci, and Sašo Džeroski. Cham: Springer International Publishing, 2017, pp. 114–126. ISBN: 978-3-319-71273-4 (pp. 152, 159, 161, 235).

[103] Joshua Eckroth and Eric Schoen. “A genetic algorithm for finding a small and diverse set of recent news stories on a given subject: How we generate aaai’s ai-alert”. In: 2019, pp. 9357–9364. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85090801224%5C&partnerID=40%5C&md5=f3391d595e00df8a0cba7802c9043ebd (pp. 146, 161, 235).

[104] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994. ISBN: 9781000064988. URL: https://books.google.be/books?id=MWC1DwAAQBAJ (p. 165).

[105] Ibrahim Elgendi, Md. Farhad Hossain, Abbas Jamalipour, and Kumudu S. Munasinghe. “Protecting Cyber Physical Systems Using a Learned MAPE-K Model”. In: *IEEE Access* 7 (2019), pp. 90954–90963. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2927037 (p. 38).

[106] Ehsan Elhamifar and René Vidal. “Sparse Subspace Clustering: Algorithm, Theory, and Applications”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2765–2781. DOI: 10.1109/TPAMI.2013.57 (p. 47).

[107] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. “FUSION: A Framework for Engineering Self-Tuning Self-Adaptive Software Systems”. In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE ’10. Santa Fe, New Mexico, USA: Association for Computing Machinery, 2010, pp. 7–16. ISBN: 9781605587912. DOI: 10.1145/1882291.1882296 (pp. 4, 6, 11, 30, 39, 54).

[108] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. “Model Evolution by Run-Time Parameter Adaptation”. In: *Proceedings of the 31st International Conference on Software Engineering*. ICSE ’09. USA: IEEE Computer Society, 2009, pp. 111–121. ISBN: 9781424434534. DOI: 10.1109/ICSE.2009.5070513 (pp. 4, 27).

[109] Beyza Ermis, Patrick Ernst, Yannik Stein, and Giovanni Zappella. “Learning to Rank in the Position Based Model with Bandit Feedback”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. CIKM ’20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 2405–2412. ISBN: 9781450368599. DOI: 10.1145/3340531.3412723 (pp. 137, 161, 233).

[110] Vladimir M. Erthal, Bruno P. de Souza, Paulo Sérgio M. dos Santos, and Guilherme H. Travassos. “A Literature Study to Characterize Continuous Experimentation in Software Engineering”. In: 2022. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85137064966%5C&partnerID=40%5C&md5=04240b73ab90eb841083173be558b33f (p. 128).

[111] Naeem Esfahani, Ahmed Elkhodary, and Sam Malek. “A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems”. In: *IEEE Transactions on Software Engineering* 39.11 (Nov. 2013), pp. 1467–1493. ISSN: 2326-3881. DOI: 10.1109/TSE.2013.37 (pp. 14, 39, 44, 45).

[112] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. “Taming Uncertainty in Self-Adaptive Software”. In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ESEC/FSE ’11. Szeged, Hungary: Association for Computing Machinery, 2011, pp. 234–244. ISBN: 9781450304436. DOI: 10.1145/2025113.2025147 (p. 4).

[113] Maria Esteller-Cucala, Vicenc Fernandez, and Diego Villuendas. “Experimentation Pitfalls to Avoid in A/B Testing for Online Personalization”. In: *Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization*. UMAP’19 Adjunct. Larnaca, Cyprus: Association for Computing Machinery, 2019, pp. 153–159. ISBN: 9781450367110. DOI: 10.1145/3314183.3323853 (pp. 161, 162, 164, 231).

[114] Aleksander Fabijan, Benjamin Arai, Pavel Dmitriev, and Lukas Vermeer. “It takes a Flywheel to Fly: Kickstarting and Growing the A/B testing Momentum at Scale”. In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2021, pp. 109–118. DOI: 10.1109/SEAA53835.2021.00023 (pp. 164, 172).

[115] Aleksander Fabijan, Pavel Dmitriev, Helena Holmstrom Olsson, and Jan Bosch. “The Online Controlled Experiment Lifecycle”. In: *IEEE Software* 37.02 (Mar. 2020), pp. 60–67. ISSN: 1937-4194. DOI: 10.1109/MS.2018.2875842 (p. 127).

[116] Aleksander Fabijan, Pavel Dmitriev, Colin McFarland, Lukas Vermeer, Helena Holmström Olsson, and Jan Bosch. “Experimentation growth: Evolving trustworthy A/B testing capabilities in online software companies”. In: *Journal of Software: Evolution and Process* 30.12 (2018). e2113 JSME-17-0210.R2,

e2113. DOI: 10.1002/smr.2113. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2113 (pp. 9, 137, 161, 235).

[117] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. "The Benefits of Controlled Experimentation at Scale". In: *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2017, pp. 18–26. DOI: 10.1109/SEAA.2017.47 (pp. 9, 155, 159, 161, 171, 234).

[118] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. "The Evolution of Continuous Experimentation in Software Product Development: From Data to a Data-Driven Organization at Scale". In: *Proceedings of the 39th International Conference on Software Engineering*. ICSE '17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 770–780. ISBN: 9781538638682. DOI: 10.1109/ICSE.2017.76 (pp. 124, 127, 161, 163, 164, 232).

[119] Aleksander Fabijan, Jayant Gupchup, Somit Gupta, Jeff Omhover, Wen Qin, Lukas Vermeer, and Pavel Dmitriev. "Diagnosing Sample Ratio Mismatch in Online Controlled Experiments: A Taxonomy and Rules of Thumb for Practitioners". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2156–2164. ISBN: 9781450362016. DOI: 10.1145/3292500.3330722 (pp. 137, 139, 154, 161, 218, 232).

[120] Aleksander Fabijan, Helena Holmström Olsson, and Jan Bosch. "Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review". In: *Software Business*. Ed. by João M. Fernandes, Ricardo J. Machado, and Krzysztof Wnuk. Cham: Springer International Publishing, 2015, pp. 139–153. ISBN: 978-3-319-19593-3. DOI: 10.1007/978-3-319-19593-3_12 (p. 128).

[121] Aleksander Fabijan, Helena Holmström Olsson, and Jan Bosch. "The Lack of Sharing of Customer Data in Large Software Organizations: Challenges and Implications". In: *Agile Processes, in Software Engineering, and Extreme Programming*. Ed. by Helen Sharp and Tracy Hall. Cham: Springer International Publishing, 2016, pp. 39–52. ISBN: 978-3-319-33515-5. DOI: 10.1007/978-3-319-33515-5_4 (p. 129).

[122] Fabian Fagerholm, Alejandro Sanchez Guinea, Hanna Mäenpää, and Jürgen Münch. "The RIGHT model for Continuous Experimentation". In: *Journal of Systems and Software* 123 (2017), pp. 292–305. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.03.034 (p. 197).

[123] Yaron Fairstein, Elad Haramaty, Arnon Lazerson, and Liane Lewin-Eytan. "External Evaluation of Ranking Models under Extreme Position-Bias". In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. WSDM '22. Virtual Event, AZ, USA: Association for Computing Machinery, 2022, pp. 252–261. ISBN: 9781450391320. DOI: 10.1145/3488560.3498420 (pp. 142, 161, 233).

[124] Feature-Flags. https://www.split.io/product/feature-flags/. 2023 (p. 172).

[125] Rico de Feijter, Rob van Vliet, Erik Jagroep, Sietse Overbeek, and Sjaak Brinkkemper. *Towards the adoption of DevOps in software product organizations: A maturity model approach.* Tech. rep. Utrecht University, 2017 (p. 127).

[126] Elea McDonnell Feit and Ron Berman. "Test & Roll: Profit-Maximizing A/B Tests". In: *Marketing Science* 38.6 (2019), pp. 1038–1058. DOI: 10.1287/mksc.2019.1194 (pp. 161, 236).

[127] Dawei Feng and Cecile Germain. "Fault Monitoring with Sequential Matrix Factorization". In: *ACM Trans. Auton. Adapt. Syst.* 10.3 (Oct. 2015). ISSN: 1556-4665. DOI: 10.1145/2797141 (p. 43).

[128] Lorenzo Fernández Maimó, Ángel Luis Perales Gómez, Félix J. García Clemente, Manuel Gil Pérez, and Gregorio Martínez Pérez. "A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks". In: *IEEE Access* 6 (2018), pp. 7700–7712. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2803446 (pp. 8, 30, 39, 40, 43, 47, 73, 85).

[129] Matteo Ferroni, Andrea Corna, Andrea Damiani, Rolando Brondolin, John D. Kubiatowicz, Donatella Sciuto, and Marco D. Santambrogio. "MARC: A Resource Consumption Modeling Service for Self-Aware Autonomous Agents". In: *ACM Trans. Auton. Adapt. Syst.* 12.4 (Nov. 2017). ISSN: 1556-4665. DOI: 10.1145/3127499 (p. 43).

[130] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. "Run-time Efficient Probabilistic Model Checking". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 341–350. ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985840 (p. 56).

[131] Antonio Filieri, Henry Hoffmann, and Martina Maggio. "Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees". In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 299–310. ISBN: 9781450327565. DOI: 10.1145/2568225.2568272 (p. 4).

[132] Benito E Flores. "A pragmatic view of accuracy measurement in forecasting". In: *Omega* 14.2 (1986), pp. 93–98. ISSN: 0305-0483. DOI: 10.1016/0305-0483(86)90013-7 (p. 85).

[133] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. “Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption”. In: *IEEE International Conference on Software Architecture*. 2017. DOI: 10.1109/ICSA.2017.24 (p. 197).

[134] João M. Franco, Francisco Correia, Raul Barbosa, Mário Zenha-Rela, Bradley Schmerl, and David Garlan. “Improving self-adaptation planning through software architecture-based stochastic modeling”. In: *Journal of Systems and Software* 115 (2016), pp. 42–60. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.01.026 (p. 2).

[135] Antonino Freno. “Practical Lessons from Developing a Large-Scale Recommender System at Zalando”. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys ’17. Como, Italy: Association for Computing Machinery, 2017, pp. 251–259. ISBN: 9781450346528. DOI: 10.1145/3109859.3109897 (pp. 152, 161, 234).

[136] Alexander Frömmgen, Robert Rehner, Max Lehn, and Alejandro Buchmann. “Fossa: Learning ECA Rules for Adaptive Distributed Systems”. In: *2015 IEEE International Conference on Autonomic Computing*. July 2015, pp. 207–210. DOI: 10.1109/ICAC.2015.37 (pp. 35, 49).

[137] Kun Fu, Fanlin Meng, Jieping Ye, and Zheng Wang. “CompactETA: A Fast Inference System for Travel Time Prediction”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 3337–3345. ISBN: 9781450379984. DOI: 10.1145/3394486.3403386 (pp. 138, 161, 233).

[138] Burkhardt Funk. “Optimizing price levels in e-commerce applications: An empirical study”. In: 2009, pp. 37–43. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-74549181430%5C&partnerID=40%5C&md5=6dfdde67b807b3964c62fc8c1929dcf0 (pp. 161, 236).

[139] Matthias Galster and Danny Weyns. “Empirical Research in Software Architecture: How Far have We Come?” In: *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. Los Alamitos, CA, USA: IEEE Press, 2016, pp. 11–20. DOI: 10.1109/WICSA.2016.10 (p. 127).

[140] Alessio Gambi, Giovanni Toffetti, and Mauro Pezzè. “Assurance of Self-adaptive Controllers for the Cloud”. In: *Assurances for Self-Adaptive Systems: Principles, Models, and Techniques*. Ed. by Javier Cámara, Rogério de Lemos, Carlo Ghezzi, and Antónia Lopes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 311–339. ISBN: 978-3-642-36249-1. DOI: 10.1007/978-3-642-36249-1_12 (p. 28).

[141] Juan Cruz Gardey and Alejandra Garrido. "User Experience Evaluation through Automatic A/B Testing". In: *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*. IUI '20. Cagliari, Italy: Association for Computing Machinery, 2020, pp. 25–26. ISBN: 9781450375139. DOI: 10.1145/3379336.3381514 (pp. 5, 11, 171).

[142] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure". In: *Computer* 37.10 (Oct. 2004), pp. 46–54. ISSN: 0018-9162. DOI: 10.1109/MC.2004.175 (pp. 2, 4, 22, 24, 52).

[143] V. Geetha Lekshmy, P. A. Vishnu, and P. S. Harikrishnan. "Adaptive IoT System for Precision Agriculture". In: *Inventive Computation and Information Technologies*. Ed. by S. Smys, Valentina Emilia Balas, and Ram Palanisamy. Singapore: Springer Singapore, 2022, pp. 39–49. ISBN: 978-981-16-6723-7. DOI: 10.1007/978-981-16-6723-7_4 (p. 83).

[144] Simos Gerasimou, Radu Calinescu, and Alec Banks. "Efficient Runtime Quantitative Verification Using Caching, Lookahead, and Nearly-optimal Reconfiguration". In: *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS 2014. Hyderabad, India: ACM, 2014, pp. 115–124. ISBN: 978-1-4503-2864-7. DOI: 10.1145/2593929.2593932 (p. 56).

[145] Ilias Gerostathopoulos, Ali Naci Uysal, Christian Prehofer, and Tomas Bures. "A Tool for Online Experiment-Driven Adaptation". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2018, pp. 100–105. DOI: 10.1109/FAS-W.2018.00032 (p. 172).

[146] Ilias Gerostathopoulos, Dominik Skoda, Frantisek Plasil, Tomas Bures, and Alessia Knauss. "Architectural Homeostasis in Self-Adaptive Software-Intensive Cyber-Physical Systems". In: *Software Architecture*. Ed. by Bedir Tekinerdogan, Uwe Zdun, and Ali Babar. Cham: Springer International Publishing, 2016, pp. 113–128. ISBN: 978-3-319-48992-6. DOI: 10.1007/978-3-319-48992-6_8 (p. 43).

[147] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees". In: *Machine Learning* 63 (2006), pp. 3–42. DOI: 10.1007/s10994-006-6226-1 (p. 83).

[148] Sona Ghahremani, Christian M. Adriano, and Holger Giese. "Training Prediction Models for Rule-Based Self-Adaptive Systems". In: *2018 IEEE International Conference on Autonomic Computing (ICAC)*. Sept. 2018, pp. 187–192. DOI: 10.1109/ICAC.2018.00031 (pp. 31, 56).

[149] Sona Ghahremani, Holger Giese, and Thomas Vogel. “Efficient Utility-Driven Self-Healing Employing Adaptation Rules for Large Dynamic Architectures”. In: *2017 IEEE International Conference on Autonomic Computing (ICAC)*. 2017, pp. 59–68. DOI: 10.1109/ICAC.2017.35 (p. 56).

[150] Omid Gheibi and Danny Weyns. “Lifelong Self-Adaptation: Self-Adaptation Meets Lifelong Machine Learning”. In: *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 1–12. ISBN: 9781450393058. DOI: 10.1145/3524844.3528052 (p. 218).

[151] Omid Gheibi, Danny Weyns, and Federico Quin. “Applying Machine Learning in Self-Adaptive Systems: A Systematic Literature Review”. In: *ACM Trans. Auton. Adapt. Syst.* 15.3 (Aug. 2021). ISSN: 1556-4665. DOI: 10.1145/3469440 (pp. 7, 18, 21, 53, 210, 289).

[152] Omid Gheibi, Danny Weyns, and Federico Quin. “On the Impact of Applying Machine Learning in the Decision-Making of Self-Adaptive Systems”. In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 2021, pp. 104–110. DOI: 10.1109/SEAMS51251.2021.00023 (p. 290).

[153] Carlo Ghezzi. “The Fading Boundary between Development Time and Run Time”. In: *2011 IEEE Ninth European Conference on Web Services*. 2011, pp. 11–11. DOI: 10.1109/ECOWS.2011.33 (p. 52).

[154] Federico Giaimo, Hugo Andrade, and Christian Berger. “Continuous experimentation and the cyber–physical systems challenge: An overview of the literature and the industrial perspective”. In: *Journal of Systems and Software* 170 (2020), p. 110781. ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110781 (p. 129).

[155] Carlos A. Gomez-Uribe and Neil Hunt. “The Netflix Recommender System: Algorithms, Business Value, and Innovation”. In: *ACM Trans. Manage. Inf. Syst.* 6.4 (Dec. 2016). ISSN: 2158-656X. DOI: 10.1145/2843948 (pp. 161, 233).

[156] Anjan Goswami, Wei Han, Zhenrui Wang, and Angela Jiang. “Controlled experiments for decision-making in e-Commerce search”. In: *2015 IEEE International Conference on Big Data (Big Data)*. Los Alamitos, CA, USA: IEEE Press, 2015, pp. 1094–1102. DOI: 10.1109/BigData.2015.7363863 (pp. 124, 143, 154, 159, 161, 162, 164, 165, 170, 171, 234).

[157] Samuel Greengard. *The Internet of Things*. Massachusetts Institute of Technology Press, 2015. ISBN: 9780262527736. URL: https://mitpress.mit.edu/9780262527736/the-internet-of-things/ (p. 1).

[158] Alois Gruson, Praveen Chandar, Christophe Charbuillet, James McInerney, Samantha Hansen, Damien Tardieu, and Ben Carterette. "Offline Evaluation to Make Decisions About PlaylistRecommendation Algorithms". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. WSDM '19. Melbourne VIC, Australia: Association for Computing Machinery, 2019, pp. 420–428. ISBN: 9781450359405. DOI: 10.1145/3289600.3291027 (pp. 146, 158, 161, 162, 172, 233).

[159] Xiaodong Gu. "IDES: Self-adaptive Software with Online Policy Evolution Extended from Rainbow". In: *Computer and Information Science 2012*. Ed. by Roger Lee. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 181–195. ISBN: 978-3-642-30454-5. DOI: 10.1007/978-3-642-30454-5_13 (p. 31).

[160] Huan Gui, Ya Xu, Anmol Bhasin, and Jiawei Han. "Network A/B Testing: From Sampling to Estimation". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 399–409. ISBN: 9781450334693. DOI: 10.1145/2736277.2741081 (pp. 131, 158, 161, 162, 180, 231).

[161] Yongyi Guo, Dominic Coey, Mikael Konutgan, Wenting Li, Chris Schoener, and Matt Goldman. "Machine Learning for Variance Reduction in Online Experiments". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 8637–8648. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/488b084119a1c7a4950f00706ec7ea16-Paper.pdf (pp. 12, 173).

[162] Jayant Gupchup, Yasaman Hosseinkashi, Pavel Dmitriev, Daniel Schneider, Ross Cutler, Andrei Jefremov, and Martin Ellis. "Trustworthy Experimentation Under Telemetry Loss". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM '18. Torino, Italy: Association for Computing Machinery, 2018, pp. 387–396. ISBN: 9781450360142. DOI: 10.1145/3269206.3271747 (pp. 137, 161, 162, 233).

[163] Shubham Gupta and Sneha Chokshi. "Digital Marketing Effectiveness Using Incrementality". In: *Advances in Computing and Data Sciences*. Ed. by Mayank Singh, P. K. Gupta, Vipin Tyagi, Jan Flusser, Tuncer Ören, and Gianluca Valentino. Singapore: Springer Singapore, 2020, pp. 66–75. ISBN: 978-981-15-6634-9 (pp. 125, 140, 143, 161, 235).

[164] Somit Gupta et al. "Top Challenges from the First Practical Online Controlled Experiments Summit". In: *SIGKDD Explor. Newsl.* 21.1 (May 2019), pp. 20–35. ISSN: 1931-0145. DOI: 10.1145/3331651.3331655 (pp. 5, 11, 171, 199).

[165] Somit Gupta, Lucy Ulanova, Sumit Bhardwaj, Pavel Dmitriev, Paul Raff, and Aleksander Fabijan. "The Anatomy of a Large-Scale Experimentation Platform". In: *2018 IEEE International Conference on Software Architecture*

(ICSA). 2018, pp. 1–109. DOI: 10.1109/ICSA.2018.00009 (pp. 153, 161, 163, 234).

[166] MyungJoo Ham, Sangjung Woo, Jaeyun Jung, Wook Song, Gichan Jang, Yongjoo Ahn, and Hyoungjoo Ahn. “Toward Among-Device AI from on-Device AI with Stream Pipelines”. In: *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 285–294. ISBN: 9781450392266. DOI: 10.1145/3510457.3513026 (p. 1).

[167] Yan He and Miao Chen. “A Probabilistic, Mechanism-Indepedent Outlier Detection Method for Online Experimentation”. In: *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2017, pp. 640–647. DOI: 10.1109/DSAA.2017.64 (pp. 155, 161, 235).

[168] Yan He, Lin Yu, Miao Chen, William Choi, and Don Matheson. “A Cluster-Based Nearest Neighbor Matching Algorithm for Enhanced A/A Validation in Online Experimentation”. In: *Companion Proceedings of the Web Conference 2022*. WWW ’22. Virtual Event, Lyon, France: Association for Computing Machinery, 2022, pp. 136–140. ISBN: 9781450391306. DOI: 10.1145/3487553.3524220 (pp. 161, 162, 219, 232).

[169] Francis Heylighen. “The Science of Self-organization and Adaptivity”. In: *Knowledge Management, Organizational Intelligence and Learning, and Complexity: v. 3*. Ed. by L. D. Kiel. EOLSS Publishers Co Ltd, 2002. ISBN: 978-1-84826-913-2 (p. 22).

[170] Sara M. Hezavehi, Danny Weyns, Paris Avgeriou, Radu Calinescu, Raffaela Mirandola, and Diego Perez-Palacin. “Uncertainty in Self-Adaptive Systems: A Research Community Perspective”. In: *ACM Trans. Auton. Adapt. Syst.* 15.4 (Dec. 2021). ISSN: 1556-4665. DOI: 10.1145/3487921 (pp. 6, 14).

[171] Han Nguyen Ho and Eunseok Lee. “Model-Based Reinforcement Learning Approach for Planning in Self-Adaptive Software System”. In: *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*. IMCOM ’15. Bali, Indonesia: Association for Computing Machinery, 2015. ISBN: 9781450333771. DOI: 10.1145/2701126.2701191 (pp. 44, 45).

[172] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 1st. Illinois, IL, USA: Addison-Wesley Professional, 2010. ISBN: 0321601912 (pp. 9, 127).

[173] Pawel Idziak and Siobhán Clarke. “An Analysis of Decision-Making Techniques in Dynamic, Self-Adaptive Systems”. In: *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. Sept. 2014, pp. 137–143. DOI: 10.1109/SASOW.2014.23 (p. 55).

[174] M. Usman Iftikhar, Jonas Lundberg, and Danny Weyns. “A Model Interpreter for Timed Automata”. In: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*. Ed. by Tiziana Margaria and Bernhard Steffen. Cham: Springer International Publishing, 2016, pp. 243–258. ISBN: 978-3-319-47166-2. DOI: 10.1007/978-3-319-47166-2_17 (pp. 91, 106).

[175] M. Usman Iftikhar, Gowri Sankar Ramachandran, Pablo Bollansée, Danny Weyns, and Danny Hughes. “DeltaIoT: A Self-Adaptive Internet of Things Exemplar”. In: *IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2017, pp. 76–82. DOI: 10.1109/SEAMS.2017.21 (pp. 16, 86).

[176] M. Usman Iftikhar and Danny Weyns. “ActivFORMS: Active Formal Models for Self-Adaptation”. In: *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS 2014. Hyderabad, India: ACM, 2014, pp. 125–134. ISBN: 9781450328647. DOI: 10.1145/2593929.2593944 (pp. 4, 24, 53, 218).

[177] Azlan Ismail and Valeria Cardellini. “Decentralized Planning for Self-Adaptation in Multi-cloud Environment”. In: *Advances in Service-Oriented and Cloud Computing*. Ed. by Guadalupe Ortiz and Cuong Tran. Cham: Springer International Publishing, 2015, pp. 76–90. ISBN: 978-3-319-14886-1. DOI: 10.1007/978-3-319-14886-1_9 (p. 44).

[178] David Issa Mattos, Pavel Dmitriev, Aleksander Fabijan, Jan Bosch, and Helena Holmström Olsson. “An Activity and Metric Model for Online Controlled Experiments”. In: *Product-Focused Software Process Improvement*. Ed. by Marco Kuhrmann, Kurt Schneider, Dietmar Pfahl, Sousuke Amasaki, Marcus Ciolkowski, Regina Hebig, Paolo Tell, Jil Klünder, and Steffen Küpper. Cham: Springer International Publishing, 2018, pp. 182–198. ISBN: 978-3-030-03673-7 (p. 170).

[179] Pooyan Jamshidi, Javier Cámara, Bradley Schmerl, Christian Kästner, and David Garlan. “Machine Learning Meets Quantitative Planning: Enabling Self-Adaptation in Autonomous Robots”. In: *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’19. Montreal, Quebec, Canada: IEEE Press, May 2019, pp. 39–50. DOI: 10.1109/SEAMS.2019.00015 (pp. 6, 8, 54).

[180] Pooyan Jamshidi, Claus Pahl, and Nabor C. Mendonça. “Managing Uncertainty in Autonomic Cloud Elasticity Controllers”. In: *IEEE Cloud Computing* 3.3 (2016), pp. 50–60. DOI: 10.1109/MCC.2016.66 (p. 30).

[181] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. “Microservices: The Journey So Far and Challenges Ahead”. In: *IEEE Software* 35.3 (2018). DOI: 10.1109/MS.2018.2141039 (p. 198).

[182] Pooyan Jamshidi, Amir Sharifloo, Claus Pahl, Hamid Arabnejad, Andreas Metzger, and Giovani Estrada. “Fuzzy Self-Learning Controllers for Elasticity Management in Dynamic Cloud Architectures”. In: *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*. Apr. 2016, pp. 70–79. DOI: 10.1109/QoSA.2016.13 (pp. 4, 5, 27, 43, 44).

[183] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. “Learning to Sample: Exploiting Similarities across Environments to Learn Performance Models for Configurable Systems”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018. Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, pp. 71–82. ISBN: 9781450355735. DOI: 10.1145/3236024.3236074 (pp. 31, 83).

[184] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. “Transfer Learning for Improving Model Predictions in Highly Configurable Software”. In: *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 31–41. ISBN: 9781538615508. DOI: 10.1109/SEAMS.2017.11 (pp. 44, 45).

[185] Janne Järvinen, Tua Huomo, Tommi Mikkonen, and Pasi Tyrväinen. “From Agile Software Development to Mercury Business”. In: *Software Business. Towards Continuous Value Delivery*. Ed. by Casper Lassenius and Kari Smolander. Cham: Springer International Publishing, 2014, pp. 58–71. ISBN: 978-3-319-08738-2 (p. 197).

[186] Hao Jiang, Fan Yang, and Wutao Wei. “Statistical Reasoning of Zero-Inflated Right-Skewed User-Generated Big Data A/B Testing”. In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, pp. 1533–1544. DOI: 10.1109/BigData50022.2020.9377996 (pp. 144, 161, 163, 165, 234).

[187] Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. “Peeking at A/B Tests: Why It Matters, and What to Do about It”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’17. Halifax, NS, Canada: Association for Computing Machinery, Aug. 2017, pp. 1517–1525. ISBN: 9781450348874. DOI: 10.1145/3097983.3097992 (pp. 124, 152, 161, 232).

[188] Nianqiao Ju, Diane Hu, Adam Henderson, and Liangjie Hong. “A Sequential Test for Selecting the Better Variant: Online A/B Testing, Adaptive Allocation, and Continuous Monitoring”. In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. WSDM ’19. Melbourne VIC, Australia: Association for Computing Machinery, 2019, pp. 492–500. ISBN: 9781450359405. DOI: 10.1145/3289600.3291025 (pp. 137, 144, 158, 161–163, 232).

[189] Dhrgam AL-Kafaf, Dae-Kyoo Kim, and Lunjin Lu. “A three-phase decision making approach for self-adaptive systems using web services”. In: *Complex Adaptive Systems Modeling* 6.1 (Oct. 2018). ISSN: 2194-3206. DOI: 10.1186/s40294-018-0059-1 (p. 43).

[190] Andrew Karpan. *The Internet of Things*. Current Controversies. MIT Press, 2021. ISBN: 9781534507746. URL: https://books.google.be/books?id=joReEAAAQBAJ (p. 1).

[191] Hadeel El-Kassabi, Mohamed Adel Serhani, Salah Bouktif, and Abdelghani Benharref. “Multi-Model Deep Learning for Cloud Resources Prediction to Support Proactive Workflow Adaptation”. In: *2019 IEEE Cloud Summit*. 2019, pp. 78–85. DOI: 10.1109/CloudSummit47114.2019.00019 (pp. 31, 56).

[192] Staffs Keele et al. *Guidelines for performing systematic literature reviews in software engineering*. Tech. rep. Technical report, Ver. 2.3 EBSE Technical Report. EBSE, 2007 (pp. 14, 23, 124, 130).

[193] Jeffrey O. Kephart and David M. Chess. “The Vision of Autonomic Computing”. In: *Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055 (pp. 1, 2, 4, 5, 22, 24, 52, 60, 174, 204).

[194] Manzoor Ahmed Khan and Hamidou Tembine. “Meta-Learning for Realizing Self-x Management of Future Networks”. In: *IEEE Access* 5 (2017), pp. 19072–19083. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2745999 (p. 37).

[195] Eugene Kharitonov, Alexey Drutsa, and Pavel Serdyukov. “Learning Sensitive Combinations of A/B Test Metrics”. In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM ’17. Cambridge, United Kingdom: Association for Computing Machinery, 2017, pp. 651–659. ISBN: 9781450346757. DOI: 10.1145/3018661.3018708 (pp. 131, 161, 162, 164, 173, 232).

[196] Eugene Kharitonov, Aleksandr Vorobev, Craig Macdonald, Pavel Serdyukov, and Iadh Ounis. “Sequential Testing for Early Stopping of Online Experiments”. In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’15. Santiago, Chile: Association for Computing Machinery, 2015, pp. 473–482. ISBN: 9781450336215. DOI: 10.1145/2766462.2767729 (pp. 139, 161, 162, 165, 232).

[197] Rochelle King, Elizabeth F. Churchill, and Caitlin Tan. *Designing with Data: Improving the User Experience with A/B Testing*. Sebastopol, CA, USA: O’Reilly Media, Inc., 2017. ISBN: 9781449334833 (pp. 131, 170).

[198] Barbara Kitchenham. “Procedures for Performing Systematic Reviews”. In: *Keele, UK, Keele Univ.* 33 (Aug. 2004) (p. 14).

[199] Ron Kohavi, Alex Deng, Brian Frasca, Roger Longbotham, Toby Walker, and Ya Xu. "Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained". In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. Beijing, China: Association for Computing Machinery, 2012, pp. 786–794. ISBN: 9781450314626. DOI: 10.1145/2339530.2339653 (pp. 161, 162, 231).

[200] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. "Online Controlled Experiments at Large Scale". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: Association for Computing Machinery, 2013, pp. 1168–1176. ISBN: 9781450321747. DOI: 10.1145/2487575.2488217 (pp. 144, 154, 161, 170, 209, 231).

[201] Ron Kohavi, Alex Deng, Roger Longbotham, and Ya Xu. "Seven Rules of Thumb for Web Site Experimenters". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 1857–1866. ISBN: 9781450329569. DOI: 10.1145/2623330.2623341 (pp. 140, 161, 235).

[202] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. "Practical Guide to Controlled Experiments on the Web: Listen to Your Customers Not to the Hippo". In: *13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Jose, California, USA, 2007, pp. 959–967. ISBN: 9781595936097. DOI: 10.1145/1281192.1281295 (pp. 5, 9, 11, 170, 197).

[203] Ron Kohavi and Roger Longbotham. "Online Controlled Experiments and A/B Testing". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer, 2017, pp. 922–929. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_891 (pp. 9, 124, 131, 197).

[204] Ron Kohavi and Roger Longbotham. "Unexpected Results in Online Controlled Experiments". In: *SIGKDD Explor. Newsl.* 12.2 (Mar. 2011), pp. 31–35. ISSN: 1931-0145. DOI: 10.1145/1964897.1964905 (pp. 161, 231).

[205] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal Henne. "Controlled experiments on the web: Survey and practical guide". In: *Data Mining and Knowledge Discovery* 18 (Feb. 2009), pp. 140–181. DOI: 10.1007/s10618-008-0114-1 (p. 131).

[206] Ron Kohavi, Diane Tang, and Ya Xu. *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing*. Cambridge, United Kingdom: Cambridge University Press, Mar. 2020. ISBN: 9781108724265. DOI: 10.1017/9781108653985 (p. 127).

[207] Anastasiia Kornilova and Lucas Bernardi. "Mining the Stars: Learning Quality Ratings with User-Facing Explanations for Vacation Rentals". In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 976–983. ISBN: 9781450382977. DOI: 10.1145/3437963.3441812 (pp. 161, 233).

[208] Kostantinos Koukouvis, Roberto Alcañiz Cubero, and Patrizio Pelliccione. "A/B Testing in E-commerce Sales Processes". In: *Software Engineering for Resilient Systems*. Ed. by Ivica Crnkovic and Elena Troubitsyna. Cham: Springer International Publishing, 2016, pp. 133–148. ISBN: 978-3-319-45892-2 (pp. 139, 152, 154, 161–163, 235).

[209] David Kramer and Wolfgang Karl. "Realizing a Proactive, Self-Optimizing System Behavior within Adaptive, Heterogeneous Many-Core Architectures". In: *2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*. Sept. 2012, pp. 39–48. DOI: 10.1109/SASO.2012.26 (pp. 8, 35).

[210] Jeff Kramer and Jeff Magee. "Self-Managed Systems: an Architectural Challenge". In: June 2007, pp. 259–268. ISBN: 0-7695-2829-5. DOI: 10.1109/FOSE.2007.19 (pp. 2, 24, 52).

[211] Christian Krupitzer, Julian Otto, Felix Maximilian Roth, Alexander Frömmgen, and Christian Becker. "Adding Self-Improvement to an Autonomic Traffic Management System". In: *2017 IEEE International Conference on Autonomic Computing (ICAC)*. July 2017, pp. 209–214. DOI: 10.1109/ICAC.2017.16 (p. 31).

[212] Christian Krupitzer, Martin Pfannemüller, Jean Kaddour, and Christian Becker. "SATISFy: Towards a Self-Learning Analyzer for Time Series Forecasting in Self-Improving Systems". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. Sept. 2018, pp. 182–189. DOI: 10.1109/FAS-W.2018.00045 (p. 43).

[213] Anuj Kumar and Kartik Hosanagar. "Measuring the Value of Recommendation Links on Product Demand". In: *SSRN Electronic Journal* (Jan. 2017). DOI: 10.2139/ssrn.2909971 (pp. 161, 235).

[214] Ranjitha Kumar. "Data-Driven Design: Beyond A/B Testing". In: *Proceedings of the 2019 Conference on Human Information Interaction and Retrieval*. CHIIR '19. Glasgow, Scotland UK: Association for Computing Machinery, 2019, pp. 1–2. ISBN: 9781450360258. DOI: 10.1145/3295750.3300046 (p. 171).

[215] Ratnakar Kumar and Nitasha Hasteer. “Evaluating usability of a web application: A comparative analysis of open-source tools”. In: *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*. 2017, pp. 350–354. DOI: 10.1109/CESYS.2017.8321296 (pp. 161, 234).

[216] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. “Adversarial Machine Learning at Scale”. In: 2017. URL: https://arxiv.org/abs/1611.01236 (p. 47).

[217] Ilja Kuzborskij, Francesco Orabona, and Barbara Caputo. “From N to N+1: Multiclass Transfer Incremental Learning”. In: June 2013, pp. 3358–3365. DOI: 10.1109/CVPR.2013.431 (p. 80).

[218] Mounia Lalmas, Janette Lehmann, Guy Shaked, Fabrizio Silvestri, and Gabriele Tolomei. “Promoting Positive Post-Click Experience for In-Stream Yahoo Gemini Users”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’15. Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 1929–1938. ISBN: 9781450336642. DOI: 10.1145/2783258.2788581 (pp. 5, 161, 233).

[219] Nicholas Larsen, Jonathan Stallrich, Srijan Sengupta, Alex Deng, Ron Kohavi, and Nathaniel Stevens. “Statistical Challenges in Online Controlled Experiments: A Review of A/B Testing Methodology”. In: *arXiv preprint arXiv:2212.11366* (2023). DOI: 10.48550/arXiv.2212.11366 (p. 170).

[220] Euijong Lee, Young-Duk Seo, and Young-Gab Kim. “A Nash equilibrium based decision-making method for internet of things”. In: *Journal of Ambient Intelligence and Humanized Computing* (June 2019), pp. 1–9. ISSN: 1868-5145. DOI: 10.1007/s12652-019-01367-2 (p. 37).

[221] Minyong R. Lee and Milan Shen. “Winner’s Curse: Bias Estimation for Total Effects of Features in Online Controlled Experiments”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 491–499. ISBN: 9781450355520. DOI: 10.1145/3219819.3219905 (pp. 161, 163, 232).

[222] Rogério de Lemos, David Garlan, Carlo Ghezzi, Holger Giese, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Danny Weyns, Luciano Baresi, Nelly Bencomo, Yuriy Brun, Javier Camara, Radu Calinescu, Myra B. Cohen, Alessandra Gorla, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jezequel, Sam Malek, Raffaela Mirandola, Marco Mori, Hausi A. Müller, Romain Rouvoy, Cecília M. F. Rubira, Eric Rutten, Mary Shaw, Giordano Tamburrelli, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, and Franco Zambonelli. “Software Engineering for Self-Adaptive Systems: Research Challenges in the Provision of Assurances”. In: *Software Engineering for Self-Adaptive Systems III. Assurances*. Ed. by Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese. Cham: Springer International Publishing, 2017, pp. 3–30. ISBN: 978-3-319-74183-3 (p. 174).

[223] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Rick Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap". In: *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Ed. by Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32. ISBN: 978-3-642-35813-5. DOI: 10.1007/978-3-642-35813-5_1 (pp. 2, 174, 217).

[224] Florian Lettner, Clemens Holzmann, and Patrick Hutflesz. "Enabling A/B Testing of Native Mobile Applications by Remote User Interface Exchange". In: *Computer Aided Systems Theory - EUROCAST 2013*. Ed. by Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 458–466. ISBN: 978-3-642-53862-9 (pp. 161, 162, 236).

[225] Chengbo Li, Lin Zhu, Guangyuan Fu, Longzhi Du, Canhua Zhao, Tianlun Ma, Chang Ye, and Pei Lee. "Learning to Bundle Proactively for On-Demand Meal Delivery". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, pp. 3898–3905. ISBN: 9781450384469. DOI: 10.1145/3459637.3481931 (pp. 140, 146, 161, 233).

[226] Hannah Li, Geng Zhao, Ramesh Johari, and Gabriel Y. Weintraub. "Interference, Bias, and Variance in Two-Sided Marketplace Experimentation: Guidance for Platforms". In: *Proceedings of the ACM Web Conference 2022*. WWW '22. Virtual Event, Lyon, France: Association for Computing Machinery, 2022, pp. 182–192. ISBN: 9781450390965. DOI: 10.1145/3485447.3512063 (pp. 140, 161, 162, 234).

[227] Lihong Li, Jin Young Kim, and Imed Zitouni. "Toward Predicting the Outcome of an A/B Experiment for Search Relevance". In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. WSDM '15. New York, NY, USA: Association for Computing Machinery, Feb. 2015, pp. 37–46. ISBN: 978-1-4503-3317-7. DOI: 10.1145/2684822.2685311 (pp. 12, 124, 170, 173).

[228] Paul Luo Li, Xiaoyu Chai, Frederick Campbell, Jilong Liao, Neeraja Abburu, Minsuk Kang, Irina Niculescu, Greg Brake, Siddharth Patil, James Dooley, and Brandon Paddock. "Evolving Software to be ML-Driven Utilizing Real-World A/B Testing: Experiences, Insights, Challenges". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2021, pp. 170–179. DOI: 10.1109/ICSE-SEIP52600.2021.00026 (pp. 142, 152, 155, 161, 163, 232).

[229] Paul Luo Li, Pavel Dmitriev, Huibin Mary Hu, Xiaoyu Chai, Zoran Dimov, Brandon Paddock, Ying Li, Alex Kirshenbaum, Irina Niculescu, and Taj Thoresen. "Experimentation in the Operating System: The Windows Experimentation Platform". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Los Alamitos, CA, USA: IEEE Press, 2019, pp. 21–30. DOI: 10.1109/ICSE-SEIP.2019.00011 (pp. 124, 146, 154, 161, 162, 170, 232).

[230] Ye Li, Hong Xie, Yishi Lin, and John C.S. Lui. "Unifying Offline Causal Inference and Online Bandit Learning for Data Driven Decision". In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 2291–2303. ISBN: 9781450383127. DOI: 10.1145/3442381.3449982 (pp. 161, 172, 233).

[231] Yiyang Li, Guanyu Tao, Weinan Zhang, Yong Yu, and Jun Wang. "Content Recommendation by Noise Contrastive Transfer Learning of Feature Representation". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM '17. Singapore, Singapore: Association for Computing Machinery, 2017, pp. 1657–1665. ISBN: 9781450349185. DOI: 10.1145/3132847.3132855 (pp. 161, 233).

[232] Kevin Liou and Sean J. Taylor. "Variance-Weighted Estimators to Improve Sensitivity in Online Experiments". In: *Proceedings of the 21st ACM Conference on Economics and Computation*. EC '20. Virtual Event, Hungary: Association for Computing Machinery, 2020, pp. 837–850. ISBN: 9781450379755. DOI: 10.1145/3391403.3399542 (pp. 137, 144, 160, 161, 232).

[233] Sophia Liu, Aleksander Fabijan, Michael Furchtgott, Somit Gupta, Pawel Janowski, Wen Qin, and Pavel Dmitriev. "Enterprise-Level Controlled Experiments at Scale: Challenges and Solutions". In: *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2019, pp. 29–37. DOI: 10.1109/SEAA.2019.00013 (pp. 139, 144, 146, 154, 161, 162, 165, 234).

[234] Teng Liu, Chao Yang, Chuanzheng Hu, Hong Wang, Li Li, Dongpu Cao, and Fei-Yue Wang. "Reinforcement Learning-Based Predictive Control for Autonomous Electrified Vehicles". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. June 2018, pp. 185–190. DOI: 10.1109/IVS.2018.8500719 (pp. 11, 31, 55).

[235] Yang Liu, Di Bai, and Wenpin Jiao. “Generating Adaptation Rules of Software Systems: A Method Based on Genetic Algorithm”. In: *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*. ICMLC 2018. Macau, China: Association for Computing Machinery, 2018, pp. 347–356. ISBN: 9781450363532. DOI: 10.1145/3195106.3195137 (p. 43).

[236] Yuchu Liu, David Issa Mattos, Jan Bosch, Helena Holmström Olsson, and Jonn Lantz. “Size matters? Or not: A/B testing with limited sample in automotive embedded software”. In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2021, pp. 300–307. DOI: 10.1109/SEAA53835.2021.00046 (pp. 160, 161, 234).

[237] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. “A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments”. In: *Journal of Grid Computing* 12.4 (Dec. 2014), pp. 559–592. ISSN: 1572-9184. DOI: 10.1007/s10723-014-9314-7 (p. 28).

[238] Widad Machmouchi, Ahmed Hassan Awadallah, Imed Zitouni, and Georg Buscher. “Beyond Success Rate: Utility as a Search Quality Metric for Online Experiments”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM ’17. Singapore, Singapore: Association for Computing Machinery, 2017, pp. 757–765. ISBN: 9781450349185. DOI: 10.1145/3132847.3132850 (pp. 137, 139, 146, 154, 161, 162, 170, 232).

[239] Lech Madeyski, Wojciech Orzeszyna, Richard Torkar, and Mariusz Józala. “Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation”. In: *IEEE Transactions on Software Engineering* 40.1 (2014), pp. 23–42. DOI: 10.1109/TSE.2013.44 (p. 134).

[240] Maria Madlberger and Jiri Jizdny. “Impact of promotional social media content on click-through rate - Evidence from a FMCG company”. In: 2021, pp. 3–10. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85124068035%5C&partnerID=40%5C&md5=c0b8f49a3b48b3d561fd0ed305eb1895 (pp. 158, 161, 163, 235).

[241] Martina Maggio, Henry Hoffmann, Alessandro V. Papadopoulos, Jacopo Panerati, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. “Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems”. In: *ACM Trans. Auton. Adapt. Syst.* 7.4 (Dec. 2012). ISSN: 1556-4665. DOI: 10.1145/2382570.2382572 (pp. 30, 43).

[242] Sara Mahdavi-Hezavehi, Vinicius H.S. Durelli, Danny Weyns, and Paris Avgeriou. “A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems”. In: *Information and Software Technology* 90 (2017), pp. 1–26. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2017.03.013 (pp. 52, 134).

[243] Mohammad Masdari and Afsane Khoshnevis. "A survey and classification of the workload forecasting methods in cloud computing". In: *Cluster Computing* 23.4 (Dec. 2020), pp. 2399–2424. ISSN: 1573-7543. DOI: 10.1007/s10586-019-03010-3 (p. 28).

[244] Taisei Masuda, Kyoko Murakami, Kenkichi Sugiura, Sho Sakui, Ron Philip Schuring, and Mitsuhiro Mori. "A phase 1/2 randomised placebo-controlled study of the COVID-19 vaccine mRNA-1273 in healthy Japanese adults: An interim report". In: *Vaccine* 40.13 (2022), pp. 2044–2052. ISSN: 0264-410X. DOI: 10.1016/j.vaccine.2022.02.030 (p. 127).

[245] David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. "More for Less: Automated Experimentation in Software-Intensive Systems". In: *Product-Focused Software Process Improvement*. Ed. by Michael Felderer, Daniel Méndez Fernández, Burak Turhan, Marcos Kalinowski, Federica Sarro, and Dietmar Winkler. Cham: Springer International Publishing, 2017, pp. 146–161. ISBN: 978-3-319-69926-4 (pp. 11, 161, 162, 172, 235).

[246] David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. "Challenges and Strategies for Undertaking Continuous Experimentation to Embedded Systems: Industry and Research Perspectives". In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Juan Garbajosa, Xiaofeng Wang, and Ademar Aguiar. Cham: Springer International Publishing, 2018, pp. 277–292. ISBN: 978-3-319-91602-6. DOI: 10.1007/978-3-319-91602-6_20 (pp. 129, 218).

[247] David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. "Your System Gets Better Every Day You Use It: Towards Automated Continuous Experimentation". In: *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2017, pp. 256–265. DOI: 10.1109/SEAA.2017.15 (pp. 11, 138, 159, 161, 162, 164, 171, 235).

[248] David Issa Mattos, Jan Bosch, Helena Holmstrom Olsson, Aita Maryam Korshani, and Jonn Lantz. "Automotive A/B testing: Challenges and Lessons Learned from Practice". In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2020, pp. 101–109. DOI: 10.1109/SEAA51224.2020.00026 (pp. 161, 162, 234).

[249] Pavel Metrikov, Fernando Diaz, Sebastien Lahaie, and Justin Rao. "Whole Page Optimization: How Page Elements Interact with the Position Auction". In: *Proceedings of the Fifteenth ACM Conference on Economics and Computation*. EC '14. Palo Alto, California, USA: Association for Computing Machinery, 2014, pp. 583–600. ISBN: 9781450325653. DOI: 10.1145/2600057.2602871 (pp. 140, 151, 155, 161, 233).

[250] Andreas Metzger, Clément Quinton, Zoltan Mann, Luciano Baresi, and Klaus Pohl. "Feature-Model-Guided Online Learning for Self-Adaptive Systems". In: *ArXiv* abs/1907.09158 (2019) (p. 54).

[251] Andreas Metzger, Clément Quinton, Zoltán ádám Mann, Luciano Baresi, and Klaus Pohl. “Feature Model-Guided Online Reinforcement Learning for Self-Adaptive Services”. In: *Service-Oriented Computing: 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings*. Dubai, United Arab Emirates: Springer-Verlag, 2020, pp. 269–286. ISBN: 978-3-030-65309-5. DOI: 10.1007/978-3-030-65310-1_20 (pp. 5, 8, 55).

[252] Mathias Meyer. “Continuous Integration and Its Tools”. In: *IEEE Software* 31.03 (May 2014), pp. 14–16. ISSN: 1937-4194. DOI: 10.1109/MS.2014.58 (pp. 9, 197).

[253] Risto Miikulainen, Myles Brundage, Jonathan Epstein, Tyler Foster, Babak Hodjat, Neil Iscoe, Jingbo Jiang, Diego Legrand, Sam Nazari, Xin Qiu, Michael Scharff, Cory Schoolland, Robert Severn, and Aaron Shagrin. “Ascend by Evolv: AI-Based Massively Multivariate Conversion Rate Optimization”. In: *AI Magazine* 41.1 (Apr. 2020), pp. 44–60. DOI: 10.1609/aimag.v41i1.5256 (pp. 161, 163, 235).

[254] Alok Mishra and Ziadoon Otaiwi. “DevOps and software quality: A systematic mapping”. In: *Computer Science Review* 38 (2020), p. 100308. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2020.100308 (p. 197).

[255] Tom M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: https://books.google.be/books?id=EoYBngEACAAJ (p. 24).

[256] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. “Adaptive Runtime Response Time Control in PLC-Based Real-Time Systems Using Reinforcement Learning”. In: *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 217–223. ISBN: 9781450357159. DOI: 10.1145/3194133.3194153 (p. 44).

[257] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. “Flexible and efficient decision-making for proactive latency-aware self-adaptation”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 13.1 (2018), pp. 1–36. DOI: 10.1145/3149180 (p. 56).

[258] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. “Proactive Self-Adaptation under Uncertainty: A Probabilistic Model Checking Approach”. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, pp. 1–12. ISBN: 9781450336758. DOI: 10.1145/2786805.2786853 (pp. 4, 6).

[259] Gabriel A. Moreno, Bradley Schmerl, and David Garlan. “SWIM: An Exemplar for Evaluation and Comparison of Self-Adaptation Approaches for Web Applications”. In: *13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2018 (p. 199).

[260] Kenji Mori, Naoko Okubo, Yasushi Ueda, Masafumi Katahira, and Toshiyuki Amagasa. “Supporting Viewpoints to Review the Lack of Requirements in Space Systems with Machine Learning”. In: *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 38–44. ISBN: 9781450379625. DOI: 10.1145/3387939.3391610 (p. 95).

[261] Andrés Muñoz Medina, Sergei Vassilvitskii, and Dong Yin. “Online Learning for Non-Stationary A/B Tests”. In: *ACM International Conference on Information and Knowledge Management*. 2018. ISBN: 9781450360142. DOI: 10.1145/3269206.3271718 (p. 172).

[262] Tadashi Okoshi, Kota Tsubouchi, and Hideyuki Tokuda. “Real-World Product Deployment of Adaptive Push Notification Scheduling on Smartphones”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2792–2800. ISBN: 9781450362016. DOI: 10.1145/3292500.3330732 (pp. 161, 233).

[263] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, D. Heimhigner, G. Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. “An architecture-based approach to self-adaptive software”. In: *IEEE Intelligent Systems and their Applications* 14.3 (1999), pp. 54–62. DOI: 10.1109/5254.769885 (pp. 2, 4, 5, 52).

[264] Takumi Ozawa, Akiyuki Sekiguchi, and Kazuhiko Tsuda. “A Method for the Construction of User Targeting Knowledge for B2B Industry Website”. In: *Procedia Computer Science* 96 (2016). Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 20th International Conference KES-2016, pp. 1147–1155. ISSN: 1877-0509. DOI: 10.1016/j.procs.2016.08.157 (pp. 161, 235).

[265] Alexander Palm, Andreas Metzger, and Klaus Pohl. “Online Reinforcement Learning for Self-adaptive Information Systems”. In: *Advanced Information Systems Engineering*. Ed. by Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant. Cham: Springer International Publishing, 2020, pp. 169–184. ISBN: 978-3-030-49435-3. DOI: 10.1007/978-3-030-49435-3_11 (p. 8).

[266] Ashutosh Pandey, Bradley Schmerl, and David Garlan. "Instance-Based Learning for Hybrid Planning". In: *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. Sept. 2017, pp. 64–69. DOI: 10.1109/FAS-W.2017.122 (pp. 35, 44).

[267] Dimitrios Papamartzivanos, Félix Gómez Mármol, and Georgios Kambourakis. "Introducing Deep Learning Self-Adaptive Misuse Network Intrusion Detection Systems". In: *IEEE Access* 7 (2019), pp. 13546–13560. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2893871 (pp. 31, 35, 43, 47).

[268] H. Van Dyke Parunak and Sven A. Brueckner. "Software engineering for self-organizing systems". In: *The Knowledge Engineering Review* 30.4 (2015), pp. 419–434. DOI: 10.1017/S0269888915000089 (p. 22).

[269] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, M. Perrot, and Edouard Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (pp. 83, 88, 188).

[270] Alejandro Pelaez, Andres Quiroz, and Manish Parashar. "Dynamic adaptation of policies using machine learning". In: *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE. 2016, pp. 501–510. DOI: 10.1109/CCGrid.2016.64 (p. 44).

[271] Dan Pelleg, Oleg Rokhlenko, Idan Szpektor, Eugene Agichtein, and Ido Guy. "When the Crowd is Not Enough: Improving User Experience with Social Media through Automatic Quality Analysis". In: *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. CSCW '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1080–1090. ISBN: 9781450335928. DOI: 10.1145/2818048.2820022 (pp. 152, 161, 235).

[272] Ladislav Peska and Peter Vojtas. "Off-Line vs. On-Line Evaluation of Recommender Systems in Small E-Commerce". In: *Proceedings of the 31st ACM Conference on Hypertext and Social Media*. HT '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 291–300. ISBN: 9781450370981. DOI: 10.1145/3372923.3404781 (pp. 152, 157, 161, 162, 234).

[273] Barry Porter and Roberto Rodrigues Filho. "Losing Control: The Case for Emergent Software Systems Using Autonomous Assembly, Perception, and Learning". In: *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. Sept. 2016, pp. 40–49. DOI: 10.1109/SASO.2016.10 (pp. 11, 55).

[274] Alexey Poyarkov, Alexey Drutsa, Andrey Khalyavin, Gleb Gusev, and Pavel Serdyukov. “Boosted Decision Tree Regression Adjustment for Variance Reduction in Online Controlled Experiments”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 235–244. ISBN: 9781450342322. DOI: 10.1145/2939672.2939688 (pp. 12, 139, 161, 162, 164, 170, 173, 231).

[275] Lara Qasim, Marija Jankovic, Sorin Olaru, and Jean-Luc Garnier. “Model-Based System Reconfiguration: A Descriptive Study of Current Industrial Challenges”. In: *Complex Systems Design & Management*. Ed. by Eric Bonjour, Daniel Krob, Luca Palladino, and François Stephan. Cham: Springer International Publishing, 2019, pp. 97–108. ISBN: 978-3-030-04209-7 (p. 6).

[276] Wenyi Qian, Xin Peng, Bihuan Chen, John Mylopoulos, Huanhuan Wang, and Wenyun Zhao. “Rationalism with a dose of empiricism: combining goal reasoning and case-based reasoning for self-adaptive software systems”. In: *Requirements Engineering* 20.3 (Sept. 2015), pp. 233–252. ISSN: 1432-010X. DOI: 10.1007/s00766-015-0227-1 (pp. 43, 44).

[277] Xiulei Qin, Wei Wang, Wenbo Zhang, Jun Wei, Xin Zhao, and Tao Huang. “Elasticat: A load rebalancing framework for cloud-based key-value stores”. In: *2012 19th International Conference on High Performance Computing*. 2012, pp. 1–10. DOI: 10.1109/HiPC.2012.6507481 (p. 43).

[278] Xiulei Qin, Wei Wang, Wenbo Zhang, Jun Wei, Xin Zhao, Hua Zhong, and Tao Huang. “PRESC2: efficient self-reconfiguration of cache strategies for elastic caching platforms”. In: *Computing* 96.5 (May 2014), pp. 415–451. ISSN: 1436-5057. DOI: 10.1007/s00607-013-0365-6 (p. 44).

[279] Jia Qu and Jing Zhang. “Validating Mobile Designs with Agile Testing in China: Based on Baidu Map for Mobile”. In: *Design, User Experience, and Usability: Design Thinking and Methods*. Ed. by Aaron Marcus. Cham: Springer International Publishing, 2016, pp. 491–498. ISBN: 978-3-319-40409-7 (pp. 161, 235).

[280] Federico Quin. “Systematic Approach to Engineer Decentralized Self-adaptive Systems”. In: *Software Architecture*. Ed. by Henry Muccini, Paris Avgeriou, Barbora Buhnova, Javier Camara, Mauro Caporuscio, Mirco Franzago, Anne Koziolek, Patrizia Scandurra, Catia Trubiani, Danny Weyns, and Uwe Zdun. Cham: Springer International Publishing, 2020, pp. 38–50. ISBN: 978-3-030-59155-7 (p. 290).

[281] Federico Quin and Danny Weyns. “Automating Pipelines of A/B Tests with Population Split Using Self-Adaptation and Machine Learning”. In: *arXiv preprint arXiv:2306.01407*. 2023. DOI: 10.48550/arXiv.2306.01407 (pp. 17, 18, 169, 290).

[282] Federico Quin and Danny Weyns. *SEAByTE Website*. https://people.cs.kuleuven.be/danny.weyns/software/SEAByTE/ (pp. 194, 200, 209, 210).

[283] Federico Quin and Danny Weyns. "SEAByTE: A Self-Adaptive Micro-Service System Artifact for Automating A/B Testing". In: *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 77–83. ISBN: 9781450393058. DOI: 10.1145/3524844.3528081 (pp. 17, 19, 182, 196, 290).

[284] Federico Quin, Danny Weyns, Thomas Bamelis, Sarpreet Singh Buttar, and Sam Michiels. "Efficient Analysis of Large Adaptation Spaces in Self-Adaptive Systems Using Machine Learning". In: *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '19. Montreal, Quebec, Canada: IEEE Press, 2019, pp. 1–12. DOI: 10.1109/SEAMS.2019.00011 (pp. 27, 31, 34, 44, 45, 53, 55, 93, 210, 289).

[285] Federico Quin, Danny Weyns, and Matthias Galster. *Study Systematic Literature Review on A/B Testing*. 2023. URL: https://people.cs.kuleuven.be/danny.weyns/material/SLR_AB/ (pp. 130, 131).

[286] Federico Quin, Danny Weyns, Matthias Galster, and Camila Costa Silva. "A/B Testing: A Systematic Literature Review". In: *arXiv preprint arXiv:2308.04929* (2023). DOI: 10.48550/arXiv.2308.04929 (pp. 18, 123, 171, 290).

[287] Federico Quin, Danny Weyns, and Omid Gheibi. "Decentralized Self-Adaptive Systems: A Mapping Study". In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 2021, pp. 18–29. DOI: 10.1109/SEAMS51251.2021.00014 (p. 290).

[288] Federico Quin, Danny Weyns, and Omid Gheibi. *Full reproduction package Reducing Large Adaptation Spaces in Self-Adaptive Systems Using Machine Learning*. https://people.cs.kuleuven.be/danny.weyns/material/ML4SAS/ML2ASR/ (p. 120).

[289] Federico Quin, Danny Weyns, and Omid Gheibi. "Reducing large adaptation spaces in self-adaptive systems using classical machine learning". In: *Journal of Systems and Software* 190 (2022), p. 111341. ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111341 (pp. 16, 18, 51, 289).

[290] Jan Renz, Daniel Hoffmann, Thomas Staubitz, and Christoph Meinel. "Using A/B testing in MOOC environments". In: *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge*. LAK '16. New York, NY, USA: Association for Computing Machinery, Apr. 2016, pp. 304–313. ISBN: 978-1-4503-4190-5. DOI: 10.1145/2883851.2883876 (p. 127).

[291] Ádám Révész and Norbert Pataki. “A/B Testing via Continuous Integration and Continuous Delivery”. In: *Geoinformatics and Data Analysis*. Ed. by Salah Bourennane and Petr Kubicek. Cham: Springer International Publishing, 2022, pp. 165–174. ISBN: 978-3-031-08017-3 (pp. 11, 172).

[292] Maxim Reynvoet, Omid Gheibi, Federico Quin, and Danny Weyns. “Detecting and Mitigating Jamming Attacks in IoT Networks Using Self-Adaptation”. In: *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. 2022, pp. 7–12. DOI: 10.1109/ACSOSC56246.2022.00019 (p. 290).

[293] Mohi Reza, Juho Kim, Ananya Bhattacharjee, Anna N. Rafferty, and Joseph Jay Williams. “The MOOClet Framework: Unifying Experimentation, Dynamic Improvement, and Personalization in Online Courses”. In: *Proceedings of the Eighth ACM Conference on Learning Scale*. LS ’21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 15–26. ISBN: 9781450382151. DOI: 10.1145/3430895.3460128 (pp. 138, 139, 161, 233).

[294] Barbara Riegel, Tiny Jaarsma, and Anna Strömberg. “A Middle-Range Theory of Self-Care of Chronic Illness”. In: *ANS. Advances in nursing science* 35 (June 2012), pp. 194–204. DOI: 10.1097/ANS.0b013e318261b1ba (p. 101).

[295] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586 (p. 83).

[296] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M. Verner, and Markku Oivo. “Continuous deployment of software intensive products and services: A systematic mapping study”. In: *Journal of Systems and Software* 123 (2017), pp. 263–291. ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.12.015 (pp. 9, 128, 197).

[297] Rasmus Ros and Per Runeson. “Continuous Experimentation and A/B Testing: A Mapping Study”. In: *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*. RCoSE ’18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 35–41. ISBN: 9781450357456. DOI: 10.1145/3194760.3194766 (pp. 5, 124, 128, 171, 197).

[298] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65 6 (1958), pp. 386–408 (p. 83).

[299] Nir Rosenfeld, Yishay Mansour, and Elad Yom-Tov. “Predicting Counterfactuals from Large Historical Data and Small Randomized Trials”. In: *Proceedings of the 26th International Conference on World Wide Web Companion*. WWW ’17 Companion. Perth, Australia: International World Wide Web Conferences

Steering Committee, 2017, pp. 602–609. ISBN: 9781450349147. DOI: 10.1145/3041021.3054190 (pp. 161, 162, 233).

[300] Sandra Sajeev, Jade Huang, Nikos Karampatziakis, Matthew Hall, Sebastian Kochman, and Weizhu Chen. "Contextual Bandit Applications in a Customer Support Bot". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD '21. Virtual Event, Singapore: Association for Computing Machinery, 2021, pp. 3522–3530. ISBN: 9781450383325. DOI: 10.1145/3447548.3467165 (pp. 11, 161, 170, 234).

[301] Mazeiar Salehie and Ladan Tahvildari. "Self-Adaptive Software: Landscape and Research Challenges". In: *ACM Trans. Auton. Adapt. Syst.* 4.2 (May 2009). DOI: 10.1145/1516533.1516538 (pp. 1, 4, 52).

[302] Felix Salfner and Miroslaw Malek. "Architecting Dependable Systems with Proactive Fault Management". In: *Architecting Dependable Systems VII*. Ed. by Antonio Casimiro, Rogério de Lemos, and Cristina Gacek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 171–200. ISBN: 978-3-642-17245-8. DOI: 10.1007/978-3-642-17245-8_8 (p. 43).

[303] Areeg Samir and Claus Pahl. "Self-Adaptive Healing for Containerized Cluster Architectures with Hidden Markov Models". In: *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. 2019, pp. 68–73. DOI: 10.1109/FMEC.2019.8795322 (p. 85).

[304] Theresia Ratih Dewi Saputri and Seok-Won Lee. "The Application of Machine Learning in Self-Adaptive Systems: A Systematic Literature Review". In: *IEEE Access* 8 (2020), pp. 205948–205967. DOI: 10.1109/ACCESS.2020.3036037 (p. 28).

[305] Suhrid Satyal, Ingo Weber, Hye-young Paik, Claudio Di Ciccio, and Jan Mendling. "AB-BPM: Performance-Driven Instance Routing for Business Process Improvement". In: *Business Process Management*. Ed. by Josep Carmona, Gregor Engels, and Akhil Kumar. Cham: Springer International Publishing, 2017, pp. 113–129. ISBN: 978-3-319-65000-5 (pp. 161, 235).

[306] Suhrid Satyal, Ingo Weber, Hye-young Paik, Claudio Di Ciccio, and Jan Mendling. "Business process improvement with the AB-BPM methodology". In: *Information Systems* 84 (2019), pp. 283–298. ISSN: 0306-4379. DOI: 10.1016/j.is.2018.06.007 (pp. 143, 161, 235).

[307] Martin Saveski, Jean Pouget-Abadie, Guillaume Saint-Jacques, Weitao Duan, Souvik Ghosh, Ya Xu, and Edoardo M. Airoldi. "Detecting Network Effects: Randomizing Over Randomized Experiments". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1027–1035. ISBN: 9781450348874. DOI: 10.1145/3097983.3098192 (pp. 161, 231).

[308] Gerald Schermann, Dominik Schöni, Philipp Leitner, and Harald C. Gall. "Bifrost: Supporting Continuous Deployment with Automated Enactment of Multi-Phase Live Testing Strategies". In: *Proceedings of the 17th International Middleware Conference*. Middleware '16. Trento, Italy: Association for Computing Machinery, 2016. ISBN: 9781450343008. DOI: 10.1145/2988336.2988348 (pp. 142, 148, 161–163, 172, 233).

[309] Burr Settles. *Active learning literature survey*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences, 2009 (pp. 47, 78).

[310] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014 (pp. 25, 38).

[311] Shahriar Shariat, Burkay Orten, and Ali Dasdan. "Online Evaluation of Bid Prediction Models in a Large-Scale Computational Advertising Platform: Decision Making and Insights". In: *Knowl. Inf. Syst.* 51.1 (Apr. 2017), pp. 37–60. ISSN: 0219-1377. DOI: 10.1007/s10115-016-0972-6 (pp. 143, 161, 235).

[312] Amir Molzam Sharifloo, Andreas Metzger, Clément Quinton, Luciano Baresi, and Klaus Pohl. "Learning and Evolution in Dynamic Software Product Lines". In: *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '16. Austin, Texas: Association for Computing Machinery, 2016, pp. 158–164. ISBN: 9781450341875. DOI: 10.1145/2897053.2897058 (p. 49).

[313] Sanaz Sheikhi and Seyed Morteza Babamir. "Using a Recurrent Artificial Neural Network for Dynamic Self-Adaptation of Cluster-Based Web-Server Systems". In: *Applied Intelligence* 48.8 (Aug. 2018), pp. 2097–2111. ISSN: 0924-669X. DOI: 10.1007/s10489-017-1059-0 (pp. 44, 45).

[314] Stepan Shevtsov, Mihaly Berekmeri, Danny Weyns, and Martina Maggio. "Control-Theoretical Software Adaptation: A Systematic Literature Review". In: *IEEE Transactions on Software Engineering* 44.8 (Aug. 2018), pp. 784–810. ISSN: 2326-3881. DOI: 10.1109/TSE.2017.2704579 (pp. 22, 119).

[315] Stepan Shevtsov and Danny Weyns. "Keep It SIMPLEX: Satisfying Multiple Goals with Guarantees in Control-Based Self-Adaptive Systems". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: Association for Computing Machinery, 2016, pp. 229–241. ISBN: 9781450342186. DOI: 10.1145/2950290.2950301 (p. 4).

[316] Stepan Shevtsov, Danny Weyns, and Martina Maggio. "SimCA*: A Control-Theoretic Approach to Handle Uncertainty in Self-Adaptive Systems with Guarantees". In: *ACM Trans. Auton. Adapt. Syst.* 13.4 (July 2019). ISSN: 1556-4665. DOI: 10.1145/3328730 (p. 217).

[317] Fanjuan Shi, Chirine Ghedira, and Jean-Luc Marini. “Context Adaptation for Smart Recommender Systems”. In: *IT Professional* 17.6 (2015), pp. 18–26. DOI: 10.1109/MITP.2015.96 (pp. 142, 146, 161, 234).

[318] Janet Siegmund, Norbert Siegmund, and Sven Apel. “Views on Internal and External Validity in Empirical Software Engineering”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. Los Alamitos, CA, USA: IEEE Press, 2015, pp. 9–19. DOI: 10.1109/ICSE.2015.24 (p. 127).

[319] Natalia Silberstein, Oren Somekh, Yair Koren, Michal Aharon, Dror Porat, Avi Shahar, and Tingyi Wu. “Ad Close Mitigation for Improved User Experience in Native Advertisements”. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. WSDM ’20. Houston, TX, USA: Association for Computing Machinery, 2020, pp. 546–554. ISBN: 9781450368223. DOI: 10.1145/3336191.3371798 (pp. 161, 234).

[320] Carlos Eduardo da Silva, José Diego Saraiva da Silva, Colin Paterson, and Radu Calinescu. “Self-Adaptive Role-Based Access Control for Business Processes”. In: *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2017. DOI: 10.1109/SEAMS.2017.13 (p. 22).

[321] Jorge Gabriel Siqueira and Melise M. V. de Paula. “IPEAD A/B Test Execution Framework”. In: *Proceedings of the XIV Brazilian Symposium on Information Systems*. SBSI’18. Caxias do Sul, Brazil: Association for Computing Machinery, 2018. ISBN: 9781450365598. DOI: 10.1145/3229345.3229360 (pp. 154, 161, 162, 232).

[322] Dan Siroker and Pete Koomen. *A/B Testing: The Most Powerful Way to Turn Clicks Into Customers*. 1st. Hoboken, NJ, USA: Wiley Publishing, 2013. ISBN: 1118536096 (pp. 125, 170).

[323] Kornel Skałkowski and Krzysztof Zieliński. “Automatic Adaptation of SOA Systems Supported by Machine Learning”. In: *Technological Innovation for the Internet of Things*. Ed. by Luis M. Camarinha-Matos, Slavisa Tomic, and Paula Graça. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 61–68. ISBN: 978-3-642-37291-9. DOI: 10.1007/978-3-642-37291-9_7 (p. 44).

[324] Matthias Sommer, Sven Tomforde, Jorg Hahner, and Dominik Auer. “Learning a Dynamic Re-combination Strategy of Forecast Techniques at Runtime”. In: *2015 IEEE International Conference on Autonomic Computing*. July 2015, pp. 261–266. DOI: 10.1109/ICAC.2015.70 (pp. 5, 31).

[325] Vítor E. Silva Souza, Alexei Lapouchnian, Konstantinos Angelopoulos, and John Mylopoulos. “Requirements-driven software evolution”. In: *Computer Science - Research and Development* 28.4 (Nov. 2013), pp. 311–329. ISSN: 1865-2042. DOI: 10.1007/s00450-012-0232-2 (p. 4).

[326] Bruce Spang, Veronica Hannan, Shravya Kunamalla, Te-Yuan Huang, Nick McKeown, and Ramesh Johari. "Unbiased Experiments in Congested Networks". In: *Proceedings of the 21st ACM Internet Measurement Conference*. IMC '21. Virtual Event: Association for Computing Machinery, 2021, pp. 80–95. ISBN: 9781450391290. DOI: 10.1145/3487552.3487851 (pp. 161, 232).

[327] Akshitha Sriraman, Abhishek Dhanotia, and Thomas F. Wenisch. "SoftSKU: Optimizing Server Architectures for Microservice Diversity Scale". In: *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 2019, pp. 513–526 (pp. 140, 161, 234).

[328] Anthony Stein, Sven Tomforde, Ada Diaconescu, Jörg Hähner, and Christian Müller-Schloer. "A Concept for Proactive Knowledge Construction in Self-Learning Autonomous Systems". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. Sept. 2018, pp. 204–213. DOI: 10.1109/FAS-W.2018.00048 (p. 44).

[329] Clay Stevens and Hamid Bagheri. "Reducing Run-Time Adaptation Space via Analysis of Possible Utility Bounds". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE '20. Seoul, South Korea: Association for Computing Machinery, 2020, pp. 1522–1534. ISBN: 9781450371216. DOI: 10.1145/3377811.3380365 (p. 55).

[330] Fei Sun, Peng Jiang, Hanxiao Sun, Changhua Pei, Wenwu Ou, and Xiaobo Wang. "Multi-Source Pointer Network for Product Title Summarization". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM '18. Torino, Italy: Association for Computing Machinery, 2018, pp. 7–16. ISBN: 9781450360142. DOI: 10.1145/3269206.3271722 (pp. 161, 233).

[331] Daniel Sykes, Domenico Corapi, Jeff Magee, Jeff Kramer, Alessandra Russo, and Katsumi Inoue. "Learning Revised Models for Planning in Adaptive Systems". In: *International Conference on Software Engineering*. IEEE Press, 2013. ISBN: 9781467330763. DOI: 10.1109/ICSE.2013.6606552 (pp. 31, 43).

[332] Vasilis Syrgkanis, Victor Lei, Miruna Oprescu, Maggie Hei, Keith Battocchi, and Greg Lewis. "Machine Learning Estimation of Heterogeneous Treatment Effects with Instruments". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Red Hook, NY, USA: Curran Associates Inc., 2019, pp. 15193–15202. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/3b2acfe2e38102074656ed938abf4ac3-Paper.pdf (pp. 12, 173).

[333] Idan Szpektor, Yoelle Maarek, and Dan Pelleg. “When Relevance is Not Enough: Promoting Diversity and Freshness in Personalized Question Recommendation”. In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW ’13. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, pp. 1249–1260. ISBN: 9781450320351. DOI: 10.1145/2488388.2488497 (pp. 142, 161, 234).

[334] Yukihiro Tagami, Toru Hotta, Yusuke Tanaka, Shingo Ono, Koji Tsukamoto, and Akira Tajima. “Filling Context-Ad Vocabulary Gaps with Click Logs”. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 1955–1964. ISBN: 9781450329569. DOI: 10.1145/2623330.2623334 (pp. 137, 144, 161, 170, 234).

[335] Giordano Tamburrelli and Alessandro Margara. “Towards Automated A/B Testing”. In: *Search-Based Software Engineering*. Ed. by Claire Le Goues and Shin Yoo. Cham: Springer International Publishing, 2014, pp. 184–198. ISBN: 978-3-319-09940-8 (pp. 11, 148, 152, 154, 161–164, 172, 235).

[336] Gabriel Tamura, Norha M. Villegas, Hausi A. Müller, João Pedro Sousa, Basil Becker, Gabor Karsai, Serge Mankovskii, Mauro Pezzè, Wilhelm Schäfer, Ladan Tahvildari, and Kenny Wong. “Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems”. In: *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Ed. by Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 108–132. ISBN: 978-3-642-35813-5. DOI: 10.1007/978-3-642-35813-5_5 (p. 5).

[337] Jimin Tan, Jianan Yang, Sai Wu, Gang Chen, and Jake Zhao. “A critical look at the current train/test split in machine learning”. In: *CoRR* abs/2106.04525 (2021). arXiv: 2106.04525. URL: https://arxiv.org/abs/2106.04525 (p. 121).

[338] Diane Tang, Ashish Agarwal, Deirdre O’Brien, and Mike Meyer. “Overlapping Experiment Infrastructure: More, Better, Faster Experimentation”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 17–26. ISBN: 9781450300551. DOI: 10.1145/1835804.1835810 (pp. 9, 148, 154, 161–164, 170, 236).

[339] Zhen Tang, Wei Wang, Lei Sun, Yu Huang, Heng Wu, Jun Wei, and Tao Huang. “IO dependent SSD cache allocation for elastic Hadoop applications”. In: *Science China Information Sciences* 61.5 (Apr. 2018), p. 050104. ISSN: 1869-1919. DOI: 10.1007/s11432-017-9401-y (p. 43).

[340] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. "On the use of hybrid reinforcement learning for autonomic resource allocation". In: *Cluster Computing* 10.3 (2007), pp. 287–299. DOI: 10.1007/s10586-007-0035-6 (pp. 31, 39, 43).

[341] Sebastian Thrun and Tom M. Mitchell. "Lifelong robot learning". In: *Robotics and Autonomous Systems* 15.1 (July 1995), pp. 25–46. ISSN: 0921-8890. DOI: 10.1016/0921-8890(95)00004-Y (p. 44).

[342] Viet Ha-Thuc, Avishek Dutta, Ren Mao, Matthew Wood, and Yunli Liu. "A Counterfactual Framework for Seller-Side A/B Testing on Marketplaces". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 2288–2296. ISBN: 9781450380164. DOI: 10.1145/3397271.3401434 (pp. 124, 137, 152, 161, 162, 170, 232).

[343] Mert Toslali, Srinivasan Parthasarathy, Fabio Oliveira, and Ayse K. Coskun. "JACKPOT: Online experimentation of cloud microservices". In: *Proceedings of the 12th USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'20. USA: USENIX Association, 2020. URL: https://www.usenix.org/system/files/hotcloud20_paper_toslali.pdf (pp. 159, 161, 163, 194, 235).

[344] Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. *Introducing MLOps: How to Scale Machine Learning in the Enterprise*. O'Reilly Media, Incorporated, 2020. ISBN: 9781492083290. URL: https://books.google.be/books?id=fR64zQEACAAJ (p. 172).

[345] Ye Tu, Kinjal Basu, Cyrus DiCiccio, Romil Bansal, Preetam Nandy, Padmini Jaikumar, and Shaunak Chatterjee. "Personalized Treatment Selection Using Causal Heterogeneity". In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 1574–1585. ISBN: 9781450383127. DOI: 10.1145/3442381.3450075 (pp. 161, 162, 165, 232).

[346] Yutaro Ueoka, Kota Tsubouchi, and Nobuyuki Shimizu. "Tackling Cannibalization Problems for Online Advertisement". In: *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*. UMAP '20. Genoa, Italy: Association for Computing Machinery, 2020, pp. 358–362. ISBN: 9781450368612. DOI: 10.1145/3340631.3394875 (pp. 142, 161, 233).

[347] Paulo Valente Klaine, Muhammad Imran, Oluwakayode Onireti, and Richard Souza. "A Survey of Machine Learning Techniques Applied to Self Organizing Cellular Networks". In: *IEEE Communications Surveys & Tutorials* 19.4 (July 2017), pp. 2392–2431. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2727878 (p. 28).

[348] Jeroen Van Der Donckt, Danny Weyns, M. Usman Iftikhar, and Sarpreet Singh Buttar. "Effective Decision Making in Self-adaptive Systems Using Cost-Benefit Analysis at Runtime and Online Learning of Adaptation Spaces". In: *Evaluation of Novel Approaches to Software Engineering*. Ed. by Ernesto Damiani, George Spanoudakis, and Leszek A. Maciaszek. Cham: Springer International Publishing, 2019, pp. 373–403. ISBN: 978-3-030-22559-9. DOI: 10.1007/978-3-030-22559-9_17 (p. 83).

[349] Jeroen Van Der Donckt, Danny Weyns, Federico Quin, Jonas Van Der Donckt, and Sam Michiels. "Applying Deep Learning to Reduce Large Adaptation Spaces of Self-Adaptive Systems with Multiple Types of Goals". In: *IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '20. Seoul, Republic of Korea, 2020, pp. 20–30. ISBN: 9781450379625. DOI: 10.1145/3387939.3391605 (pp. 53, 55, 91, 106, 289).

[350] Jean Vanderdonckt, Mathieu Zen, and Radu-Daniel Vatavu. "AB4Web: An On-Line A/B Tester for Comparing User Interface Design Alternatives". In: *Proc. ACM Hum.-Comput. Interact.* 3.EICS (June 2019). DOI: 10.1145/3331160 (pp. 125, 143, 161, 233).

[351] Deepak Kumar Vasthimal, Pavan Kumar Srirama, and Arun Kumar Akkinapalli. "Scalable Data Reporting Platform for A/B Tests". In: *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. 2019, pp. 230–238. DOI: 10.1109/BigDataSecurity-HPSC-IDS.2019.00052 (pp. 139, 146, 161, 163, 164, 234).

[352] Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. "A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems". In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 80–89. ISBN: 9781450305754. DOI: 10.1145/1988008.1988020 (p. 29).

[353] Thomas Vogel and Holger Giese. "Model-Driven Engineering of Self-Adaptive Software with EUREMA". In: *ACM Trans. Auton. Adapt. Syst.* 8.4 (Jan. 2014). ISSN: 1556-4665. DOI: 10.1145/2555612 (p. 2).

[354] Daniel Walper, Julia Kassau, Philipp Methfessel, Timo Pronold, and Wolfgang Einhauser. "Optimizing user interfaces in food production: gaze tracking is more sensitive for A-B-testing than behavioral data alone". In: *ACM Symposium on Eye Tracking Research and Applications*. ETRA '20 Short Papers. New York, NY, USA: Association for Computing Machinery, June 2020, pp. 1–4. ISBN: 978-1-4503-7134-6. DOI: 10.1145/3379156.3391351 (p. 127).

[355] Jiewen Wan, Qingshan Li, Lu Wang, Liu He, and Yvjie Li. "A self-adaptation framework for dealing with the complexities of software changes". In: *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. Nov. 2017, pp. 521–524. DOI: 10.1109/ICSESS.2017.8342969 (p. 44).

[356] Jian Wang and David Hardtke. "User Latent Preference Model for Better Downside Management in Recommender Systems". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 1209–1219. ISBN: 9781450334693. DOI: 10.1145/2736277.2741126 (pp. 161, 235).

[357] Weinan Wang and Xi Zhang. "CONQ: CONtinuous Quantile Treatment Effects for Large-Scale Online Controlled Experiments". In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 202–210. ISBN: 9781450382977. DOI: 10.1145/3437963.3441779 (pp. 161, 163, 232).

[358] Yu Wang, Somit Gupta, Jiannan Lu, Ali Mahmoudzadeh, and Sophia Liu. "On Heavy-user Bias in A/B Testing". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 2425–2428. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3358143 (pp. 124, 170).

[359] Zenan Wang, Carlos Carrion, Xiliang Lin, Fuhua Ji, Yongjun Bao, and Weipeng Yan. "Adaptive Experimentation with Delayed Binary Feedback". In: *Proceedings of the ACM Web Conference 2022*. WWW '22. Virtual Event, Lyon, France: Association for Computing Machinery, 2022, pp. 2247–2255. ISBN: 9781450390965. DOI: 10.1145/3485447.3512097 (pp. 140, 161, 233).

[360] B. L. Welch. "The Generalisation OF 'Student's' Problem when Several Different Population. Varlances are Involved". In: *Biometrika* 34.1-2 (1947), pp. 28–35. ISSN: 0006-3444. DOI: 10.1093/biomet/34.1-2.28. eprint: https://academic.oup.com/biomet/article-pdf/34/1-2/28/553093/34-1-2-28.pdf (p. 202).

[361] Danny Weyns. *Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective*. Wiley, 2020. ISBN: 9781119574941 (pp. 1–3, 27, 52, 59, 174, 210).

[362] Danny Weyns. "Software Engineering of Self-adaptive Systems". In: *Handbook of Software Engineering*. Cham: Springer International Publishing, 2019, pp. 399–443. ISBN: 978-3-030-00262-6. DOI: 10.1007/978-3-030-00262-6_11 (pp. 22, 47).

[363] Danny Weyns and Tanvir Ahmad. “Claims and Evidence for Architecture-Based Self-adaptation: A Systematic Literature Review”. In: *Software Architecture*. Ed. by Khalil Drira. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 249–265. ISBN: 978-3-642-39031-9. DOI: 10.1007/978-3-642-39031-9_22 (p. 29).

[364] Danny Weyns and Jesper Andersson. “From Self-Adaptation to Self-Evolution Leveraging the Operational Design Domain”. In: *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 90–96. DOI: 10.1109/SEAMS59076.2023.00022 (p. 6).

[365] Danny Weyns, Nelly Bencomo, Radu Calinescu, Javier Camara, Carlo Ghezzi, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jezequel, Sam Malek, Raffaela Mirandola, Marco Mori, and Giordano Tamburrelli. “Perpetual Assurances for Self-Adaptive Systems”. In: *Software Engineering for Self-Adaptive Systems III. Assurances*. Ed. by Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese. Cham: Springer International Publishing, 2017, pp. 31–63. ISBN: 978-3-319-74183-3 (p. 5).

[366] Danny Weyns, Nelly Bencomo, Radu Calinescu, Javier Cámara, Carlo Ghezzi, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jézéquel, Sam Malek, Raffaela Mirandola, Marco Mori, and Giordano Tamburrelli. “Perpetual Assurances for Self-Adaptive Systems”. In: *Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers*. Ed. by Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese. Vol. 9640. Lecture Notes in Computer Science. Springer, 2013, pp. 31–63. DOI: 10.1007/978-3-319-74183-3_2 (p. 11).

[367] Danny Weyns and Radu Calinescu. “Tele Assistance: A Self-Adaptive Service-Based System Exemplar”. In: *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2015. DOI: 10.1109/SEAMS.2015.27 (pp. 16, 86, 101, 199).

[368] Danny Weyns, Omid Gheibi, Federico Quin, and Jeroen Van Der Donckt. “Deep Learning for Effective and Efficient Reduction of Large Adaptation Spaces in Self-Adaptive Systems”. In: *ACM Trans. Auton. Adapt. Syst.* 17.1–2 (July 2022). ISSN: 1556-4665. DOI: 10.1145/3530192 (p. 289).

[369] Danny Weyns and M. Usman Iftikhar. “Model-Based Simulation at Runtime for Self-Adaptive Systems”. In: *IEEE International Conference on Autonomic Computing (ICAC)*. 2016, pp. 364–373. DOI: 10.1109/ICAC.2016.67 (pp. 24, 53).

[370] Danny Weyns, M. Usman Iftikhar, Sam Malek, and Jesper Andersson. "Claims and supporting evidence for self-adaptive systems: A literature study". In: *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2012, pp. 89–98. DOI: 10.1109/SEAMS.2012.6224395 (p. 52).

[371] Danny Weyns and Usman Iftikhar. "ActivFORMS: A Formally Founded Model-Based Approach to Engineer Self-Adaptive Systems". In: *ACM Transactions on Software Engineering and Methodology* (2022). (in print) (p. 53).

[372] Danny Weyns, Usman Iftikhar, Danny Hughes, and Nelson Matthys. "Applying Architecture-Based Adaptation to Automate the Management of Internet-of-Things". In: *Software Architecture*. Ed. by C. Cuesta, D. Garlan, and J. Pérez. Springer, 2018, pp. 49–67. ISBN: 978-3-030-00761-4. DOI: 10.1007/978-3-030-00761-4_4 (pp. 22, 47).

[373] Danny Weyns and Usman M. Iftikhar. "ActivFORMS: A Formally Founded Model-Based Approach to Engineer Self-Adaptive Systems". In: *ACM Trans. Softw. Eng. Methodol.* 32.1 (Feb. 2023). ISSN: 1049-331X. DOI: 10.1145/3522585 (p. 217).

[374] Danny Weyns, Sam Malek, and Jesper Andersson. "FORMS: A Formal Reference Model for Self-Adaptation". In: *Proceedings of the 7th International Conference on Autonomic Computing*. ICAC '10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 205–214. ISBN: 9781450300742. DOI: 10.1145/1809049.1809078 (pp. 4, 52).

[375] Danny Weyns, Sam Malek, and Jesper Andersson. "FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems". In: *ACM Trans. Auton. Adapt. Syst.* 7.1 (May 2012). ISSN: 1556-4665. DOI: 10.1145/2168260.2168268 (pp. 3, 22, 24).

[376] Danny Weyns, M. Usman Iftikhar, and Joakim Söderlund. "Do External Feedback Loops Improve the Design of Self-adaptive Systems? A Controlled Experiment". In: *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '13. San Francisco, CA, USA: IEEE Press, 2013, pp. 3–12. ISBN: 978-1-4673-4401-2. DOI: 10.1109/SEAMS.2013.6595487 (pp. 2, 174, 204).

[377] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H.C. Cheng, and Jean-Michel Bruel. "RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems". In: *2009 17th IEEE International Requirements Engineering Conference*. 2009, pp. 79–88. DOI: 10.1109/RE.2009.36 (p. 4).

[378] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Jan. 2014, pp. 1–332. ISBN: 978-3-662-43838-1. DOI: 10.1007/978-3-662-43839-8 (p. 14).

[379] Roel Wieringa. “Design Science Methodology: Principles and Practice”. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*. ICSE ’10. Cape Town, South Africa: Association for Computing Machinery, 2010, pp. 493–494. ISBN: 9781605587196. DOI: 10.1145/1810295.1810446 (p. 14).

[380] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012 (pp. 16, 98).

[381] Liang Wu and Mihajlo Grbovic. “How Airbnb Tells You Will Enjoy Sunset Sailing in Barcelona? Recommendation in a Two-Sided Travel Marketplace”. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 2387–2396. ISBN: 9781450380164. DOI: 10.1145/3397271.3401444 (pp. 5, 146, 161, 234).

[382] Yuhang Wu, Zeyu Zheng, Guangyu Zhang, Zuohua Zhang, and Chu Wang. “Non-Stationary A/B Tests”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD ’22. Washington DC, USA: Association for Computing Machinery, 2022, pp. 2079–2089. ISBN: 9781450393850. DOI: 10.1145/3534678.3539325 (pp. 161, 162, 231).

[383] Tong Xia, Sumit Bhardwaj, Pavel Dmitriev, and Aleksander Fabijan. “Safe Velocity: A Practical Guide to Software Deployment at Scale using Controlled Rollout”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2019, pp. 11–20. DOI: 10.1109/ICSE-SEIP.2019.00010 (pp. 152–154, 161, 163, 234).

[384] Bin Xiao. “Self-Evolvable Knowledge-Enhanced IoT Data Mobility for Smart Environment”. In: *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*. IML ’17. Liverpool, United Kingdom: Association for Computing Machinery, 2017. ISBN: 9781450352437. DOI: 10.1145/3109761.3109789 (p. 47).

[385] Yuxiang Xie, Nanyu Chen, and Xiaolin Shi. “False Discovery Rate Controlled Heterogeneous Treatment Effect Detection for Online Controlled Experiments”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 876–885. ISBN: 9781450355520. DOI: 10.1145/3219819.3219860 (pp. 138, 161, 162, 231).

[386] Yuxiang Xie, Meng Xu, Evan Chow, and Xiaolin Shi. “How to Measure Your App: A Couple of Pitfalls and Remedies in Measuring App Performance in Online Controlled Experiments”. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM ’21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 949–957. ISBN: 9781450382977. DOI: 10.1145/3437963.3441742 (pp. 137, 161, 232).

[387] Ya Xu and Nanyu Chen. "Evaluating Mobile Apps with A/B and Quasi A/B Tests". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 313–322. ISBN: 9781450342322. DOI: 10.1145/2939672.2939703 (pp. 124, 161, 162, 170, 232).

[388] Ya Xu, Nanyu Chen, Addrian Fernandez, Omar Sinno, and Anmol Bhasin. "From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 2227–2236. ISBN: 9781450336642. DOI: 10.1145/2783258.2788602 (pp. 161, 162, 164, 165, 171, 180, 232).

[389] Yanbo Xu, Divyat Mahajan, Liz Manrao, Amit Sharma, and Emre Kıcıman. "Split-Treatment Analysis to Rank Heterogeneous Causal Effects for Prospective Interventions". In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 409–417. ISBN: 9781450382977. DOI: 10.1145/3437963.3441821 (pp. 161, 233).

[390] Ye Xu, Zang Li, Abhishek Gupta, Ahmet Bugdayci, and Anmol Bhasin. "Modeling Professional Similarity by Mining Professional Career Trajectories". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 1945–1954. ISBN: 9781450329569. DOI: 10.1145/2623330.2623368 (pp. 137, 161, 234).

[391] Satoru Yamagata, Hiroyuki Nakagawa, Yuichi Sei, Yasuyuki Tahara, and Akihiko Ohsuga. "Self-Adaptation for Heterogeneous Client-Server Online Games". In: *International Conference on Intelligence Science*. Springer. 2019, pp. 65–79 (p. 44).

[392] Sezin Gizem Yaman, Myriam Munezero, Jürgen Münch, Fabian Fagerholm, Ossi Syd, Mika Aaltola, Christina Palmu, and Tomi Männistö. "Introducing continuous experimentation in large software-intensive product and service organisations". In: *Journal of Systems and Software* 133 (2017), pp. 195–211. ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.07.009 (p. 127).

[393] Wanshan Yang, Gemeng Yang, Ting Huang, Lijun Chen, and Youjian Eugene Liu. "Whales, Dolphins, or Minnows? Towards the Player Clustering in Free Online Games Based on Purchasing Behavior via Data Mining Technique". In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 4101–4108. DOI: 10.1109/BigData.2018.8622067 (pp. 137, 146, 161, 234).

[394] Runlong Ye, Pan Chen, Yini Mao, Angela Wang-Lin, Hammad Shaikh, Angela Zavaleta Bernuy, and Joseph Jay Williams. “Behavioral Consequences of Reminder Emails on Students’ Academic Performance: A Real-World Deployment”. In: *Proceedings of the 23rd Annual Conference on Information Technology Education*. SIGITE ’22. Chicago, IL, USA: Association for Computing Machinery, 2022, pp. 16–22. ISBN: 9781450393911. DOI: 10.1145/3537674.3554740 (pp. 139, 161, 233).

[395] Joseph L. Hellerstein an Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley and Sons, Inc., 2004. ISBN: 9780471668800. DOI: 10.1002/047166880X (p. 22).

[396] Takeshi Yoneda, Shunsuke Kozawa, Keisuke Osone, Yukinori Koide, Yosuke Abe, and Yoshifumi Seki. “Algorithms and System Architecture for Immediate Personalized News Recommendations”. In: *IEEE/WIC/ACM International Conference on Web Intelligence*. WI ’19. Thessaloniki, Greece: Association for Computing Machinery, 2019, pp. 124–131. ISBN: 9781450369343. DOI: 10.1145/3350546.3352509 (pp. 137, 139, 161, 232).

[397] Scott W. H. Young. “Improving Library User Experience with A/B Testing: Principles and Process”. In: *Weave: Journal of Library User Experience* 1 (Aug. 2014). DOI: 10.3998/weave.12535642.0001.101 (pp. 161, 236).

[398] Miao Yu, Wenbin Lu, and Rui Song. “A new framework for online testing of heterogeneous treatment effect”. In: 2020, pp. 10310–10317. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85106588123%5C&partnerID=40%5C&md5=53544f162212be7cd129e1f196debcd8 (pp. 144, 161, 235).

[399] Seid Žapčević and Peter Butala. “Adaptive process control based on a self-learning mechanism in autonomous manufacturing systems”. In: *The International Journal of Advanced Manufacturing Technology* 66.9 (June 2013), pp. 1725–1743. ISSN: 1433-3015. DOI: 10.1007/s00170-012-4453-0 (p. 37).

[400] He Zhang and Muhammad Ali Babar. “On Searching Relevant Studies in Software Engineering”. In: *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering*. EASE’10. UK: BCS Learning & Development Ltd., 2010, pp. 111–120 (p. 131).

[401] Tianqi Zhao, Wei Zhang, Haiyan Zhao, and Zhi Jin. “A Reinforcement Learning-Based Framework for the Generation and Evolution of Adaptation Rules”. In: *2017 IEEE International Conference on Autonomic Computing (ICAC)*. July 2017, pp. 103–112. DOI: 10.1109/ICAC.2017.47 (pp. 44, 173).

[402] Zhenyu Zhao and Totte Harinen. “Uplift Modeling for Multiple Treatments with Cost Optimization”. In: *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2019, pp. 422–431. DOI: 10.1109/DSAA.2019.00057 (p. 171).

[403] Zhenyu Zhao, Yan He, and Miao Chen. “Inform Product Change through Experimentation with Data-Driven Behavioral Segmentation”. In: *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2017, pp. 69–78. DOI: 10.1109/DSAA.2017.65 (pp. 137, 146, 159, 161, 171, 235).

[404] Xingquan Zuo, Guoxiang Zhang, and Wei Tan. “Self-Adaptive Learning PSO-Based Deadline Constrained Task Scheduling for Hybrid IaaS Cloud”. In: *IEEE Transactions on Automation Science and Engineering* 11.2 (2014), pp. 564–573. DOI: 10.1109/TASE.2013.2272758 (p. 39).

List of publications

Journal publications

- Omid Gheibi, Danny Weyns, and Federico Quin. “Applying Machine Learning in Self-Adaptive Systems: A Systematic Literature Review”. In: *ACM Trans. Auton. Adapt. Syst.* 15.3 (Aug. 2021). ISSN: 1556-4665. DOI: 10.1145/3469440
- Federico Quin, Danny Weyns, and Omid Gheibi. “Reducing large adaptation spaces in self-adaptive systems using classical machine learning”. In: *Journal of Systems and Software* 190 (2022), p. 111341. ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111341
- Danny Weyns, Omid Gheibi, Federico Quin, and Jeroen Van Der Donckt. “Deep Learning for Effective and Efficient Reduction of Large Adaptation Spaces in Self-Adaptive Systems”. In: *ACM Trans. Auton. Adapt. Syst.* 17.1–2 (July 2022). ISSN: 1556-4665. DOI: 10.1145/3530192

Conference publications

- Federico Quin, Danny Weyns, Thomas Bamelis, Sarpreet Singh Buttar, and Sam Michiels. “Efficient Analysis of Large Adaptation Spaces in Self-Adaptive Systems Using Machine Learning”. In: *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’19. Montreal, Quebec, Canada: IEEE Press, 2019, pp. 1–12. DOI: 10.1109/SEAMS.2019.00011
- Jeroen Van Der Donckt, Danny Weyns, Federico Quin, Jonas Van Der Donckt, and Sam Michiels. “Applying Deep Learning to Reduce Large Adaptation Spaces of Self-Adaptive Systems with Multiple Types of Goals”. In: *IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-*

Managing Systems. SEAMS '20. Seoul, Republic of Korea, 2020, pp. 20–30. ISBN: 9781450379625. DOI: 10.1145/3387939.3391605

- Federico Quin. "Systematic Approach to Engineer Decentralized Self-adaptive Systems". In: *Software Architecture*. Ed. by Henry Muccini et al. Cham: Springer International Publishing, 2020, pp. 38–50. ISBN: 978-3-030-59155-7
- Federico Quin, Danny Weyns, and Omid Gheibi. "Decentralized Self-Adaptive Systems: A Mapping Study". In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 2021, pp. 18–29. DOI: 10.1109/SEAMS51251.2021.00014
- Omid Gheibi, Danny Weyns, and Federico Quin. "On the Impact of Applying Machine Learning in the Decision-Making of Self-Adaptive Systems". In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 2021, pp. 104–110. DOI: 10.1109/SEAMS51251.2021.00023
- Federico Quin and Danny Weyns. "SEAByTE: A Self-Adaptive Micro-Service System Artifact for Automating A/B Testing". In: *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 77–83. ISBN: 9781450393058. DOI: 10.1145/3524844.3528081
- Maxim Reynvoet, Omid Gheibi, Federico Quin, and Danny Weyns. "Detecting and Mitigating Jamming Attacks in IoT Networks Using Self-Adaptation". In: *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. 2022, pp. 7–12. DOI: 10.1109/ACSOSC56246.2022.00019

Under review (submitted)

- Federico Quin and Danny Weyns. "Automating Pipelines of A/B Tests with Population Split Using Self-Adaptation and Machine Learning". In: *arXiv preprint arXiv:2306.01407*. 2023. DOI: 10.48550/arXiv.2306.01407
- Federico Quin, Danny Weyns, Matthias Galster, and Camila Costa Silva. "A/B Testing: A Systematic Literature Review". In: *arXiv preprint arXiv:2308.04929* (2023). DOI: 10.48550/arXiv.2308.04929

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
IMEC-DISTRINET
Celestijnenlaan 200A box 2402
B-3001 Leuven
federico.quin@kuleuven.be
https://distrinet.cs.kuleuven.be/