The present work was submitted to
the Research Group
Software Construction

of the Faculty of Mathematics,
Computer Science, and
Natural Sciences

Bachelor Thesis

An Overview of Machine Learning Approaches to support Software Architecting

presented by

Keven Hu

Aachen, August 8, 2024

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Alex Sabau, M.Sc.

Acknowledgment

I would like to thank my supervisor Alex Sabau for his guidance and feedback throughout my work on this thesis. Despite being very busy with many other tasks and responsibilities, he always managed to find time to guide me and offer valuable feedback on my work. I am also very grateful to the Department of Computer Science of RWTH Aachen University for teaching me the various aspects of computer science in interesting and exciting ways and helping me in the matters of my studies when I was unsure. Lastly, I would like to thank my family for always supporting me throughout my studies. I would not have been able to reach this point without their help and encouragement.

Keven Hu

Abstract

Software and software systems require well-designed software architectures to fulfil ever more complex tasks and to ensure ease of understanding of the system, maintainability, evolution and other aspects. Thus, tools or approaches that support effectively and efficiently software architects who directly work on software and software systems can be very useful. With machine learning being applied to a growing list of domains such as medicine, autonomous driving or cybersecurity, naturally an interest in its applications regarding software architecture exists. However, there is no current overview of the available research. In order to offer such an overview of machine learning approaches that support software architecting, a systematic literature review was conducted. The results of the systematic literature review show that a wide variety of machine learning approaches that support software architecting exist. A total of 25 studies which proposed a machine learning approach were found. The approaches can be classified into different fields with the field of *Evaluation*, which concentrates on evaluating software architecture artefacts, having the most studies with ten in total. In terms of challenges and future work directions, data gathering is the most mentioned challenge with nine studies and conducting an efficiency analysis being one of several potential research ideas.

Contents

List of Tables

List of Figures

1 Introduction

In a world where digitalisation continues to grow in importance with a global spending forecast of more than five trillion dollars in the information technology sector [Gar24], the number of software systems that require good software architecture also grows.

Software architecture describes the structure of a software system consisting of software components, their characteristics and their relationship to each other [Va09]. Indeed, the modular decomposition of the software or software system into software components plays an important role in the field of software architecture. It is necessary for fundamental principles of software engineering like information hiding or the separation of concerns. Information hiding describes the principle of making information, that a developer should not use for other components, inaccessible. The principle of separation of concerns ensures that a software component has one specific group of tasks which it is responsible for [LL23]. Thus, software architecture plays an important part in software engineering and its creation requires the software architect to consider aspects such as system design or requirements engineering. Even after designing and implementing the software architecture, the architecture is seldom final since the software or software system is often subject to changes, due to, for example, maintenance or evolution [Va09]. An activity which a software architect may have to complete is the modelling of the software architecture and its behaviour using a modelling language like MontiArcAutomaton [RRW15]. Thus, tools that support software architects in their software architecting efforts, like code generators in the case of modelling architecture and behaviour using MontiArcAutomaton [RRW15], are useful.

In recent years, artificial intelligence, which includes machine learning, has become prominent in real-world use cases. With large language model applications like ChatGPT reaching more than 100 million monthly active users [Hu23] machine learning has reached widespread use. Machine learning employs statistics and computer science to learn and extract knowledge from data. So-called machine learning models are trained and tested on datasets, so they can make predictions about the output for some given input [MG16]. Most machine learning can be differentiated into three types: supervised learning, unsupervised learning and reinforcement learning. In supervised learning, a machine learning model is trained on often manually pre-labelled training datasets. In contrast to supervised learning, in unsupervised learning a machine learning model is trained on unlabelled data. Reinforcement learning aims to find the optimal actions that maximise some form of reward or minimise some form of penalty [Bis06]. The machine learning models employ various algorithms to learn from the training dataset. Some common machine learning models are k-Nearest-Neighbour, naive Bayes, Decision Tree, Support Vector Machines or k-Means-Clustering [MG16].

Since machine learning has proven its practicality in real-world use cases, such as

with ChatGPT [Hu23], there is interest in the research of machine learning in software engineering. An example of a research topic regarding machine learning in software engineering is the automation and data management of the machine learning training process. It is explored in the paper submitted by Rumpe et al. by defining and testing a new artefact model [Rum+21]. Thus, other research topics may deal specifically with the sub-field of employing machine learning to support software architecting. An overview that shows the current state of machine learning approaches that support software architecting would be useful for considering further research into the topic. In order to create an overview of the topic, a systematic literature review was conducted which was used to answer the following three research questions:

- RQ1: What machine learning approaches exist for the support of software architecting?
- RQ2: How can existing approaches be classified in terms of their relationship to software architecting?
- RQ3: What are the current challenges and research gaps of machine learning approaches for supporting software architecting?

RQ1 is there to create a rough overview of what kind of machine learning approaches exist that support software architecting. The systematic literature review contains a table of relevant papers that fit the topic for this purpose. With RQ2 it is possible to take a closer look at the current trend of the research. It shows how the papers can be ordered into groups which subtopics in the research are explored the most and which topics may have potential for further research. Since this overview strives to also offer ideas or inspiration for future work to interested researchers, RQ3 is necessary. In order to offer inspiration for future work in the research field of machine learning approaches that support software architecting, R3 aims to find common challenges and apparent research gaps. It is possible to propose ideas or directions for future work based on these challenges and research gaps, as they show what research may be helpful or missing.

This thesis is structured in the following into chapters, beginning with the chapter 2 which deals with related work to the topic of this thesis. After the chapter about related work, the research method of a systematic literature review is explained and applied in chapter 3. The chapter 3 also includes tables and figures for visualisation purposes. Chapter 4 presents the results of the systematic literature review. These results are also compiled into diagrams, which are included in chapter 4. After the result presentation in chapter 4, the research questions are answered in chapter 5 by discussing the results from the previous chapter. Finally, in chapter 6, the thesis will be summarised in a concise manner and potential ideas for future work are presented.

2 Related Work

In terms of related work, the systematic literature review by Wang et al. investigates how the complexity of applying machine learning and deep learning solutions to software engineering problems leads to issues regarding the replicability of those studies. In addition, the review explores the differences and commonalities between machine learning and deep learning applications in the field of software engineering, such as the improvements of deep learning over machine learning or differences in training or implementation. Among the findings is that the rationale for choosing deep learning is achieving better performance or solving complex problems, while machine learning is chosen for better performance, robustness to diverse data, simplicity of the task, better interpretability of prediction results and simple implementation through available implementations [Wan+23].

Furthermore, the study by Barenkamp, Rebstadt, and Thomas conducts a systematic review and five interviews with software developers to determine the current development status, future development potentials and risks associated with the application of machine learning in software engineering. The main potentials of machine learning are the automation of routine work, the structured analysis of large data pools to discover novel information clusters and the structured evaluation of the data in neural networks, according to the study. The five interviews reveal the belief of the interviewed software developers that software developers will continue to have a defining role during software development since software innovation requires creativity, which machine learning is supposedly not capable of [BRT20].

The survey conducted by Yang et al. explores the application of deep learning in software engineering and tries to summarise deep learning applications, classify them, and determine challenges in the research field. The most common deep learning architectures that were applied are recurrent neural networks, concurrent neural networks and feedforward neural networks, according to the survey. Also, deep learning tackled problems in software engineering activities, software requirements, software design, software implementation, software testing and debugging, maintenance and software management. As for challenges, the lack of transparency for understanding the deep learning output, the lack of real-world datasets used for training and evaluation, reliance on large datasets and the lack of performance analysis are mentioned [Yan+20].

3 Research Method

This thesis follows the guidelines proposed by Kitchenham and Charters in "Guidelines for performing Systematic Literature Reviews in Software Engineering" [KC07]. Due to the time- and resource limitations of a bachelor thesis, the systematic literature review was not strictly conducted according to the guidelines. This applies especially to the activities that require more than one researcher, such that activities had to be adjusted or cut for this review. The systematic literature review process, which was followed for this thesis, can be divided into four steps: *Planning*, *Data Aggregation*, *Data Extraction* and *Data Synthesis* as seen in 3.1.

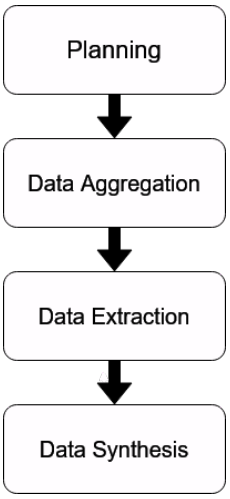

Figure 3.1: Process of the systematic literature review.

3.1 Planning

In the *Planning* step of the systematic literature review, several pre-review activities have to be addressed. First of all, the reviewer has to confirm the need for a review on the given topic. A systematic literature review's purpose is to summarise all information regarding a topic or phenomenon thoroughly in an unbiased manner or be a starting point for further research. Once the need for a systematic literature review has been confirmed by the reviewer, the research questions that the review aims to answer have to be defined. For this purpose, any of the six criteria *population*, *intervention*, *comparison*, *outcomes*, *context*, and *experimental designs* can be considered while defining the research questions.

Lastly, the reviewer has to develop a review protocol in order to reduce researcher bias and specify the methods used for conducting the review. The review protocol consists of all elements of the review in addition to some planning information [KC07].

The first task in the *Planning* step of the systematic literature review is to show the necessity of the review. As illustrated in 1, machine learning and its applications have been intensively researched in the past few years. Alongside the importance of high-quality software architecture and its activities relating to it, the interest in exploring machine learning approaches or tools that support the activities of software architects is reasonable. Thus, an overview of the current state of research and its challenges in the topic of machine learning approaches or tools to support software architecture activities would be a helpful starting point for researchers who are new to the field. However, as far as I have found from searches in the databases of IEEExplore and Engineering Village, there are currently no reviews, overviews or surveys that deal with the topic of machine learning approaches that support software architecting activities. Therefore, a new systematic literature review of the machine learning approaches that support software architecting is necessary to understand the current state of the research.

A research question is necessary for a systematic literature review since it determines the goal and direction that the review attempts to achieve. To define the research question, the criteria *population*, *intervention*, *comparison*, *outcomes*, *context* and *experimental design* can be considered. In this review, the criteria *population* and *intervention* were considered for the definition of the research question. Other criteria were ignored in order to avoid limiting search results relating to the topic of machine learning approaches that support software architecting. The criterion of *population* is supposed to address software engineering roles or -categories, application areas or industry groups, according to Kitchenham and Charters [KC07]. Here, the application area of software architecture and the role of software architects and their activities are chosen as the *population*. An *intervention* is a "software methodology/tool/technology/procedure that addresses a specific issue", as defined by Kitchenham and Charters [KC07]. In this review, the considered *intervention* are machine learning approaches – methodologies, tools or procedures – that support the activities of software architects. As a result, the research question according to the defined criteria coincides with the first research question of this thesis: "What machine learning approaches exist for the support of software architecting?". The *population* is represented by the activities that fall under "software architecting" and the *intervention* is mentioned as "machine learning approaches".

The review protocol consists of the chapters Research Method 3, Results 4 and Discussion 5 in this thesis. A timeline of the review process can be seen in figure 3.2.

3.2 Data Aggregation

In order to gather as many relevant papers as possible, the reviewer has to identify suitable data sources for the search first. After the data source selection, the reviewer has to craft a *search string* which is used on the data sources. This *search string* can be based on the structured research question and the considered criteria, which have been

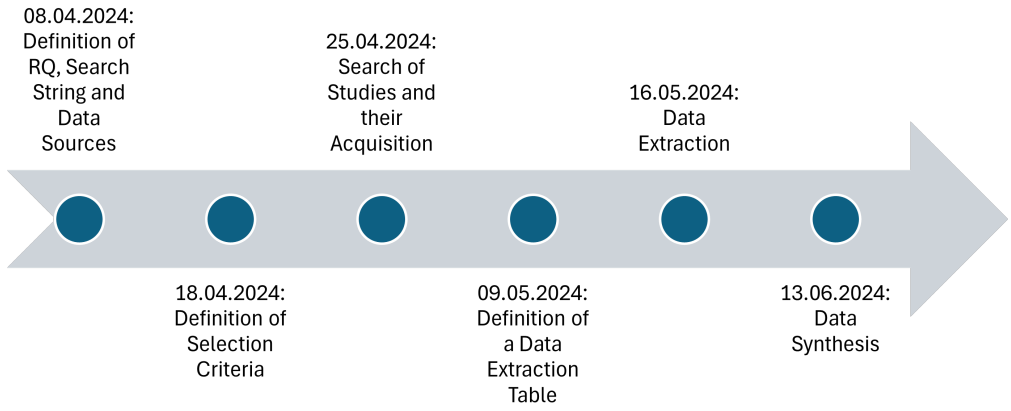

Figure 3.2: Timeline of the review process.

defined in the earlier step. Following the creation of the *search string*, *selection criteria* have to be determined for the search in the data sources. The purpose of the *selection criteria* is to identify the studies that provide evidence to answer the research question. The search itself has also to be sufficiently documented for replication purposes [KC07].

For this review, IEEExplore and Engineering Village have been chosen as data sources for studies. IEEExplore is a digital platform that enables access to content published by IEEE with more than six million documents [IEE] and its focus on computer science makes it a suitable source of studies for a systematic literature review. Engineering Village is also a digital platform that concentrates on providing access to many scientific papers regarding engineering by drawing from the content of several databases, such as Compendex [Vil]. The Compendex database by itself contains more than ten million papers [Els], thus enabling a wide search of studies regarding the review topic.

In order to craft a *search string* for the data sources, I have incorporated the *population* and *intervention*, which were determined in the *Planning* step earlier. The search string is

```
("machine learning" OR ml OR ai OR "artificial intelligence")
AND (approach OR application OR tool) AND ("software architect*")
```

.

This *search string* can be split into three parts, based on the parentheses. The first and second parts list several terms that describe the *intervention* that is examined in the review. The third part represents the *population* of the review.

Applying the search string may yield studies that do not fit the topic of machine learning approaches that support software architecting. Therefore, selection criteria are helpful since they can be used to filter out unrelated papers. In the next task, I have determined which *selection criteria* to use for filtering the search results. The criteria were differentiated into *inclusion-* and *exclusion criteria*.

Inclusion criteria:

- The paper has to be about a field in software architecture.
- The paper has to be about a machine learning approach regarding the field.

Exclusion criteria:

- The paper is a survey, overview or review.
- The paper has no English or German version.
- The paper is not about machine learning approaches in the field of software architecture.
- The paper is only a proposal without any implementation or testing.

The *inclusion criteria* ensure that the papers are about the review topic. Naturally, papers that are not about the research topic are excluded. Meanwhile, surveys, overviews or reviews have been excluded as well, since their nature as a work that also summarises other studies requires the analysis of those other studies as well, which is not feasible with the time and resource limitations of this thesis. The exclusion of papers without an English or German version stems from the limitations of my personal language skills. Papers that only offer a proposal for a tool, approach or methodology without any implementation or testing are excluded since they offer no results that can be considered in the systematic literature review.

A search was conducted with the *search string* and a publication and indexing time range from 2019 to 06.05.2024. 2019 has been used as a cut-off year in order to offer an overview of the current state of the research. On IEEExplore the Advanced Search function was used. The *search string* was split up according to the parentheses into three sub-search terms; the parentheses were removed and the subterms were linked with the logical AND operator. Each of the subterms was searched in the index terms of the database. Once the results of the search were displayed, a publication topic filter was also used. The publication topics that should be included in the results were ”Software Architecture” and ”Machine Learning”. This superficial search offered 162 results on IEEExplore. On Engineering Village the Quick Search function was used for the search. Like on IEEExplore, the *search string* was split up according to the parentheses, the parentheses removed and the subterms linked with the logical AND operator. Also, the same time range as for IEEExplore was used. In addition, the search results were filtered by using the Controlled Vocabulary filter offered by Engineering Village. The papers had to include the vocabulary ”Software Architecture” and ”Machine Learning” on Engineering Village which led to 195 search results. It should be mentioned, however, that since the initial searches, additional papers may have been indexed, which can make a replication of the searches more difficult.

Since a close examination of more than 300 studies was not feasible for this thesis, a superficial screening of the title and, if unclear, the abstract was done based on the

review topic. One further study was also recommended by a supervisor, and that was not covered by the database search, fit the review topic, and thus was included.

37 papers are from the search on the databases and one was recommended by the supervisor of this thesis. Therefore, a total of 38 papers remained as search results. For the study acquisition, the reference management and knowledge organisation program Citavi 6 [Cit] and its browser extension was used to download and automatically fill out the papers and their reference information using, for example, the digital object identifier of the study.

Finally, the *selection criteria* were applied to all papers, including the one recommended by the supervisor, to determine the final body of papers for the systematic literature review. Thus, the final body of research that passed the selection consisted of 25 papers.

3.3 Data Extraction

Once the search is done and the reviewer has applied all selection criteria to the search results, a data extraction form can be used to gather the information from the selected studies. The information that is gathered with the form should facilitate finding answers to the research questions. In addition, the form should include data like the name of the reviewer, date of the data extraction, title, author and other publication information of the studies [KC07]. The data extraction form was an Excel file, which contained a table with the relevant data.

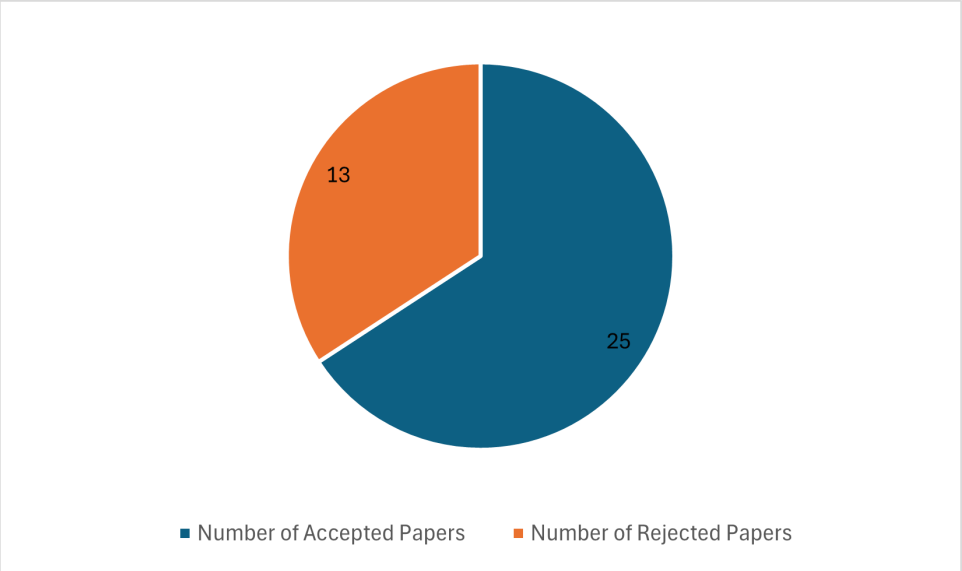

Figure 3.3: Ratio of studies after initial screening which passed the study selection.

This data includes the following information:

- ID, which allows an easy and unique identification and reference of the papers
- Title of the paper
- Authors of the paper
- Result of the study selection
- Data source of the paper
- In case of failure, the reason why a paper failed the study selection
- Software architecting activity tackled in the paper
- Maturity of the tool or approach
- Common potential future work directions
- Common perceived challenges
- Existence of efficiency tests or experiments

The extracted data is included as tables in this thesis. In the following, I will present the information from the data extraction table. The table is split into multiple ones due to the limitations of the page dimensions. The first two tables, 3.1 and 3.2 are to define an index of the examined papers. They also offer an answer to RQ1. With the index, any paper can be easily referred to with its respective ID. The authors of the examined papers are listed in the tables 3.3 and 3.4.

After the selection criteria were applied to the studies, the results of the selection were documented alongside a short explanation of why a paper did not pass the selection. The exact results can be seen in the tables 3.5 and 3.6. Studies that passed the study selection are marked with the keyword "TRUE" in the second column of the tables.

As mentioned in the *Planning* chapter, the studies must include any testing or implementation so that the results can be considered in this review. Thus, any paper that is only a proposal without any testing or implementation is not examined in this review. Any studies that do not deal with software architecture or machine learning approaches directly are excluded since they do not fit the topic of this review. Furthermore, a study has to be written understandably since ambiguity may lead to misunderstandings. A diagram of the result is presented in 3.3 and shows that from the 38 papers that passed the initial screening, 25 passed the study selection based on the *selection criteria* defined in 3. Among the papers that failed the study selection, five failed since they were only proposals without any implementation or testing, seven failed since they were not about the topic of this review and one failed since the language was too ambiguous. The paper in question with the ID 35 explains software complexity as the required level of knowledge for understanding the design of a software program or component. Furthermore, the paper states that its contribution is the proposal of a system that can

apply machine learning to predict the software complexity level [Aka+22]. This implies that the proposed system can make some sort of measure that describes the software complexity level of a given software program or component. However, study 35 trains a machine learning model on user data, which is then used to predict the user's software complexity level [Aka+22] which may be interpreted as the experience and knowledge of the user. Therefore, it is unclear whether the software complexity level of software or the experience and knowledge of the user are predicted by the machine learning model.

In addition, the papers were classified for later data synthesis. The classification itself will be defined in the relevant chapter 4.

Furthermore, I attempted to determine the maturity of the proposed tools and approaches. The term *practical* means that the paper shows a successful application of the proposed tool or approach and that the tool or approach may also apply to other examples, including real-world examples. If a tool or approach is not likely to work easily with other examples, which may be due to extensive, required adjustments to the example or the approach, the respective paper is labelled as *experimental*. Since the understanding of the terms *practical* and *experimental* is still subjective, despite the definition provided earlier, the categorisation of the machine learning approaches has also a subjective nature. However, various aspects were considered that steered the categorization. It was considered what kind of format the initial input requires. If the required format is common, like when requiring a model to adhere to the rules of the Unified Modelling Language, then this format requirement is more practical than requiring an input to be in a for the study newly defined format. Furthermore, it was considered whether the proposed machine learning tool or approach was tested on a real-world example. Lastly, the implementation of the tool or definition of the approach was considered in terms of their completeness.

In order to determine current challenges and future directions of machine learning approaches that support software architecting, I have summarised the directions and challenges of the papers in table 3.7. These future directions and challenges are generalised since especially the proposed future work of the papers refers to activities specific to the application instance of the machine learning tool or approach. These specific future work activities were cut if they did not apply to any other paper.

During the reading of the papers, two challenges appeared to be relevant in multiple papers. Firstly, researchers had difficulties when they tried to find high-quality and quantity data for training and testing their machine learning models. Secondly, due to limitations in time and resources, several authors decided to limit the application domain of their machine learning approach. The application domain means, in this case, aspects like the operating system or programming language. The exact data is available in table 3.7.

3.4 Data Synthesis

In the last step of the systematic literature review, the reviewer has to perform data synthesis by summarising and collating the information from the selected studies. The

synthesis itself can be descriptive, quantitative or qualitative in nature. As part of a *descriptive synthesis*, the extracted information of the studies should be tabulated and the differences highlighted. In the *quantitative synthesis*, data about sample size, estimates about effect size, differences between mean values for the interventions and more are tabulated and then compiled into a forest plot. During a *qualitative synthesis* of data, the reviewer attempts to integrate studies that use natural language results and conclusions and translate the cases into each other[KC07].

The examined papers differ in their application field and the way machine learning is applied. A *quantitative synthesis* requires the results of the studies to be comparable to each other, which is not possible with the papers examined in this review. A translation of researched cases from one study to the other is also not possible since the application fields can be entirely different from each other. Due to this heterogeneous nature of all papers, a *quantitative-* or *qualitative synthesis* were not an option, and a *descriptive synthesis* was chosen instead. For the *data synthesis*, diagrams have been created in the Excel file, which will be included in this thesis.

ID	Paper Title
1	A comparison of machine learning-based text classifiers for mapping source code to architectural modules
2	A Hybrid Approach to MVC Architectural Layers Analysis
3	A Proposed Model-Driven Approach to Manage Architectural Technical Debt Life Cycle
4	A Sketch of a Deep Learning Approach for Discovering UML Class Diagrams from System's Textual Specification
5	Architecture-based Failure Prediction via LSTM and Bayesian Network in Service-oriented Systems
6	ArchTacRV: Detecting and Runtime Verifying Architectural Tactics in Code
7	Artificial Intelligence-Based Centralized Resource Management Application for Distributed Systems
8	Automated Planning for Software Architectural Migration
9	Automatic Examining of Software Architectures on Mobile Applications Codebases
10	Automatic Quality Attribute Scenarios Identification and Generation from Quality Attribute Requirements
11	ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction
12	Helping Software Architects Familiarize with the General Data Protection Regulation
13	Continuous and Proactive Software Architecture Evaluation: An IoT Case
14	Data-driven Adaptation in Microservice-based IoT Architectures
15	Deep Attentive Anomaly Detection for Microservice Systems with Multimodal Time-Series Data
16	Detecting architectural integrity violation patterns using machine learning
17	Detecting Model View Controller Architectural Layers using Clustering in Mobile Codebases
18	From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures
19	Enhanced Service Point Approach for Microservices Based Applications Using Machine Learning Techniques
20	Code Vectorization and Sequence of Accesses Strategies for Monolith Microservices Identification
21	A Decision Support System for Pattern-Driven Software Architecture

Table 3.1: First table of the index for the examined papers.

ID	Paper Title
22	Flexible Architecture for Data-Driven Predictive Maintenance with Support for Offline and Online Machine Learning Techniques
23	From Prose to Prototype: Synthesising Executable UML Models from Natural Language
24	Classifying Model-View-Controller Software Applications Using Self-Organizing Maps
25	Using Automatically Recommended Seed Mappings for Machine Learning-Based Code-to-Architecture Mappers
26	Identification of Architecturally Significant Non-Functional Requirement
27	InMap: Automated Interactive Code-to-Architecture Mapping Recommendations
28	Tackling Software Architecture Erosion: Joint Architecture and Implementation Repairing by a Knowledge-based Approach
29	Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study
30	Evaluation of Move Method Refactorings Recommendation Algorithms: Are We Doing It Right?
31	Preprocessing Requirements Documents for Automatic UML Modelling
32	Quantitative Verification-Aided Machine Learning: A Tandem Approach for Architecting Self-Adaptive IoT Systems
33	Deriving Architectural Responsibilities from Textual Requirements
34	Run-time evaluation of architectures: A case study of diversification in IoT
35	Software Complexity Automation Tool for Industrial Practices with Qualitative and Quantitative Aspects
36	Towards Automatic Classification of Design Decisions from Developer Conversations
37	Where to Handle an Exception? Recommending Exception Handling Locations from a Global Perspective
38	Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical study

Table 3.2: Second table of the index for the examined papers.

ID	Authors
1	Florean, Alexander; Jalal, Laoa; Sinkala, Zipani Tom; Herold, Sebastian
2	Dobrean, Dragoş; Dioşan, Laura
3	Perez, Boris; Correal, Dario; Astudillo, Hernan
4	Rigou, Yves; Lamontagne, Dany; Khriss, Ismail
5	Wu, Xin
6	Ge, Ning; Wang, Ze; Zhang, Li; Zhao, Jiuang; Zhou, Yufei; Liu, Zewei
7	Hettiarachchi, Lasal Sandeepa; Jayadeva, Senura Vihan; Bandara, Rusiru Abhisheak Vikum; Palliyaguruge, Dilmi; Arachchillage, Udara Srimath S. Samaratunge; Kasthurirathna, Dharshana
8	Chondamrongkul, Nacha; Sun, Jing; Warren, Ian
9	Dobrean, Dragos
10	Tessema, Amsalu; Alemneh, Esubalew
11	Cámara, Javier; Wohlrab, Rebekka; Garlan, David; Schmerl, Bradley
12	Colesky, Michael; Demetzou, Katerina; Fritsch, Lothar; Herold, Sebastian
13	Sobhy, Dalia; Minku, Leandro; Bahsoon, Rami; Kazman, Rick
14	Sanctis, Martina de; Muccini, Henry; Vaidhyanathan, Karthik
15	Chen, Yufu; Yan, Meng; Yang, Dan; Zhang, Xiaohong; Wang, Ziliang
16	Zakurdaeva, Alla; Weiss, Michael; Muegge, Steven
17	Dobrean, Dragos; Diosan, Laura
18	Eisenreich, Tobias; Speth, Sandro; Wagner, Stefan
19	Raj, Vinay; Ravichandra, Sadam
20	Faria, Vasco; Silva, António Rito
21	Farshidi, Siamak; Jansen, Slinger
22	Canito, Alda; Fernandes, Marta; Mourinho, Joao; Tosun, Serkan; Kaya, Kamer; Turupcu, Aysegul; Lagares, Angel; Karabulut, Huseyin; Marreiros, Goreti
23	Ramackers, Guus J.; Griffioen, Pepijn P.; Schouten, Martijn B.J.; Chaudron, Michel R.V.
24	Guaman, Daniel; Delgado, Soledad; Perez, Jennifer
25	Herold, Sebastian; Sinkala, Zipani
26	Mohammed, Esmael; Alemneh, Esubalew
27	Sinkala, Zipani Tom; Herold, Sebastian
28	Knieke, Christoph; Rausch, Andreas; Schindler, Mirco
29	Haque, Md. Ariful; Abdur Rahman, Md.; Siddik, Md Saeed
30	Novozhilov, Evgenii; Veselov, Ivan; Pravilov, Mikhail; Bryksin, Timofey

Table 3.3: First table with the authors of the examined papers.

ID	Authors
31	Schouten, Martijn B. J.; Ramackers, Guus J.; Verberne, Suzan
32	Camara, Javier; Muccini, Henry; Vaidhyanathan, Karthik
33	Rodriguez, Guillermo; Díaz-Pace, J. Andrés; Berdun, Luis; Misra, Sanjay
34	Sobhy, Dalia; Minku, Leandro; Bahsoon, Rami; Chen, Tao; Kazman, Rick
35	Akalanka, M.H.N; Weerasinghe, W.A.H.T.; Perera, H.K.P.S.; Kumari, T. N.; Wijendra, D. R.; Krishara, J.
36	Josephs, Alyssa; Gilson, Fabian; Galster, Matthias
37	Jia, Xiangyang; Chen, Songqiang; Zhou, Xingqi; Li, Xintong; Yu, Run; Chen, Xu; Xuan, Jifeng
38	Dhar, Rudra; Vaidhyanathan, Karthik; Varma, Vasudeva

Table 3.4: Second table with the authors of the examined papers.

ID	Passed?	Reason for failing
1	TRUE	
2	TRUE	
3	TRUE	
4	FALSE	only a proposal without any prototype implementation or preliminary testing
5	TRUE	
6	TRUE	
7	FALSE	only indirectly about software architecture (it is primarily about deployment optimizations)
8	TRUE	
9	FALSE	only a proposal without any prototype implementation or preliminary testing
10	TRUE	
11	TRUE	
12	FALSE	no application of machine learning
13	TRUE	
14	FALSE	only proposal without any prototype implementation or preliminary testing
15	TRUE	
16	TRUE	
17	TRUE	

Table 3.5: First table of selection results and reasons for failing.

ID	Passed?	Reason for failing
18	FALSE	only proposal without any prototype implementation or preliminary testing
19	TRUE	
20	TRUE	
21	FALSE	no application of machine learning
22	FALSE	no application of machine learning
23	TRUE	
24	TRUE	
25	TRUE	
26	TRUE	
27	FALSE	no application of machine learning
28	FALSE	only proposal without any prototype implementation or preliminary testing
29	TRUE	
30	FALSE	no application of machine learning
31	TRUE	
32	TRUE	
33	TRUE	
34	TRUE	
35	FALSE	very difficult to understand due to ambiguous language (it is unclear whether a machine learning model is used to predict the complexity of software or the user's experience regarding the complexity of software
36	TRUE	
37	FALSE	not about software architecture
38	TRUE	

Table 3.6: Second table of selection results and reasons for failing.

ID	Future Work	Challenges
1	test more models	
2	expand test cases	
3	expand use cases, recommendation system	data gathering
4		
5	expand test cases, test more models	data gathering
6	expand use cases	
7		
8	expand test cases	
9		
10	increase dataset, test more models	data gathering
11	expand use cases	
12		
13	expand use cases	
14		
15		
16	expand use cases, test more models	
17		
18		
19		data gathering
20		l. domain
21		
22		
23	expand use cases, recommendation system	
24	increase dataset, recommendation system	l. domain
25	recommendation system	data gathering, l. domain
26	increase dataset	data gathering
27		
28		
29	test more models	
30		
31	increase dataset	data gathering
32	expand use cases	
33		
34	test more models	
35		
36	increase dataset	data gathering
37		
38		data gathering

Table 3.7: Common directions for future work or perceived challenges in the papers.

ID	Software Architecting Activity	Maturity of Tool/Approach
1	Evaluation	experimental
2	Evaluation	experimental
3	Evaluation	practical
5	Quality Assurance	experimental
6	Maintenance	practical
8	Planning	practical
10	Requirements Engineering	experimental
11	Quality Assurance	practical
13	Evaluation	experimental
15	Maintenance	experimental
16	Evaluation	experimental
17	Evaluation	experimental
19	Planning	experimental
20	System Design	experimental
23	System Design	experimental
24	Evaluation	experimental
25	Evaluation	experimental
26	Requirements Engineering	experimental
29	Requirements Engineering	experimental
31	Requirements Engineering	experimental
32	Evaluation	experimental
33	Requirements Engineering	experimental
34	Evaluation	experimental
36	System Design	practical
38	System Design	experimental

Table 3.8: Classification of papers and the maturity of their tool or approach.

4 Results

For *data synthesis*, four diagrams have been created to further highlight the results of the synthesis.

In order to offer a more structured overview of the studies, which exist in the field of machine learning approaches that support software architecting, I defined a classification for the studies. The classification includes the classes *Planning*, *Requirements Engineering*, *System Design*, *Quality Assurance*, *Maintenance* and *Evaluation*. While *Requirements Engineering*, *System Design*, *Quality Assurance* and *Maintenance* are the same as their software architecture sub-field counterparts, *Planning* and *Evaluation* require a more detailed definition. *Planning* includes the activities that define future activities or assess aspects of the software project. Future activities can be an evolution path that software architects have to follow or, an estimation of the required effort for a software project can be an assessment. The activity field *Evaluation* involves the analysis and assessment of existing software architecture artefacts such as documentation, diagrams, implementation or deployed instances of the software architecture. Possible examples may be the evaluation of operation data in real time or the analysis of implementations regarding the employed architectural patterns. It should be noted that membership in one class does not always exclude an overlap to different classes, since, for example, a study classified as *Evaluation* may also have overlap to another class like *Quality Assurance*. Applying the classification to all 25 studies shows that ten are classified as papers about *Evaluation*, five about *Requirements Engineering*, four about *System Desgin* and two each about *Planning*, *Quality Assurance* and *Maintenance*. This result is visualised in the diagram 4.1.

Another aspect that has been considered in this review is the practicality of the approaches or tools that the studies have proposed. For this purpose, the studies are classified according to their maturity in *practical* and *experimental* as defined in section 3.3. As seen in the diagram 4.2, five of the studies proposed a tool or approach that can be considered *practical* while 20 are *experimental*.

Figure 4.3 is a visualisation of the results regarding proposed future work. It should be mentioned that a study can propose multiple future directions and that propositions that were unique to the approach or application field were ignored. Thus, the total of all common, proposed future directions is 24, which is not equal to the number of papers that passed the study selection. A future direction of seven studies is expanding use cases for their approach. Expanding use cases means that the tool or approach is adjusted, for example, through adding functionality or allowing a greater scope of formats an input can have. Another one is testing more machine learning models, which six papers have proposed. Increasing the dataset for training the machine learning model or creating some form of recommendation system are each proposed by four studies. Lastly, three

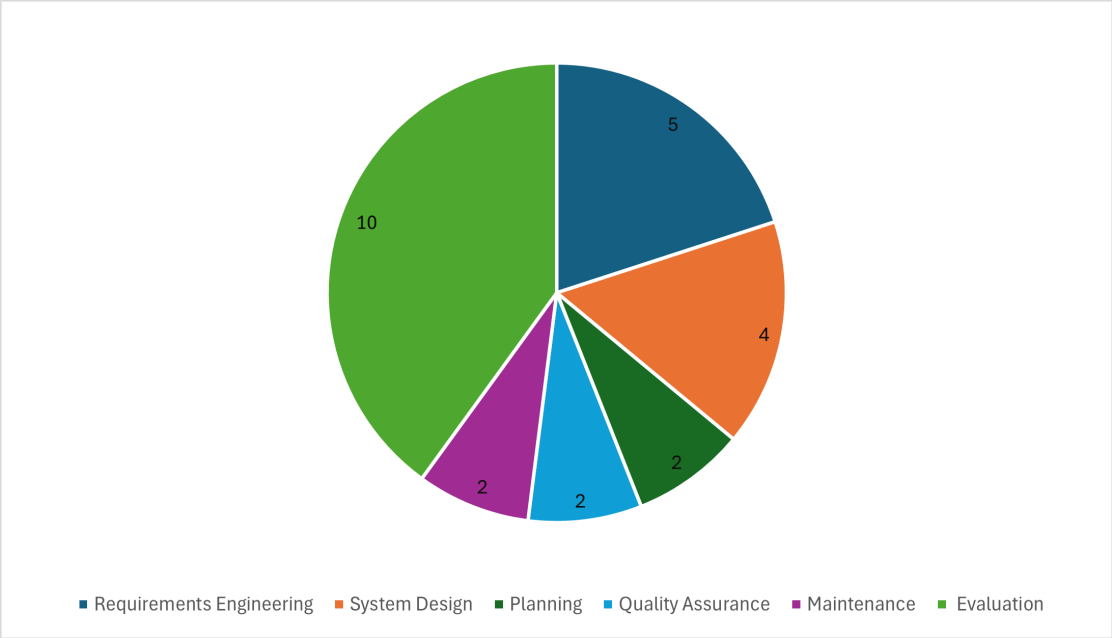

Figure 4.1: Number of papers in each classification class.

papers offer a future direction for the expansion of test cases.

Lastly, the challenges that the studies faced were the limited domain in which their tool or approach could be applied and the difficulties in gathering data for training and testing machine learning models. The limitation of the domain can mean, for example, that a machine learning approach only accepts source code written in a specific programming language since the machine learning model is only trained on that specific language. Data gathering is expressed by nine studies to be a challenge and the limitations of the domain by three. If a challenge of a study was data gathering, it means that finding and aggregating data for training and testing of machine learning was difficult since such data was either not available or difficult to find.

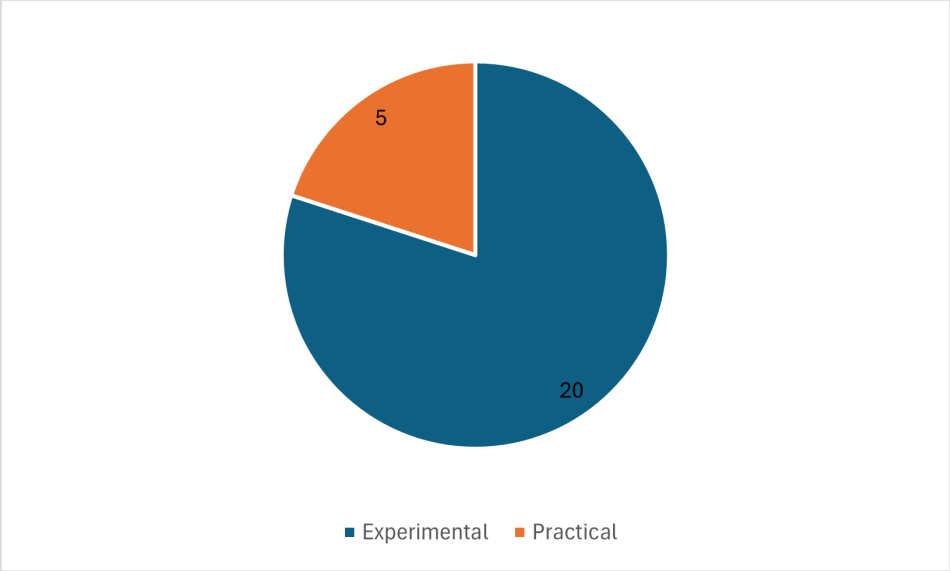

Figure 4.2: Ratio of the practical and experimental approaches.

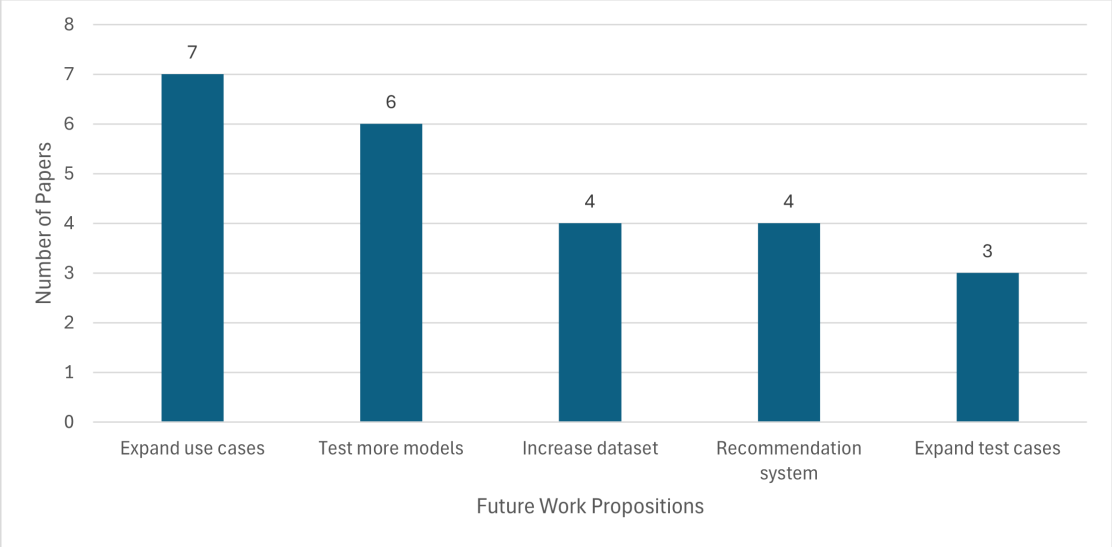

Figure 4.3: Common future directions, proposed by the studies.

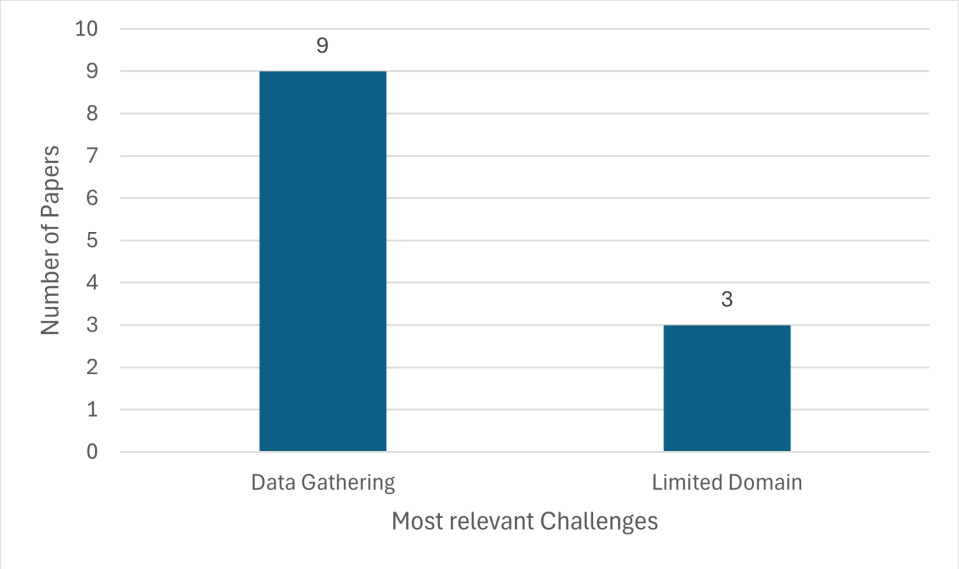

Figure 4.4: Common challenges in the papers.

5 Discussion

5.1 Research Question 1

While conducting the systematic literature review, a list of studies that proposed a machine learning tool or approach to support software architecting was created. The list is presented in the form of the tables 3.5 and 3.6 that show which studies are considered relevant for the review topic. This offers an unorganised overview of what kind of machine learning approaches exist. It is possible to conclude that the machine learning approaches are diverse in their application topic. This becomes apparent when the topics of several studies are compared against each other. Looking at the study by Florean et al. for example, which explores the potential of using machine learning-based text classifiers to map source code to architectural modules. Most approaches that combat software architecture degradation require such a source code-to-architectural modules mapping. Therefore, these approaches would benefit since automation through machine learning would replace a challenging and labour-intensive manual mapping [Flo+21]. Another study that is written by Wu proposes an approach that uses a long short-term memory network and a Bayesian network to predict time series data of software components and use the data to predict the failure of the software system. Other approaches are not well suited to make predictions based on irregular real-world data, which is why using a long short-term memory network in combination with a Bayesian network improves the accuracy of software system failures [Wu22]. The topics of those two studies are the mapping of source code to architectural modules to fight software degradation and the application of a long short-term memory network and a Bayesian network for software system failure prediction, which are very different from each other and show the diversity of the topics. Since the machine learning approaches cover a wide scope of topics, a classification would be useful for organising the studies and offering a more structured overview of the current state of research.

5.2 Research Question 2

Due to the wide scope of topics that are tackled by the machine learning tools and approaches of the examined studies, it is challenging to define a classification that not only has a level of abstraction that covers all application topics but also is still specific enough to examine the relationship of the approaches to software architecting. The classification of the studies into the classes *Requirements Engineering*, *System Design*, *Quality Assurance*, *Maintenance*, *Planning* and *Evaluation* is abstract enough to cover all topics of the examined studies, while still allowing to explore the relationship of the

machine learning tools or approaches to software architecting.

In figure 4.1, the distribution of the specific classes is visible. With 40%, most studies deal with the *Evaluation* of software architecture artefacts such as implementations or documentation. This shows a big trend in using machine learning towards the analysis and assessment of those artefacts. In addition to the aspect of automation, machine learning's property to identify abstract relationships seems to also play a part. An example where machine learning is used to evaluate software architecture artefacts is paper 1 which has already been mentioned in section 5.1 and deals with the mapping of source code to architectural modules to combat software architecture degradation [Flo+21].

Requirements Engineering is the second largest class, with 20% of studies belonging to the class. In this class, the studies primarily focus on extracting data regarding requirements from documentation or developer conversations. For this purpose, machine learning is used to identify and classify elements. Study 26 "Identification of Architecturally Significant Non-Functional Requirement" examines the performance of a support vector machine, a naive Bayes model and a k-nearest-Neighbour model in identifying architecturally significant non-functional requirements, for example, [MA21].

The class of studies regarding *System Design* takes up 16% of passed papers or four in total. A noteworthy mention in this class is study 38 "Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical study". The study compares the performance of several large language models like GPT4 or T5 regarding their architectural design decision (ADD) generation capabilities and comes to the conclusion that large language models can generate ADDs and format them in Markdown in an assistive manner [DVV24]. It is noteworthy since in the instance of the study it has been shown that machine learning can generate non-trivial software architecture artefacts, like ADDs in Markdown code.

Lastly, the classes *Planning*, *Quality Assurance* and *Maintenance* have two papers or 8% of all papers each. One example for a study in the activity field of *Planning* is written by Raj and Ravichandra with ID 19. It proposes the application of the service point approach, which is used for effort estimation, with a regression model to estimate the required effort for the migration of an architecture to microservice architecture. This approach aims to improve and simplify effort estimation of an architectural migration since good effort estimation is essential for the planning and management of time and resources [RR22].

An example where machine learning is used for *Quality Assurance* is the paper "ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction" with ID 11. The study proposes an approach that uses principal component analysis to identify the most relevant design decisions that impact the quality of service and decision tree learning to identify how design decisions impact system qualities. This makes it possible to link design decisions to the satisfaction of requirements like quality of service and understand the trade-off of design decisions. This was not possible before with other approaches that help in the search for good architecture designs [Cam+23].

As a part of the class *Maintenance*, study 6 by Ge et al. proposes a tool that detects the

methods responsible for the interaction behaviour of an implemented architectural tactic and checks its consistency with the original architectural tactic design. The detection is done by using five machine learning models, which are compared for performance, and the consistency checking is done by a runtime verification method [Ge+22].

Based on the distribution of the papers in their activity fields, it becomes apparent that the fields of *Planning*, *Quality Assurance* and *Maintenance* are researched the least. Therefore, it is possible to conclude that there is a lack of studies in these software architecting fields, which can be a reasonable option for future work on the topic.

The maturity of these tools and proposals has been considered for further insight into the relationship between the proposed machine learning approaches and software architecting. While the categorisation is partly subjective, which is due to the subjective nature of the understanding of the terms *practical* and *experimental*, it offers nonetheless a rough estimation of what the current state of the research is in terms of usability.

Figure 4.2 shows that out of 25 papers that passed the study selection, five papers, or 20% can be considered to propose *practical* tools or approaches, while 20 papers or 80%, are still in an *experimental* stage. Among the practical approaches and tools are, studies 6 and 11, which have already been presented earlier in the thesis.

Study 6 which, is about detecting architectural tactics and runtime verifying them, has trained and tested successfully the approach on 74 open-source projects and even offers an implemented tool [Ge+22]. The tool is implemented, has been applied to real-world examples, and can be tested on other examples as well. It should be considered however that only 10 architectural tactics have been implemented for detection. Despite the limitation in detectable architectural tactics, the tool can be considered *practical*.

Study 11 proposes the explanation of design decision trade-offs with principal component analysis and decision tree learning. It tests the approach successfully on a system specifically designed for research, which includes the implementation of data extraction, aggregation and normalisation processes. It is only limited to the domain of possible use cases, which are static component and connector architectures. Therefore, the approach can be considered *practical*.

The disparity between the number of *practical* and *experimental* proposed tools or approaches seems to be an indicator of the young age of the research field since most studies seem to explore new approaches or tools, which is why they are *experimental*.

5.3 Research Question 3

To answer RQ3, I have aggregated common future work directions and challenges that appear in the studies. The most common future work proposition with seven papers is to expand the use cases of the proposed tool or approach. In the case of study 3, "A Proposed Model-Driven Approach to Manage Architectural Technical Debt Life Cycle", which uses machine learning and model checking to identify and manage architectural technical debt and its repayment strategies, a machine learning model to understand diagrams or methods for automatic text extraction methods from video or voice recordings could be considered for future work [PCA19]. This shows that many approaches

are new and have room for improvement. The second most common future work direction with six studies is to test more machine learning models for the tool or approach proposed by the study. The purpose of this research direction is to explore potential performance improvements by comparing the performance of different machine learning models. Increasing the dataset and creating some form of recommendation system are both proposed four times for future work. The future direction of creating some form of recommendation system implies a more assistive nature of the machine learning approaches. The proposition to increase the dataset will be further discussed in the part about challenges in the studies. Finally, three studies proposed to test their approaches or tools on other examples to further verify their results. This again shows that the review topic is recent and the approaches are new.

In regards to current challenges, the most common challenge in the studies was the data gathering for datasets. This is explicitly a limiting factor in nine papers, as seen in figure 4.4. The training and testing of machine learning models require datasets high in quality and quantity to achieve good performance. A lack of data requires the consideration that the results of a study cannot be generalised since the study may be too overfitted to the limited dataset. Therefore, it would be a very useful contribution to create a large dataset of software architecture artefacts from informal developer conversations to formal documentation and implementations. Such a dataset would not only reduce the effort required to gather data, but it would also form a common basis for the comparison of research results since the initial dataset can be the same. Another common challenge of three studies is limitations in the domain in which the proposed tool or approach can be applied. This challenge stems primarily from the resource and time limitations of the respective studies and supposedly does not affect the general applicability of the proposed approaches according to the papers in question. Nonetheless, a study that verifies the general applicability of these approaches can be considered for future work.

Unfortunately, a very apparent research gap in the current state of research is the analysis of the approaches in terms of their efficiency. Efficiency can mean in this context aspects like the time, computational power or budget that are needed to implement and use the approaches and tools, compared to their performance. Only the studies with ID 15, 25 and 32 offer some form of efficiency analysis.

Overall, the current state of research regarding machine learning approaches that support software architecting implies a positive outlook in terms of the performance of the approaches and tools. However, there is currently a lack of efficiency analysis in the performed studies, and the software architecture fields *Planning*, *Maintenance* and *Quality Assurance* have the potential for more research. Furthermore, the challenge of limited datasets has to be tackled since machine learning models heavily rely on them.

6 Conclusion and Future Work

6.1 Conclusion

This thesis presents the process and results of a systematic literature review with the goal of offering an overview of the topic of machine learning approaches that support software architecting.

For this purpose, the guidelines proposed by Kitchenham and Charters were followed as a basis for the review. In the section 3.1, the current need for a systematic literature review is explained, and the crafting process of the research question "What machine learning approaches exist for the support of software architecting?" is shown. Following that, the choice of the data sources has been explained, the search string for the search of studies crafted, and relevant selection criteria defined, such that a search and data aggregation resulted in a body of 38 studies.

Section 3.3 deals with the presentation of information that has been gathered from the body of 38 studies. Based on the information from the *Data Extraction* it is possible to see, that machine learning approaches exist especially in the fields of *Evaluation* with 40%, followed by *Requirements Engineering* and *System Design* with 20% and 16%, respectively, which shows a current trend towards applying machine learning to support *Evaluation*. In contrast, the other fields *Planning*,*Quality Assurance* and *Maintenance* have fewer studies which, indicates a potential to expand the research in these fields. Furthermore, the research topic is young since, with 80%, most studies are considered experimental.

In terms of common challenges, data gathering was considered by nine studies to be difficult, while also three papers are limited in the application domain of their proposed approach or tool. Lastly, the review shows a research gap in the efficiency analysis of the approaches or tools in the papers.

6.2 Future Work

For future work, one can consider expanding existing approaches or tools as described in their respective studies. The expansion can be an addition of new use cases, testing of multiple machine learning models for a potential performance benefit, or the implementation of some form of recommendation system.

Adding new use cases can be, for example, the training of a machine learning model on a dataset made up of source code in another programming language and the implementation of a respective parser.

When a study only tests its approach or tool on a certain group of machine learning models and there is no important reason for the choice of the group, then it is reasonable to explore other options in the future to determine the model with the best performance.

Since machine learning can offer new insights by evaluating data or generating artefacts, it is possible to extend existing approaches by using them as recommendations or as the basis for recommendations.

Furthermore, exploring more approaches or studies in the fields *Planning*, *Quality Assurance*, or *Maintenance* might also contribute to the current state of research.

Conducting an efficiency analysis for an approach or tool is also a future direction since only three of the examined studies performed one. An efficiency analysis is interesting since machine learning requires a lot of computation power and thus energy, which may play a role when the approaches are scaled up to match larger and or more difficult cases. Finally, the creation and sharing of a dataset containing artefacts and documentation of one or multiple software architectures may be another future work idea.

The contribution of a large dataset containing multiple architectures with many artefacts, including documentation, informal developer conversations about the architecture, description models, operation data or source code, would be very useful since data gathering for machine learning training is the most common challenge in the current state of research.

Bibliography

[Aka+22] M. Akalanka et al. "Software Complexity Automation Tool for Industrial Practices with Qualitative and Quantitative Aspects." In: *2022 4th International Conference on Advancements in Computing (ICAC)*. 2022, pp. 453–458. DOI: 10.1109/ICAC57685.2022.10025257 (cit. on p. 11).

[Bis06] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 9780387310732 (cit. on p. 1).

[BRT20] M. Barenkamp, J. Rebstadt, and O. Thomas. "Applications of AI in classical Software Engineering." In: *AI Perspectives* 2 (July 2020). DOI: 10.1186/s42467-020-00005-4 (cit. on p. 3).

[Cam+23] J. Camara et al. "ExTrA: Explaining architectural design tradeoff spaces via dimensionality reduction." In: *Journal of Systems and Software* 198 (2023). ISSN: 01641212. URL: %5Curl%7Bhttp://dx.doi.org/10.1016/j.jss.2022.111578%20%7D (cit. on p. 26).

[Cit] Citavi. *Citavi 6*. https://www.citavi.com/en (cit. on p. 9).

[DVV24] R. Dhar, K. Vaidhyanathan, and V. Varma. "Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical study." In: *IEEE ICSA 2024*. 2024 (cit. on p. 26).

[Els] Elsevier. *Compendex on Engineering Village*. https://www.elsevier.com/products/engineering-village/databases/compendex (cit. on p. 7).

[Flo+21] A. Florean et al. "A comparison of machine learning-based text classifiers for mapping source code to architectural modules." In: *CEUR Workshop Proceedings*. Vol. 2978. 2021 (cit. on pp. 25, 26).

[Gar24] Gartner. *Information technology (IT) spending worldwide from 2012 to 2024, by segment (in billion U.S. dollars)*. https://www.statista.com/statistics/268938/global-it-spending-by-segment/. 2024 (cit. on p. 1).

[Ge+22] N. Ge et al. "ArchTacRV: Detecting and Runtime Verifying Architectural Tactics in Code." In: *Proceedings - 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022*. 2022, pp. 566–576. DOI: http://dx.doi.org/10.1109/SANER53432.2022.00074 (cit. on pp. 26, 27).

[Hu23] K. Hu. *ChatGPT sets record for fastest-growing user base*. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/. 2023 (cit. on pp. 1, 2).

[IEE] IEEE. *About IEEExplore*. https://ieeexplore.ieee.org/Xplorehelp/overview-of-ieee-xplore/about-ieee-xplore (cit. on p. 7).

[KC07] B. Kitchenham and S. M. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. Software Engineering Group Keele University, Department of Computer Science University of Durham, 2007 (cit. on pp. 5–7, 9, 12, 29).

[LL23] J. Ludewig and H. Lichter. *Software Engineering: Grundlagen, Menschen, Proze-sse, Techniken*. 4th ed. dpunkt. verlag, 2023. ISBN: 9783864905988 (cit. on p. 1).

[MA21] E. Mohammed and E. Alemneh. "Identification of Architecturally Significant Non-Functional Requirement." In: *2021 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*. 2021, pp. 24–29. DOI: 10.1109/ICT4DA53266.2021.9672235 (cit. on p. 26).

[MG16] A. C. Müller and S. Guido. *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016. ISBN: 9781449369897 (cit. on p. 1).

[PCA19] B. Perez, D. Correal, and H. Astudillo. "A Proposed Model-Driven Approach to Manage Architectural Technical Debt Life Cycle." In: *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. 2019, pp. 73–77. DOI: 10.1109/TechDebt.2019.00025 (cit. on p. 27).

[RR22] V. Raj and S. Ravichandra. "Enhanced Service Point Approach for Microservices Based Applications Using Machine Learning Techniques." In: *Communications in Computer and Information Science*. Vol. 1575 CCIS. 2022, pp. 78–90. DOI: http://dx.doi.org/10.1007/978-3-031-09469-9_7 (cit. on p. 26).

[RRW15] J. O. Ringert, B. Rumpe, and A. Wortmann. *Architecture and Behavior Modeling of Cyber-Physical Systems with MontiArcAutomaton*. Shaker Verlag, 2015 (cit. on p. 1).

[Rum+21] B. Rumpe et al. "Artifact and Reference Models for Generative Machine Learning Frameworks and Build Systems." In: *20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE '21)*. GPCE 2021. Association for Computing Machinery, 2021. ISBN: 9781450391122. DOI: 10.1145/3486609.3487199. URL: https://doi.org/10.1145/3486609.3487199 (cit. on p. 2).

[Va09] O. Vogel and et al. *Software-Architektur. Grundlagen - Konzepte - Praxis*. Spektrum Akademischer Verlag Heidelberg, 2009. ISBN: 978-3-8274-2267-5. DOI: https://doi.org/10.1007/978-3-8274-2267-5 (cit. on p. 1).

[Vil] E. Village. *Engineering Village*. https://www.elsevier.com/products/engineering-village (cit. on p. 7).

[Wan+23] S. Wang et al. “Machine/Deep Learning for Software Engineering: A Systematic Literature Review.” In: *IEEE Transactions on Software Engineering* 49.3 (2023), pp. 1188–1231. DOI: 10.1109/TSE.2022.3173346 (cit. on p. 3).

[Wu22] X. Wu. “Architecture-based Failure Prediction via LSTM and Bayesian Network in Service-oriented Systems.” In: *2022 4th International Conference on Applied Machine Learning (ICAML)*. 2022, pp. 36–40. DOI: 10.1109/ICAML57167.2022.00015 (cit. on p. 25).

[Yan+20] Y. Yang et al. “A Survey on Deep Learning for Software Engineering.” In: *CoRR* abs/2011.14597 (2020). DOI: https://doi.org/10.48550/arXiv.2011.14597. URL: https://arxiv.org/abs/2011.14597 (cit. on p. 3).