Architecting ML-enabled systems: challenges, best practices, and design decisions

Roger Nazir, Alessio Bucaioni[a], Patrizio Pelliccione[b]

[a] *Mälardalen University, Västerås, Sweden*
[b] *Gran Sasso Science Institute, L'Aquila, Italy*

Abstract

Context. Machine learning is increasingly used in a wide set of applications ranging from recommendation engines to autonomous systems through business intelligence and smart assistants. Designing and developing machine learning systems is a complex process that can be eased by leveraging effective design decisions tackling the most important challenges and by having a good system and software architecture.

Goal. The research goal of this work is to identify common challenges, best design practices, and main software architecture design decisions of machine learning enabled systems from the point of view of researchers and practitioners.

Method. We performed a mixed method including a systematic literature review and expert interviews. We started with a systematic literature review. From an initial set of 3038 studies, we selected 41 primary studies, which we analysed according to a data extraction, analysis, and synthesis process. In addition, we conducted 12 expert interviews that involved researchers and professionals with machine learning expertise from 9 different countries.

Findings. We identify 35 design challenges, 42 best practices and 27 design decisions when architecting machine learning systems. By eliciting main design challenges, we contribute to best practices and design decisions. In addition, we identify correlations among design challenges, decisions and best practices.

Conclusions. We believe that practitioners and researchers can benefit from this first and comprehensive analysis of current software architecture design challenges, best practices, and design decisions.

Keywords: Machine learning, software architecture, challenges, best practices, design decisions.

1. Introduction

Machine learning (ML) is attracting increasing interest in the marketplace of applications and services. Companies that produce software witnessed increasing customers in their software systems that require ML-based components and solutions [1]. ML solutions are used in several fields, including computer defence, computational biology, autonomous vehicles, robotics, and Internet of Things (IoT). Software engineering is one of the key instrument supporting the growth of ML and promoting its deployment and success [2]. ML is also requiring a substantial change on organisational aspects and development processes [3].

Properly architecting ML-enabled systems, both in terms of the pipeline that produces the ML model and the system that uses and interacts with the Artificial Intelligence (AI)/ML components, is of key importance for various factors:

- *Maintenance and Evolution are important aspects of ML Models Life Cycle* [3]: software architecture plays a key role in building solutions for deploying reliable AI systems timely, for model maintenance, and for evolution. Moreover, ML components may degrade at a different rate than the rest of the system components and this should be properly architected and engineered.
- *Quality aspects of ML-enabled systems* [4]: properly understanding and managing key quality attributes of ML-enabled systems avoid failures, e.g., boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, and changes in the external world [5]. This aspect is very important for ML-enabled systems that concern with the CACE principle of "changing anything changes everything" [5].
- *Integration with other components*: properly designing and integrating ML components with other components, as well as understanding architectural dependencies, is a key aspect for the successful development of effective ML-based systems. In fact, as highlighted by [5], "only a small fraction of real-world ML systems is composed of the ML code [..] The required surrounding infrastructure

Email addresses: alessio.bucaioni@mdu.se (Alessio Bucaioni), patrizio.pelliccione@gssi.it (Patrizio Pelliccione)

is vast and complex".

- *Uncertainty management* [6]: properly managing uncertainty permits to avoid significant resource expenses during development. This includes properly (i) architecting data pipelines from training, to deployment, maintenance, and evolution, and (ii) accounting for retraining models and incorporating new data.

Researchers and experts are investigating best design practices in the software architecture design for ML systems and further about the ML systems complexity [2]. However, at the best of our knowledge, we are still missing a study that analyses how practitioners perceive and use ML design decisions in the architecture of ML systems and applications.

In this work, we aim at filling this research gap, by identifying software architecture design challenges, best practices, and architectural design decisions for ML-enabled systems. We formulated the research goal according to the Goal-Question-Metric in [7] as follows: *identify* (purpose) *common challenges, best design practices, and main software architecture design decisions* (issue) *of ML-enabled systems* (object) *from the point of view of researchers and practitioners* (viewpoint).

This research goal is then refined into the following Research Questions (RQs):

- *RQ1*: *What are the most common software architecture design challenges in ML-enabled systems?* Architecting ML systems may be a daunting task due to the inherit demands of such systems with respect to heterogeneous qualities such as performance, scalability, etc.
- *RQ2*: *What are the best practices in architecting ML-enabled systems?* The inherit complexity of ML systems favour the birth of a set of guidelines that are know to produce good outcomes in given contexts.
- *RQ3*: *What are the main software architectural design decisions for ML-enabled systems?* Architectural design decisions are practical design choices that provide concrete solutions to the design of the software architecture.

Answers to RQ1 provide a set of commonly found software architecture design challenges that can help researchers and practitioners in understanding and avoiding them. By answering RQ2, we provide a catalogue of commonly used best practices that can be exploited by researchers and practitioners to conceive an architecture for ML-enabled systems. Answering to RQ3 provides a list of design decisions that can concretely help researchers and practitioners in taking decisions when architecting their ML-enabled systems.

To provide an answer to the above RQs, we make use of a mixed research methodology able to extract insights from both the state of the art and the state of practice. Specifically, we adopted two complementary research methods, i.e., systematic literature review and expert interviews. From an initial set of 3038 peer-reviewed publications, we identified 41 *primary* studies, which we analysed thoroughly following a meticulous data extraction, analysis, and synthesis process. We performed 12 expert interviews with researchers and professionals with ML expertise from 9 different countries. Also, we compared and discussed the findings from the systematic literature review and expert interviews. Finally, we highlight interesting correlations among challenges, best practices, and design decisions.

The remainder of this work is organised as follows: Section 2 describes the research methodology we used for performing the study. Then, Sections 3, 4, and 5 report the main findings for RQ1, RQ2, and RQ3, respectively. Section 6 summarizes the findings by correlating challenges, best practices, and design decisions. Section 7 discusses related works. The paper concludes in Section 8 with final remarks and future research directions.

2. Research method

We designed and conducted this work using a research method that combines a set of complementary research methodologies for compensating single method limitations. In particular, we built on the systematic literature reviews guidelines in software engineering by Kitchenham and Brereton [8] and on the interviews and empirical software engineering guidelines by Wohlin et al. [9]. Figure 1 shows the process we followed, which consists of three main phases being planning, conducting, and documenting.

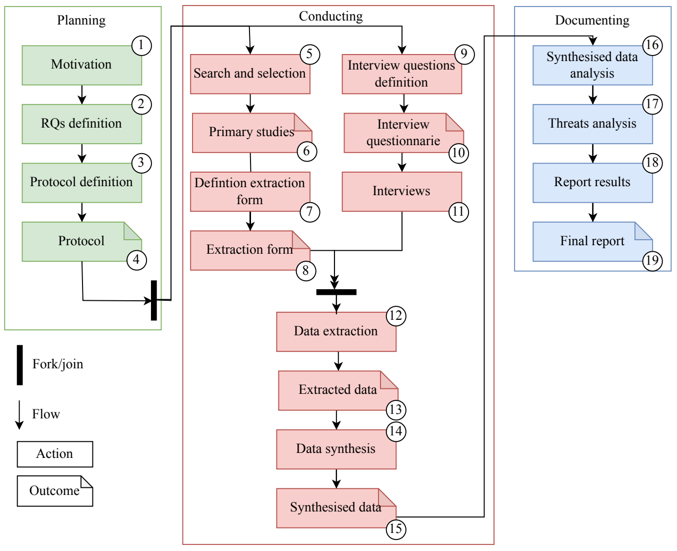

Figure 1: Overview of the adopted research method.

Planning. The main goals of the planning phase are to (i) establish the need for this work, (ii) define an overreaching research goal (RG) and related research questions (RQs), and (iii) define the research protocol to be followed by the researchers to carry out the study in a systematic manner. The main output of the planning phase is a detailed research protocol describing the main activities to be performed for the systematic literature review and the interviewees. It is worth remarking that the protocol prescribes different activities for the systematic literature review and the interviews. The former involves the activities marked with tags 5, 6, 7 and 8. The latter involves activities marked with 9, 10 and 11.

Conducting. In the conducting phase, we performed all the activities defined in the research protocol (activities 5 to 15), which are search and selection (5), definition of the data extraction form (7), definition of the interview questions (9), interviews (11), data extraction (12), and data analysis (14). We started the search and selection step by performing an automatic search of peer-reviewed literature on four scientific databases and indexing systems. We defined and used selection criteria for filtering the identified studies to obtain the set of primary studies to be included in later activities of the review. We complemented the initial automatic search with an exhausting backward and forward snowballing as suggested by Wohlin et al. [10]. Starting from the research questions and by systematically and iteratively using the standard key-wording process [11], we defined the set of parameters that we used for classifying and comparing the primary studies. According to Hackett et al. the most common data collection method in surveys is interviews and questionnaires as they have the potential of providing effective and thorough inputs on specific subjects [12]. We defined the questions for the interviews using only open-ended questions as they allow respondents to address the question in depth hence obtain more knowledge. Eventually, the interview questionnaire consisted of 15 questions. We sent interview invitations to several industry professionals, and academic researchers having ML expertise. We contacted 74 ML architects and engineers through LinkedIn and 9 of them agreed to be interviewed. We also contacted 15 ML experts through our contact networks and 3 of them agreed to the interview. A total of 12 ML experts answered the interview questionnaire. In the data extraction step, we analysed each of the identified primary studies to fill the data extraction forms. We collected and aggregated the filled forms for data analysis and synthesis. In addition, we analysed the interview data using a five-step process based on the coding technique [13]. In the data analysis step, we analysed the extracted data. The main goal of such an analysis was to provide answers to the research questions. To this end, we performed both quantitative and qualitative analysis.

Documenting. In the documenting phase, we analysed and documented possible threats to validity affecting the study, and documented the results. To support the independent replication and verification of this work, we provide a complete and public replication package [14] containing the data from search and selection, the complete list of primary studies, and the data extraction forms, as well as the interview data.

2.1. Search and Selection Strategy

We collected a set of 41 relevant research studies for our investigation by following the steps shown in Figure 2.

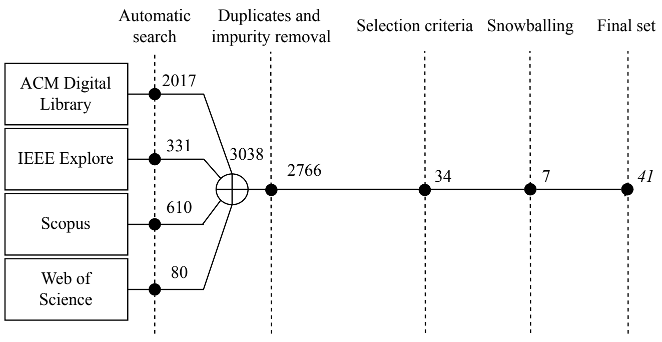

Figure 2: Overview of the search and selection process

We started with selecting four of the largest and most complete scientific databases and indexing systems in software engineering [8] that are IEEE Xplore Digital Library, ACM Digital Library, SCOPUS, and Web of Science (Table 1).

The selection was based on their high accessibility and good reputation in supporting systematic reviews in software engineering [15]. We exercised the selected databases and indexing systems using a search string that we created starting from our research goal and questions.

According to the guidelines by Kitchenham and Brereton on systematic literature reviews in software engineering, we defined a control group of studies that we used for validating our research string [8]. The control group included the studies by Amershi et al. [3], Washizaki et al. [2] and by Washizaki et al. [16]. In addition, we tried to keep the search string simple and inclusive so to collect as many relevant research works as possible. We discuss threats to validity related to the search string in Section 2.5. The search string is:

> *("software architecture" AND ("machine learning" OR "artificial intelligence" OR AI OR "deep learning") AND (challenge OR problem OR issue) AND ("best practice" OR "recommendation" OR tactics))*

We performed the search during a period from January to June 2021 and obtained an initial set of *3038* potential peer-reviewed studies. We followed the selection process

Table 1: Electronic databases, indexing systems and search engine used in this study

Name	**Type**	**URL**
IEEE Xplore Digital Library	Electronic database	`http://ieeexplore.ieee.org`
ACM Digital Library	Electronic database	`http://dl.acm.org`
SCOPUS	Indexing system	`http://www.scopus.com`
Web of Science	Indexing system	`https://www.webofscience.com/wos/woscc/basic-search`

proposed by Ali and Petersen [17] and we removed impurities and duplicated and applied selection criteria to the search results to ensure an objective selection. The set of selection criteria that we used for filtering search results studies is reported in the following.
Inclusion criteria:

1. Studies subject to peer review [9].
2. Studies written in English.
3. Studies available as full-text.
4. Studies discussing software architecture challenges or architecture best practices or design decisions for machine learning systems.

Exclusion criteria:

1. Secondary and tertiary studies (e.g., systematic literature reviews, surveys, etc.).
2. Studies in the form of tutorial papers, short papers, poster papers, editorials, and manuals, because they do not provide enough information.

We included in the next step only those studies meeting all the inclusion criteria and none of the exclusion criteria. We iteratively applied the selection criteria on title, abstract and full text and obtained a set of 34 potentially relevant studies. Then, we performed a closed recursive backwards and forward snowballing activity [10], which helped us in minimising potential bias concerning construct validity [18]. As a result, we found 79 additional peer-reviewed studies that we filtered using the same selection criteria. Eventually, we selected 7 additional studies and obtained the final set of *41*, which is attached as appendix of this paper.

2.2. Definition of the extraction form

We created the extraction form shown in Table 2 to extract and collect data from the primary studies. The extraction form is composed of three facets, one per research question. For each facet, we carried out a key-wording systematic process to develop an extraction form that could fit the set of primary studies and take their characteristics into account [11]. We first collected keywords and concepts by reading the full text of the primary studies and, then, we clustered the elicited keywords and concepts into categories by using a process similar to the sorting phase of the grounded theory methodology [19]. We refined the extraction form when new information deemed relevant was collected and was not already captured by the current extraction form. Finally, we re-analysed the primary studies according to the refined form and extracted new data.

Table 2: Data extraction form facets, clusters and categories

Facet	**Category**	**Description**	**Value**
RQ1	Challenges	most common design challenges in architecting machine learning systems as identified in the studies	String
RQ2	Best practices	list of best practices in architecting machine learning systems as identified in the studies	String
RQ3	Design decisions	list of design decisions in architecting machine learning systems as identified in the studies	String

2.3. Interview questions definition, interviews and interviewees

In this step, we built on the guidelines by Shull et al. [20] and performed a total of 12 interviews. Interviews are among the most common and effective data collection methods as they help in gathering effective and thorough inputs on given subjects from practitioners across organisations [20]. We conducted online, semi-structured, in-depth interviews for collecting quantitative and qualitative data; this allowed us to enhance our findings on architecting ML systems [21]. We composed the interviews using 15 open-ended questions that helped us minimise possible threats to validity and gather more detailed answers. We selected our target population to be researchers and practitioners with experience in ML systems. We created an interview invitation letter that outlined the main focus of the interview, its expected duration and the measures we took for ensuring privacy and confidentiality. We distributed the invitation letter to 89 researchers and practitioners from our contact networks or found via the LinkedIn[1] professional social network.

A total of 12 ML experts answered the interview questionnaire. Table 3 provides information on role, ML experience, domain and affiliation of the interviewees. The pool of interviewees included 7 ML engineers, 4 ML researchers and 1 manager. Of these, 9 interviewees worked in industry while 3 worked in academia. Looking at the ML experience, the majority of experts had 5 years of experience or more (41%). 33% of the interviewees had 3 to 5 years experience, while the remaining experts had less than 3 years of experience. All the interviews were carried out online using the Zoom platform[2] and video- and audio-recorded (except for one interview as the expert did not agree to it). The anonymized transcripts and notes taken during the interviews are available in the replication package [14]. The interviews lasted between 30 and 45 minutes.

2.4. Data extraction, synthesis and analysis

We extracted, analysed, and synthesised the extracted data using the guidelines presented by Cruzes et al. [22]. In

[1]http://www.linkedin.com
[2]https://zoom.us

Table 3: Statistical representation of the interviewees

ID	Role	ML Experience	Domain	Affiliation
I1	ML engineer	1 - 3 years	Industry	Confidential
I2	ML engineer	$\leq$ 1 year	Industry	Upland software
I3	ML researcher	3 - 5 years	Academia	Chalmers University
I4	ML researcher	$>$ 10 years	Academia	University of L'Aquila
I5	ML engineer	3 - 5 years	Industry	Quantiphi
I6	ML engineer	$>$ 10 years	Industry	PlayGround XYZ
I7	ML researcher	5 - 10 years	Academia	Chalmers University
I8	ML engineer	$>$ 10 years	Industry	Cubic Corporation
I9	ML researcher	$\leq$ 1 year	Industry	Cresta
I10	ML engineer	3 - 5 years	Industry	Inception Institute of Artificial Intelligence
I11	ML manager	$>$ 10 years	Industry	Macquarie group
I12	ML engineer	3 - 5 years	Industry	Visionet Systems Inc.

this research, we performed both quantitative and qualitative analyses combining content analysis [23] and narrative synthesis [24]. The combined analyses helped us in categorising and coding challenges, best practices and design decisions under thematic categories. The latter analysis technique provided a detailed explanation and interpretation of the findings coming from the former analysis. We used the so-called line of argument synthesis process [9] for finding trends and collecting information on each facet of the data extraction form. First, we analysed each primary study and interview individually for classifying its main features according to the parameters in the data extraction form. Later, we analysed the whole set of primary studies and interviews to uncover and reason about potential patterns. Finally, we grouped and cross-tabulated extracted data and compared pairs of facets of the data extraction form for identifying possible relations among different facets. To this end, we used contingency tables for extracting and evaluating relevant pair-wise relations.

2.5. Threats to validity

We carried out this research according to well-established guidelines for empirical studies in software engineering including those by Kitchenham and Brereton [8] and by Wohlin et al. [9]. Hereafter, we describe the main threats to validity according to the scheme by Wohlin et al. [9] and elaborate on mitigation strategies.

Internal validity is a concern that arises when the design of a study may compromise the accuracy of the results [9]. To minimize this risk, we employed a research protocol that adhered to established guidelines for systematic studies, surveys, and interviews in software engineering. The protocol was developed and validated by all authors through consensus. To further ensure the validity of our data, we applied rigorous descriptive statistical methods for analysis. To minimise the risk of finding practitioners missing the required knowledge, we relied on our knowledge network and through LinkedIn where experience and expertise is visible.

Construct validity is a concern that arises when the connection between theory and observation may be compromised [9]. To mitigate this risk, we ensured that the primary studies selected for our research were representative of the population defined by our research questions by following a well-defined and validated protocol. We further minimised the risk of data extraction errors by using a framework based on the identified questions. To minimise risks related to a non representative search string, we identified main terms and synonyms or abbreviations. Eventually, we tested the string prior to the search and selection and observed it identified a set of known relevant studies. Each author independently checked the extracted data from the studies, and any doubts were resolved through annotations and consensus. One potential threat in the survey method is the possibility of hypothesis guessing or confirmation bias, where respondents adjust their answers to align with the main goal of the study. To prevent this, we posed the questions objectively and used references to relevant sources.

External validity is a concern that arises when the results and outcomes of a study may not be generalizable to a wider population [9]. In our research, the main concern that could impact external validity is whether the set of primary studies we selected is representative of the state-of-the-art and practice. To mitigate such a potential threat, we targeted four different databases and indexing systems among the largest and most complete in the field of software engineering [15]. Besides, we complemented the automatic search with closed recursive backward and forward snowballing. However, it is important to note that because the literature review was conducted in 2021, the results may not offer insights into the most current trends in this extensively studied field. Furthermore, these best practices may be context-dependent and may not be universally applicable, nor can they be considered a universally accepted set of procedures.

Conclusion validity is a concern that arises when the relationship between the extracted data and the obtained findings may compromise the credibility of the conclusions drawn [9]. To mitigate this risk, we applied best practices from systematic studies and survey guidelines throughout our research. This includes documenting every step of our research and providing a public replication package to ensure transparency and replicability. Additionally, we reduced potential bias during the data extraction process by using an extraction form. All authors participated in data extraction, analysis, and synthesis steps. We also took measures to mitigate other threats, such as lack of expert evaluation and fishing for results.

3. Design challenges in architecting machine learning systems (RQ1)

By answering this research question, we aim at identifying the main challenges affecting Software Architectures (SAs) for ML systems. According to the synthesised data from the systematic literature review and the interviews, there are 6 main categories of challenges listed in Table 4.

Table 4: Main design challenge categories identified in the primary studies and by the interviewees.

Category	Primary study	Interviewee
Architecture	[P18], [P22], [P26], [P28], [P30], [P32], [P38], [P39]	I4, I7
Data	[P15], [P16], [P38]	I1, I2, I3, I5, I6, I7, I8
Evolution	[P27], [P34]	I9, I12
Quality assurance	[P14], [P22]	I1, I10
Model	[P8], [P22], [P39]	-
Software development life cycle	[P4]	I10, I11

The primary studies and the interviewees identified a common set of 5 design challenge categories: Quality Assurance (QA), data, architecture, evolution, and Software Development Life Cycle (SDLC). Primary studies identified an additional category, i.e., model. Architecture and data are by far the most mentioned categories (29% each) and they are followed by a cluster of 4 challenge categories being evolution (12%), QA (9.6%), model (9.6%) and SDLC (9.6%). In the following subsections, we detail each identified category by presenting and discussing the list of specific sub-challenges for each category.

3.1. Architecture design challenges

In the architecture category, we find a number of different architectural challenges as shown in Table 5.

Table 5: Main architectural design sub-challenges as identified in the primary studies and by the interviewees.

Specific architectural challenge	Source
Failure recovery in micro-service architectures	[P18]
Hard to achieve heterogeneous redundancy	[P22]
Reuse in ML-based systems is hard	[P26]
Lack of knowledge required to build ML-based systems	[P28], I4
The use of ML components might introduce uncertainty	[P30]
Lack of architectural expertise	[P32]
Difficult to identify design smells	[P38], [P39]
Run-time architecture due to distributed execution	I7

Table 6: Main data design sub-challenges as identified in the primary studies and by the interviewees.

Specific data challenge	Source
Data management	[P15], I2, I8
Data visualisation	I2
Data dependencies	[P38]
Data observability	I1
Privacy	I3
Data Accuracy	I5
Use cloud with high-volume of data	I5
Establishing a proper ML infrastructure for managing data	I7, [P16]
Use of batching scoring for saving in the database	I6

Jin et al. observe that micro-service architectures are particularly challenging when used for developing ML systems with respect to failure recovery as the failure of a service may be propagated to other services if the fault is not promptly amended [P18]. In his paper, Serban observed that heterogeneous redundancy is hard to achieve in ML-based systems [P22]. This is because only a few algorithms achieve the needed accuracy for ML tasks and all these algorithms exhibit the same weaknesses [P22]. Kusmenko et al. argue that the use of neural networks with ML as reusable building blocks with clear interfaces is still challenging [P26]. Muzaffar et al. investigate the use of ML for building mobile robots and conclude that the selection and design of an architecture for a mobile robot that satisfies both functional and quality attribute requirements is a big challenge due to less available knowledge of computer algorithms and probability theory [P28]. ML knowledge is important for developers that need to understand the data and the ML model, interviewee I4 says. Serban et al. noticed that the use of ML components may introduce uncertainty when evaluating the reliability of software architecture design [P30]. Prior information on the uncertainty of ML components employed at design time is often incomplete and their usage can influence other components in the system. Bhat et al. argued that it is challenging to identify and quantify architectural expertise in ML systems [P32]. ML design smells are challenging to find [P38]. Design smells manifest in several ways. For example, using multiple languages in the development of ML systems often increases the cost of effective testing and makes it more difficult to transfer ownership to other team members [P38]. Another design smell is that maintaining the prototyping environment is costly, and small scale rarely reflects reality at full scale [P38]. One interviewee (I7) identified run-time architectures for ML as challenging with respect to the distributed execution of the ML model.

3.2. Data design challenges

Table 6 shows the data design sub-challenges. Castellanos et al. identified data management, e.g., pre-processing and preparation, as significant design issues in ML systems

Table 7: Main evolution design sub-challenges as identified in the primary studies and by the interviewees.

Specific evolution challenge	Source
Support continuous change and evolution	[P27]
Updatability & maintainability of ML-based systems	[P34], I9
Scalability of the model	I12

with consequences lasting even after the system development [P15]. For example, to infer contextual information on extensive data as scientific imaging is challenging and time-consuming as a large amount of similar data is needed as input to the ML training system [P15]. Another challenge related to data management is reported by an interviewee (I2); he indicated that the cleaning of the data is cumbersome but crucial for guaranteeing that it is not biased and it does not break the algorithm. The same interviewee (I2) stated that data visualisation is also challenging as bioinformatic information is not common, and current techniques are rarely applied. I8 argued that data preparation is difficult especially making statistics of it. Sculley et al. looked at the dependencies between lines of code and configuration and concluded that in an ML system that is actively being developed, the number of configuration lines tends to considerably outnumber the number of lines of code and such an increment in configuration lines is challenging to handle [P38]. Data observability was reported as challenging by I1 when dealing with large ML models. One interviewee (I3) noted that one of the main challenges related to data is how to ensure privacy and confidentiality. Interviewee I5 identified 2 challenges, i.e., data accuracy and cloud. Data accuracy and completion of data are crucial when training the models of ML systems. Cloud is regarded as not suitable when a high volume of data is handled. One interviewee (I7) stated that establishing a proper infrastructure for, e.g., storing, accessing, and updating data efficiently, is also one of the main challenges. Biondi et al. remarked that ML systems have high demands on setting up and maintaining the prototyping environment as small-scale environments usually do not reflect reality in full scale [P16]. However, such demands are challenging to meet. Related to this, another interviewee (I6) acknowledged as challenging of the use of batching scoring for saving in the database.

3.3. Evolution design challenges

Table 7 shows the evolution design sub-challenges. Möstl et al. recognised that cyber-physical systems (CPSs) developed using ML have design challenges in managing the system and its operational environment needs for continuous change and evolution [P27]. As an example, they mention the interaction design challenges related to the dynamic allocation of software when multi-core architectures are used [P27].

Both Baylor et al. [P34] and an interviewee (I9) refer to updatability as one of the main challenges. In fact, input data changing over time would require updating ML-based systems since updates in models can impact system outcomes and its performance [P34]. The scalability of the model was reported as a design challenge by one interviewee (I12). In particular, the interviewee remarked that when ML models do not scale, the engine can get busy and lose some requests.

3.4. Quality assurance design challenges

Table 8 shows the QA design sub-challenges. Serban et al. identified several QA-related design challenges in the development of ML-based safety-critical systems [P22]. Some of these challenges are: (i) formal verification is either impossible or impractical [P22], (ii) ML systems that are sensitive to variations in distribution [P22], (iii) limited scenario testing [P22], and (iv) fault tolerance [P22]. One interviewee (I1) remarked that being able to explain why an ML system makes a prediction is an important challenge especially when the size of the model is increasing.

Scheerer et al. argue that deductive verification and model checking would be very useful for increasing qualities such as safety, robustness, or dependability. However, because ML is fundamentally probabilistic and non-linear in nature, methods to ensure system correctness are barely applicable and only marginally relevant [P14]. One interviewee (I10) stated that a lack of documentation can affect the quality of ML systems together with a lack of QA guidelines.

3.5. Model design challenges

Table 9 shows the model design sub-challenges. Amershi et al. identify two main design challenges related to the ML model, which are managing and versioning, and reuse [P8]. First, maintaining and versioning models required for ML systems is far more complicated and demanding than maintaining and versioning other types of data in software engineering [P8]. In fact, ML/AI models are more difficult to manage as separate modules than other software components because models can get entangled in complicated ways and exhibit non-monotonic error behaviours [P8]. The authors claim that this issue is worsened

Table 8: Main QA design sub-challenges as identified in the primary studies and by the interviewees.

Specific QA challenge	Source
Formal verification	[P22]
Variation in distribution	[P22]
Limited scenario testing	[P22]
Fault tolerance	[P22]
Limited reasoning on system and code	[P22]
Explainability	I1
Deductive verification	[P14]
Lack of documentation	I10

by the lack of knowledge of the problems and best practices for ML model maintenance, due to ever-evolving research in both ML infrastructure and algorithms [P8]. Secondly, model customization and reuse may be difficult considering that some of the required abilities are not commonly found in software teams.

Table 9: Main model design sub-challenges as identified in the primary studies and by the interviewees.

Specific model challenge	Source
Managing and versioning models	[P8]
Model customization and reuse	[P8]
Selecting appropriate models	[P39]
Increasing complexity of models	[P22]

Schelter et al. claim that wrong management of the ML model can lead to poor performance of ML systems [P39]. Model management involves training, maintenance, deployment, monitoring, organisation, and documentation of ML models. Incorrect model management can result in poor performances and high maintenance costs [P39]. In addition, Serban identified other challenges related to the increasing complexity of models and their opacity [P22].

3.6. Software development life cycle design challenges

Table 10 shows the software development life cycle (SDLC) design sub-challenges. Among the SDLC design challenges, Wan et al. highlighted that the reuse of traditional, non-ML, software processes is problematic [P4]. ML or non-ML processes have different practices with respect to requirements, design, testing/quality, process, and management. ML system architectures generally include data gathering, data cleansing, feature engineering, modelling, execution, and deployment [P4]. In contrast, non-ML system architectural design is a more creative approach that implements different structural divisions of software components and provides behavioural descriptions [P4]. The distributed architecture style is commonly favoured for ML systems due to the large volume of data. In addition, ML systems have less emphasis on low coupled components than non-ML software systems: even though various features of ML systems have distinct capabilities, development teams are closely linked [P4]. This is also remarked by an interviewee, I11, who claims that the development of ML systems should not be driven by processes for non-ML systems. One interviewee (I10) noticed that most practices like continuous integration are not followed and that increases the complexity at the time of system completion. Similar to Wan et al. [P4], the interviewee claimed that the lack of available documentation negatively affects the reuse of processes.

Table 10: Main SDLC design sub-challenges as identified in the primary studies and by the interviewees.

Specific SDLC challenge	Source
Reuse and adoption of non ML-processes	[P4], I11
Continuous integration for ML-based systems	I10
Lack of available documentation	[P4]

We identified several design challenges that affect ML-enabled systems and that should be taken into account when architecting these systems. Primary studies and interviewees shared some challenges, but they also reported on different challenges. The challenge that is largely identified by primary studies and interviewees concerns data design challenges, including data management, dependencies, observability, accuracy, privacy, visualization, and so on. The most rated category of challenges identified in primary studies is architecture, spanning from the adoption of specific architectural styles to reusability, identification of bad smells, and how to deal with uncertainty. Other challenge categories - also important, but supported by fewer primary studies and interviews - are detailed in the section.

RQ1 findings

4. Best practices in architecting machine learning systems (RQ2)

By answering this research question, we aim at identifying the main best practices in architecting ML systems. According to the synthesised data from the systematic literature review and the interviews, there are 7 main categories of best practices listed in Table 11.

Table 11: Main best practices categories identified in the primary studies and by the interviewees.

Category	Primary study	Interviewee
Architecture	[P6], [P17], [P28], [P30], [P32], [P36], [P38], [P39]	I3, I5, I10, I12
Quality assurance	[P14], [P16], [P18], [P22], [P24], [P38]	I6, I9
Software development life cycle	[P4], [P11], [P20], [P35]	I7, I9, I11, I12
Hardware & platform	[P25], [P34], [P40]	I8, I9
Model	[P8], [P26]	I1, I4
Evolution	[P1], [P39]	-
Data	-	I2

The primary studies and the interviewees identified a common set of 4 best practices categories: QA, architecture, model, and Hardware (HW) & platform. Primary studies identified 2 additional categories, i.e., evolution and SDLC, while interviewees identified one additional category, i.e., data. Architecture (32%) and QA (21.6%) are by far the most mentioned categories. They are followed by SDLC and HW & platform (13.5% each) and model (10.8%). Finally, evolution and data are reported by two studies and one interviewee. In the following subsections, we detail each identified category by presenting and discussing the list of specific sub-practices for each category. As evolution and data account for only two best practices and one best practice, respectively, we present them together in Section 4.6

4.1. Architecture best practices

In the architecture category, we find a number of different architectural best practices ranging from specific architectures (e.g., in-memory distributed learning architectures [P17] or three layers [P6]) to design smells [P38, P39].

Table 12: Main architectural best practice as identified in the primary studies and by the interviewees.

Specific architecture best practice	**Source**
Use in-memory distributed learning architecture for sophisticated learning and optimisation techniques	[P17]
Use the Siemens four-views architecture for better separation of concerns	[P28]
Explicitly model uncertainty for assessing its propagation and impact	[P30]
Exploit knowledge and experience of architects and developers	[P32]
Single container, single mode and multiple node patterns	[P36]
Use Plain-Old-Data, Multiple-Language, and Prototype smell to identify design smells	[P38], [P39]
Use three layers architecture for separating the business logic from the ML-specific aspects	[P6]
Use the microservice architecture for better focusing on the business functionalities and for facilitating maintenance and cohesion	I3, I5, I12,
Use unified architecture and minimise architectural decisions before deployment	I3
Use the client-server pattern for pipeline management	I10
Aim for low cohesion to improve modifiability	I10

Table 12 shows the architectural best practices. Panousopoulou et al. recommended to use in-memory distributed learning architectures to achieve sophisticated learning and optimisation techniques on scientific imaging data sets: they are very effective in removing distortion on ML scientific imaging databases [P17]. The four-view architecture designing approach (i.e., the conceptual, the module, the execution architecture, and the code architecture views) ensures a better separation of concerns and, hence, decreases the development complexity of mobile robotic systems developed using ML [P28]. Schleier-Smith et al. highlighted the benefit of using a three layers architecture for separating the business logic from ML components [P6]. They show the benefit by focusing on troubleshooting ML systems that might have tightly coupled functions, e.g., inference engine derived from data, and business logic code from design. In this context, the three layers architecture (i) assists engineers in breaking down the failure and traces it to the business logic part or ML components, and (ii) allows engineers to rollback the inference engine independently of the business logic when the inference engine encounters some issues. Several interviewees (I3, I5, I12) recommended the use of micro-service architectures for several reasons including the following: (i) microservice architectures enable the engineers to concentrate on building the business functionalities rather than writing glue code, (ii) microservice architectures are easier to maintain as compared to monolithic architectures as they tend to use smaller and independent components, and (iii) the low cohesion among these components helps in increasing the modifiability of ML systems (I10). I10 also suggested to use the client-server pattern for reducing the risks of breaking pipelines. Another interview (I3) suggested using unified architectures and try to minimize architectural decisions before development. Serban et al. stated that it is best to explicitly model the intrinsic uncertainty of ML components and assess how it propagates and impacts other elements in the system at the designing stage [P30]. Burns et al. suggested using a single container, single-node patterns, and multiple-node patterns for improving reusability of components and distributed development [P36]. They argue that with the above patterns it is simpler to distribute implementation across various teams and reuse components in new situations. These include the ability to upgrade components separately and the ability to write them in different languages. According to Bhat et al. [P32] it is important to properly identify individuals who could be involved in tackling new design concerns. In fact, the experience and knowledge of architects and developers play a crucial role in design decisions making: experienced architects and developers use efficient decision-making strategies. The best practice to find design smells is to focus on three types of ML design smells that are Plain-Old-Data, Multiple-Language, and Prototype smells [P38, P39]. The Plain-Old-Data smell refers to the fact that too often complex information used and produced by ML systems are encoded with plain data types such as raw floats and integers. The second smell refers to the use of multiple languages that often increases the cost of effective testing and makes transferring ownership to other team members more difficult. The Prototype smell refers to the dangerous practice of using the prototyping system as a production solution.

4.2. Quality assurance best practices

Table 13 shows the QA-specific best practices. In their paper, Biondi et al. discussed several best practices for improving certifiability, safety, time predictability and security [P16]: (i) Certifiability - use of strict and certified coding standards when developing safety-critical ML components, (ii) Safety - implement proper mechanisms for tolerating faults and failures that may occur in complex software routines; (iii) Security - explicitly designing and

Table 13: Main QA best practices as identified in the primary studies and by the interviewees.

Specific QA best practice	Source
Certifiability - coding standards	[P16]
Safety - mechanisms for fault-tolerance	[P16], [P22]
Time predictability - techniques for improving confidence	[P16]
Security - design for security	[P16]
Use principal component analysis method for components interoperability	[P18]
Federated learning simulation framework (FLSim)	[P24]
Debt for data testing, reproducibility, process management and debt to culture	[P38]
Preliminary classification using static, monitor, a posteriori and non-analysability	[P14]
Standardisation of the training and testing process	I6
Test-driven development	I9

Table 14: Main SDLC best practices as identified in the primary studies and by the interviewees.

Specific SDLC best practice	Source
Main phases when architecting ML-based systems	[P20]
Importance of documentation	[P35]
Main activities and design of ML-based systems	[P4]
Divide et impera	[P11]
Adapt SDLC to product type, organisation and business goals	I7
Ensure proper ML infrastructure	I7
Object-oriented development	I9
Prototyping	I11
Avoid hiring new resources while building the ML framework	I11
Avoid changing technology	I11
Identify whether the ML-based system requires real-time capabilities or not	I12

developing ML systems to defend vulnerable sections of the code that may be subject to cyber-attacks, and (iv) Time predictability - static, monitor, a-posteriori analysability, and non-analysability are the best practices for preliminary classifications, that in turn can provide confidence to the design process for ML systems. Concerning safety, Serban recommends to focus on guaranteeing that safety-critical systems developed using ML do not reach hazardous conditions [P22]; to this end, they suggest to turn off components to reach a safe state quickly. Li et al. recommend the use of FLSim that is a reusable and extensible federated learning simulation framework [P24]. FLSim commonly assists deep learning, machine learning frameworks, for example, PyTorch and Tensor-Flow. The authors suggest to use FLSim for creating simulators for the above mentioned type of frameworks. Jin et al. stated that the principal component analysis method is known to assist in reducing the dimension of data sets and extend the components interoperability [P18]. Sculley et al. identified best practices related to the reduction of debt for data testing, debt for reproducibility, and debt in process management [P38]. One interviewee (I9) remarked on the importance of following a test-driven development for QA and suggested the use of the Jupiter Notebook. Another interviewee (I6) argued that a standardisation of the training and testing process would improve the verifiability of the ML models.

4.3. Software development life cycle best practices

Table 14 shows the software development life cycle best practices. Anjos et al. advocated that proper documentation increases the efficiency, reusability, reproductivity, and shareability of ML-based systems and can assist in designing them [P35]. In fact, a lack of documentation hampers use, reuse, and extension of ML-based systems.

Spalazzi et al. focused on the development of ML-based digital forensics systems. These systems are part of the forensic science that deals with recovering and investigating the information contained in digital devices [P20]. For these systems, Spalazzi et al. suggested following a four phases development process that includes: seizure, acquisition, analysis, and reporting [P20]. Wan et al. [P4] provide various best practices related to the software development life cycle. First, they identify the activities that are often relevant when architecting ML-based systems: data gathering, data cleansing, feature engineering, modelling, execution, and deployment [P4]. Even though various features of ML-based systems have distinct capabilities, development teams are closely linked; for example, the performance of data modelling is dependent on data processing. Moreover, the large volume of data often favours the selection of distributed architectures. Finally, a detailed design for ML-based systems is often unfeasible due to systems complexity as, e.g., data modelling might contain tens to hundreds of ML algorithms [P4]. Muccini et al. suggested using a *divide et impera* approach that is to brake down the ML-based system design into sub-concerns that can be handled with proper and tailored design decisions [P11]. One interviewee (I11) suggested tailoring the SDLC to accommodate prototyping. The interviewee also recommended avoiding changing technology as well as hiring new resources as it may negatively affect the ML systems under development. Similarly, it can lead to changes or updates in the data architecture. Another interviewee (I12) recommends to investigate whether or not the system needs real-time capabilities before starting its design. When dealing with general purpose ML-based systems, I9 recommends the use of test-driven development coupled with Object-Oriented programming and Jupiter Notebook. I7 remarked the importance of a proper ML infrastructure and of the training and deployment processes. For example, Federated Learning requires a distributed architecture hence the team needs to design a distributed and dynamic system.

4.4. Hardware and platform best practices

Table 15: Main HW & platform best practices as identified in the primary studies and by the interviewees.

Specific HW & platform best practice	Source
Use TensorFlow-based learner	[P34], [P40], I8
Jupiter notebook	I9
Use ML Cloud technologies	[P25]

Table 16: Main model best practices identified in the primary studies and by the interviewees.

Specific model best practice	Source
Select modelling languages supporting specific concerns	[P26]
Consider training a partial model in combination with transfer learning for better performance	[P8]
Focus on data to have a correct and "fresh" model	I1
Align the creation and training of the model to requirements	I4

Table 15 shows the HW & platform-specific best practices. Baylor et al. suggested the use of TensorFlow-based learner implementation with support for continuous training and serving with production-level dependability [P34]. Similarly, Abadi et al. advocated the use of TensorFlow as a way of providing improved data visualisation [P40]. In fact, thanks to its graphical approach, TensorFlow assists the development team in debugging the nodes reducing the need for code inspections [P40]. One interviewee suggested the use of TensorFlow and pipelining to enhance the security of ML systems (I8) while another interviewee (I9) suggested the use of Jupiter notebook when dealing with general purpose ML-based systems. Fomin et al. recommended using ML cloud technologies for increasing the quality of cutting tool states recognition in the industry [P25].

4.5. Model best practices

Table 16 shows the specific best practices related to the model. When building an artificial neural network in ML systems, Kusmenko et al. recommended selecting modelling languages taking into account three specific concerns: network architecture, network training, and data set model [P26]. Network architecture plays a pivotal role as it constrains the organisation of neurons and connections among them that define the data flow [P26]. Once properly defined, the network architecture needs to be trained: a developer can modify the training procedure without changing the architecture or a developer can combine existing architectures and training models without changing the models at all [P26]. Finally, the compiler needs to know where to look for training data and how to load it to train a network. Furthermore, the data set must be divided into training and test data [P26].

Amershi et al. suggested a best practice for object identification systems using ML [P8]. They suggested training a partial model using existing general data sets (e.g., ImageNet for object identification) and then combining them with specialised data using transfer learning for better performance [P8]. One interviewee (I1) suggested focusing on analysing data so as to always have a model that is correct and not obsolete. Another interviewee (I4) focused on the practice of aligning the creation and training of the model according to system requirements.

4.6. Evolution, and data best practices

Table 17: Main evolution and data best practices as identified in the primary studies and by the interviewees.

Specific evolution best practice	Source
Train a single model for time series and retrain it each time a new forecast needs to be made	[P39]
Consider having methods to adjust the parameters of noise-cleaning algorithms when light changes	[P1]
Specific data best practice	**Source**
Build the grids (data preparation technique) for natural language techniques	I2

Table 17 shows the specific best practices related to evolution (first two rows) and data (last row). Schelter et al. proposed an evolution best practice when developing large-scale forecasting problems and time series prediction systems using ML [P39]. They suggested training a single model per time series and retraining the models every time a new forecast needs to be created [P39]. Some classical forecasting techniques can be used to develop similar systems, such as auto-regressive integrated moving average models, exponential smoothing methods, state-space formulation, etc.

Wnag et al. focused on ML systems for image recognition and suggested a best practice that takes into account the performance of these systems with respect to environmental changes [P1]. For example, the object recognition rate for these systems is higher when the environment is brighter, i.e., with more light. The best design practice to increase the efficiency of recognition is to adjust the algorithm parameters when the intensity of the light changes [P1]. Simultaneously, to complete image enhancement, the composition model of related algorithms may need to be reconstructed [P1]. Furthermore, all pattern recognition algorithms and the composition model builder should be capable of checking the information of related sensors when necessary [P1].

Grid is a data preparation technique that transforms data as other hyperparameters of the modelling pipeline. One interviewee (I2) recognised that the grid searching data preparation technique suits the processing of natural language hence the interviewee recommended best practice is to build the grids when developing ML systems for natural language processing.

We identified several best practices improving the development of ML-enabled systems. Both primary studies and interviewees identified most of the best practices categories although they also reported on exclusive ones. Similar to the challenges, the most mentioned best practice category is architecture with practices spanning from use of specific patterns, architectures or tools to recommending low cohesion for maintainability. Other less mentioned practices are detailed in the section.

RQ2 findings

5. Design decisions in architecting machine learning systems (RQ3)

By answering this research question, we aim at identifying the main design decisions that are taken when architecting ML-based systems. According to the systematic literature review and the interviews, there are 7 main categories of design decisions listed in Table 18.

The primary studies and the interviewees identified a common set of 5 design challenge categories data, architecture, model, evolution and HW & platform. Primary studies identified 2 additional categories, namely quality assurance and software development life cycle. Architecture (25%) and HW & platform (21.4%) are the most mentioned categories. They are followed by SDLC and model (14% each), and data and evolution (10.7%). QA is the least mentioned design decision. In the following subsections, we detail these 7 main architectural design decisions. Each design decision has a dedicated subsection, with an exception of data, evolution, and quality assurance design decisions that are discussed together in Section 5.5. Indeed, architectural design decisions have some connections with the best practices we described in Section 4 whereas architectural decisions are more concrete. However, this does not mean that each best practice is refined by design decisions. We discuss how best practices relate to design decisions and how they both connect to challenges in Section 6. When possible, for each decision we report advantages, disadvantages, class of systems for which the decision works

Table 18: Main design decisions identified in the primary studies and by the interviewees.

Category	**Primary study**	**Interviewee**
Architecture	[P18], [P22], [P28]	I7, I9, I10, I11
Hardware & platform	[P15], [P25], [P40], [P41]	I8, I12
Software development life cycle	[P20], [P21], [P24], [P38]	I1, I3, I7, I8
Model	[P8], [P26]	I5, I6
Data	[P17]	I2, I7
Evolution	[P1], [P36]	I4
Quality assurance	[P16], [P34]	-

best (Good for) or should be avoided (Not good for), and examples clarifying the decision.

5.1. Architecture design decisions

Table 19 shows the HW & platform-specific design decisions. Several studies and interviewees focused on the importance of choosing the right architecture as the main design decision. Both Jin et al. [P18] and one interviewee (I9) suggested the use of micro-service architecture.

Table 19: Main architecture design decisions identified in the primary studies and by the interviewees.

Specific architecture design decision	**Source**
When architectural patterns are not available, perform an early evaluation of architectural safety methods.	[P22]
Choose an appropriate architecture, both overall ML architecture and run-time architecture, based on the input data to train. This general design decision is opened in the following architectural decisions, by referring to specific architectural styles.	I11, I7
Micro-service architecture	[P18], I9, I11
Pros: help decompose a system, reduce coupling, promote flexibility ([P18]). **Good for**: ML-based natural language processing systems (I9). **Not good for**: large systems where the input data change frequently (I11).	
Siemens four views	[P28]
Good for: ML-based robot navigation component.	
Client-server architecture	I10
Pros: promote low coupling and high cohesion, and enhance security. **Good for**: object recognition and image processing systems.	

Jin et al. proposed to use micro-service architecture as a way of decomposing a big service into discrete services that can help reduce the system coupling and provide more flexibility [P18]. The interviewee (I9) advocated that micro-service architectures are a better design decision when developing ML-based natural language processing systems as they assist in performing better data cleaning that, in turn, helps the accurate parsing of the document in natural language. However, the use of micro-service architecture is not supported by all. One interview (I11) stated that such an architecture is not optimal for large systems where the input data change frequently. So the interviewee's main design decision is to choose an appropriate architecture based on the input data to train. Muzzafar et al. suggested the use of the Siemens four views architecture when developing ML-based robot navigation component [P28]. The choice of an appropriate architecture is highlighted as an important design decision by interviewee I7 that recommended considering the overall ML architecture as well as the run-time architecture in terms of, e.g., heterogeneous computing units. Interviewee I10 suggested opting for a client-server architecture when developing object recognition and image processing systems as this architecture can provide low coupling and high cohesion besides enhancing the system security (for example, with the use of a firewall). Serban et al. suggested including the evaluation process of architectural safety methods before moving to the development stages when developing safety-critical ML systems [P22]. They advocate that this step is crucial when existing architectural patterns are not available and new ones need to be developed [P22].

5.2. Hardware and platform design decisions

Table 20 shows the HW & platform-specific design decisions. Infrastructure as Code (IaC) is known as the method of managing and providing computer data centres through machine-readable specification files, rather than actual hardware setup or interactive configuration tools. Catellanos et al. pointed out that one effective design decision when building ML systems associated with databases, servers, and other IT infrastructure is to employ IaC so as to use the same structures and rules used for code development. This can reduce cost, time and risks associated with the IT infrastructure [P15].

Table 20: Main HW & platform design decisions as identified in the primary studies and by the interviewees.

Specific HW & platform design decision	Source
Use of Infrastructure as Code	[P15]
Pros: (i) permit to use for HW the same structures and rules used for code development, (ii) reduce cost, time, and risks of the IT infrastructure	
Use of Cloud technologies	[P25]
Pros: more efficient in cutting tools state recognition systems than other alternatives such as diagnostic feature selection using combinatorial analysis	
Use of TensorFlow	[P40], I8
Pros: higher performance and easy to scale	
Use an architectural solution based on Residue Number System (RNS)	[P41]
Pros: permit to reduce the number of resources used, both in terms of hardware and time **Good for**: Convolutional neural networks (CNN) and other ML computing operations	
Use different branches for training the pipelines	I12
Pros: permit to avoid the so-called pipeline jungles	

Fomin et al. advocated that the use of cloud technologies is more efficient in cutting tools state recognition systems than other alternatives such as diagnostic feature selection using combinatorial analysis [P25]. Chervyakov et al. described how to reduce the number of resources used, both in terms of hardware and time, for Convolutional neural networks (CNN) and other ML computing operations [P41]. They achieve this by proposing a CNN architecture based on Residue Number System (RNS) and a new Chinese Remainder Theorem [P41]. Abadi et al. recognised as beneficial the use of TensorFlow for large-scale ML systems [P40]. TensorFlow provides a higher level of performance that is also easy to scale [P40]. The authors recognised that other tools like Keras and PyTorch are better for smaller systems [P40]. One interviewee (I12) suggested that an important design decision when architecting ML-based systems is to separate the branches for the training of the pipelines from the training of the model avoiding the so-called pipeline jungles.

5.3. Software development life cycle design decisions

Table 21 shows the SDLC-specific design decisions. Spalazzi et al. discuss a design decision for ML-based digital forensics systems [P20]. According to the authors, these systems have four main phases that are seizure, acquisition, analysis and reporting. Their design decision is to explicitly focus on the four phases of digital forensics systems during the design-making process to achieve more design simplicity than applying traditional development stages [P20].

Berquand et al. focused on a design decision for ML systems that manipulate a large amount of spatial data [P21]. For such systems, they advocated that concurrent engineering methods and model-based system engineering are better design choices for analysing the space data and

Table 21: Main SDLC design decisions as identified in the primary studies and by the interviewees.

Specific SDLC design decision	Source
Explicitly focus on four main phases: seizure, acquisition, analysis, and reporting	[P20], I8
Pros: simplify the design with respect to the one obtained by applying traditional development phases **Good for**: ML-based digital forensics systems	
Adopt concurrent engineering methods and model-based system engineering	[P21], I8
Pros: lower the development time and productivity costs **Good for**: ML-based systems that manipulate a large amount of spatial data	
Use federation learning	[P24]
Pros: (i) enable continuous learning on end-user devices, (ii) reduce data loss, and (iii) improve the preservation of data privacy **Good for**: ML-based mobile computing systems	
Develop models separately to mitigate ML-based systems configuration	[P38]
Pros: (i) reduce cost in terms of time, computing resources, etc., (ii) help visualise the difference in the configurations, (iii) help to detect unused or redundant models in ML-based systems	
Use of prototyping to receive fast feedback about business goals fulfilment	I1
Pros: align the design of the ML systems to the related business goals	
Complement the expertise of architects with practical experience of ML developers	I3
Pros: improve the development process	
Exploit ML development and run-time architectures	I7
Pros: enable continuous ML	

space mission system design as compared to, e.g., sequential engineering methods [P21]. Concurrent engineering methods and model-based system engineering can deal with big space data and divide it into different stages instead of treating them sequentially. The different stages can run simultaneously potentially lowering the development time as well as productivity costs [P21]. Li et al. suggested federated learning as a design decision when developing ML-based mobile computing systems [P24]. The authors claimed that federated learning can enable continuous learning on end-user devices, reducing data loss as well as improving the preservation of data privacy [P24]. Configurations of ML-based systems are hard to modify and configuration mistakes can be costly in terms of time, computing resources, etc. According to Sculley et al. a design decision to mitigate ML systems configuration issues is to develop all models separately [P38]. This will help visualise the difference in the configurations as compared to other approaches [P38]. For example, the debugging model training approach can help detect common errors during the training of the model, but it is less efficient with respect to time and costs in eliminating configuration issues [P38]. Further, developing all models separately also helps to detect unused or redundant models in the ML-based systems [P38]. One interviewee (I1) suggested that an important design decision is to align the design of the ML systems to the related business goals. To ensure this, the interviewee suggests the use of prototyping hence the fast realise of system prototypes that can be sued to collect feedback from stakeholders with respect to business goals fulfilment. I7 focused on how to enable continuous ML by focusing on the ML development architecture and run-time architecture with heterogeneous computing units (e.g., CPU, GPU). I3 argued that a good decision is to leverage the practical experience of ML developers as a complement to the expertise of architects.

5.4. Model design decisions

Table 22 shows the model-specific design decisions. Two interviewees agreed that one important design decision is the selection of a model. Interviewee (I6) suggested selecting a model depending on the ML systems domain type for example batch system or real-time system. Such a design decision is also suggested by Amershi et al. that add that a wrong model may not help in fulfilling the requirements [P8]. For example, they claim that the EfficientNet model is better than the ResNet model for image classification systems due to the scaling of image dimensions by a fixed number of layers that can provide better performance in real-time ML systems [P8].

Table 22: Main model design decisions as identified in the primary studies and by the interviewees.

Specific model design decision	Source
Properly select a model based on the domain type	[P8], I6
Pros: help in fulfilling the requirements **Example**: EfficientNet model is better than the ResNet model for image classification systems	
Model and train neural processes in ML-based systems at a higher level of abstraction	[P26]
Pros: a DSL helps democratise the use of ML and makes more accessible modelling and training. **Example**: deep learning technologies are more accessible by expressing layered structures as YAML or prooftext descriptions or offering high-level Python interfaces like Keras and Lasagne.	
Make a general model and preserve it	I5
Pros: do not compromise the overall performance by permitting newly hired members to change it.	

Kusmenko et al. advocated that a design decision to achieve better performance concerns modelling and training neural processes in ML-based systems [P26]. They argued that it is preferable to have a domain-specific language (DSL) rather than dealing with low-level constraints. For example, deep learning technologies have been more accessible by expressing layered structures as YAML or prooftext descriptions or offering high-level Python interfaces [P26]. Interviewee I5 argued that it is important for the development team to first make a general model and then retain it and avoid newly hired team members changing it as this would impact the overall performance.

5.5. Data, evolution and quality assurance design decisions

Table 23 shows the data, evolution, and quality assurance-specific design decisions.

With respect to data design decisions, Panousopoulou et al. suggested adding data visualisation techniques in the design process as it can help express the relationships between data and computing tasks [P17]. In fact, large ML-based learning systems require a special focus on facilitating data analytics [P17]. Similarly, one interviewee (I7) stated that the main design decisions to take when building ML-based systems are on how to manage and access data.

I2 suggested always building the grids for ML-based natural language systems to enhance their performance. Particularly, the interviewee suggests using already existing solutions to reduce time consumption.

Evolution-specific design challenges relate to the use of the multi-node approach and how to adapt parameters to environmental scenarios. Components-based ML distributed systems are multi-node ML systems that increase efficiency and improve performance by handling large-scale input data and ML components [P36]. For these systems, Burns et al. suggested the use of the multi-node approach as it allows disturbed ML systems components to be upgraded independently [P36]. This allows the development team to handle the component as it grows even using different ad hoc technologies or languages [P36]. Wang et al. suggested design decisions for the development of ML-based mobile robots using visual capabilities [P36].

Table 23: Main data, evolution and quality assurance design decisions as identified in the primary studies and by the interviewees.

Specific data design decision	**Source**
Add data visualisation techniques in the design process	[P17]
Pros: (i) help expressing the relationships between data and computing tasks, (ii) help data analytics	
Manage and access the data	I7
Build grids for ML-based natural languages	I2
Pros: enhance their performance	
Specific evolution design decision	**Source**
Use of the multi-node approach	[P36]
Pros: (i) increase efficiency and improve performance, (ii) allow independent upgrade of components **Good for**: ML-based mobile robots using visual capabilities	
Adapt and evolve ML-based systems: adapt parameters to environmental scenarios [P21] or adapt the ML-based system software architecture to input data changes (I4)	[P1], I4
Pros: flexibility and robustness **Example**: the noise-cleaning algorithm should update its parameters to adapt to changes in light intensity	
Specific QA design decision	**Source**
Use of strict coding standards and the use of (safety) certification from authorised bodies	[P16]
Pros: improve quality	
Check the validation results before pushing the algorithm into the production environment and combine model validation with data validation	[P34]
Pros: (i) predict how a learning algorithm will behave on new data and (ii) detect corrupted training	

Their main design decision is to account for parameter changes in relation to environmental scenarios [P1]. For example, they suggested that the noise-cleaning algorithm should update its parameters to adapt to changes in light intensity [P1]. I4 suggested considering the evolution of the ML system accounting for input data change and updating the ML system software architecture accordingly.

When it comes to design decisions for QA, Biondi et al. suggested the use of strict coding standards and the use of (safety) certification from authorised bodies [P16]. Baylor et al. recognised model validation as an essential phase to predict how a learning algorithm will behave on new data [P34]. They suggest (i) always checking the validation results before pushing the algorithm into the production environment, and (ii) combining model validation with data validation to better detect corrupted training [P34].

We identified concrete design decisions that impact the development of ML-enabled systems. Primary studies and interviewees identified seven categories of design decisions of which five were identified by both studies and interviews. The architecture category collected the higher number of decisions. Micro-service architecture and other patterns such as the Siemens four views and the client-server ones were strongly suggested for different systems like natural-language processing, robotics and object recognition systems, respectively. Several design decision were provided for the HW & platform category, too. Other less mentioned practices are detailed in the section.

RQ3 findings

6. Summary of the findings

In this section, we highlight interesting correlations among challenges, best practices, and design decisions. They are clustered into six categories, namely architecture, data, evolution, QA, model, and SDLC, i.e., those that have interesting correlations. This is shown in Table 24. We omit those categories for which we could not identify any relations at all.

Architecture. The use of architectural patterns or styles is generally recognised as a best practice. Our best practices and design decisions identify benefits for specific architectures: e.g. the Siemens four view brings as benefit separation of concerns, while a microservice architecture facilitates maintenance and cohesion since it helps decompose the system, reducing coupling, and promoting flexibility. At the same time, it helps engineers and developers to focus on business functionalities. In the design decisions, we also identify the class of systems for which a specific pattern or style is good. For example, a microservice architecture is good for ML-based natural language processing systems, but it is not the ideal architecture for large systems where the input data change frequently. Architecture design decisions mainly refer to the adoption of architectural patterns and style; when a pattern is not available the recommendation is to perform an early evaluation of architectural safety methods. The use of specific architectural styles also introduces challenges; for instance, failure recovery is challenging in micro-service architectures. Among the other challenges, we mention uncertainty that is introduced by the use of ML components. Explicitly modelling uncertainty is a best practice for mitigating this challenge. No concrete architectural decision is related to uncertainty. Identifying design smells is also a challenge and the associated best practice recommends making use of techniques like Plain-Old-Data, Multiple-Language, and Prototype smell to identify design smells. Challenges to lack expertise and knowledge related to a best practice that recommends

Table 24: Summary of the findings.

Category	Sub-category	Design challenge	Best practice	Design decision
Architecture	Micro-service architecture	Failure recovery is challenging in microservice architectures	Use the microservice architecture for better focusing on the business functionalities and for facilitating maintenance and cohesion	**Pros**: help decompose a system, reduce coupling and promote flexibility. **Good for**: ML-based natural language processing systems. **Not good for**: large systems where the input data change frequently
	Siemens four-views	-	Use the Siemens four-view architecture for better separation of concerns	**Good for**: ML-based robot navigation components.
	Client-server	-	Use the client-server pattern for pipeline management	**Pros**: promote low-coupling and high cohesion, and enhance security **Good for**: object recognition and image processing systems
	Uncertainty	The use of ML components might introduce uncertainty	Explicitly model uncertainty for assessing its propagation and impact	-
	Design smells	Difficult to identify design smell	Use Plain-Old-Data, Multiple-Language, and Prototype smell to identify design smells	-
Data	Data management	Data pre-processing, cleaning and analysis	The grid searching data preparation technique suits the processing of natural language	Build grids for ML-based natural languages **Pros**: enhance their performance
	Data visualisation	Lack of techniques for bioinformatic information	-	Add visualisation techniques in the design process **Pros**: (i) help addressing the relationships between data and computing tasks, (ii) help data analytics
	Privacy	Ensure privacy and confidentiality	-	Properly manage ad access data
Evolution	Updatability & maintainability	Change in data invalidating models	Train a single model for time series and retrain it each time a new forecast needs to be made	-
			Consider having methods to adjust the parameters of noise-cleaning algorithm when light changes	Adapt and evolve ML-based systems: adapt parameters to environmental scenarios or adapt the ML-based system software architecture to input data change **Pros**: flexibility and robustness **Example**: the noise-cleaning algorithm should update its parameters to adapt to changes in light intensity
Model	Selection of proper models and/or languages	Select appropriate models	Select modelling languages supporting specific concerns	Properly select a model based on the domain type **Pros**: help in fulfilling the requirements **Example**: EfficientNet model is better than the ResNet model for image classification systems
			Model bias	Make a general model and preserve it **Pros**: do not compromise the overall performance by permitting newly hired members to change it
HW & platform	Cloud	-	Use ML Cloud technologies	**Pros**: more efficient in cutting tools state recognition systems than other alternatives such as diagnostic feature selection using combinatorial analysis
	TensorFlow	-	Use TensorFlow-based learner	**Pros**: Higher performance and easy to scale

exploiting the knowledge and experience of architects and developers.

Data. In the data category, we find nine design challenges, one best practice and three design decisions. From these, we identified three relations between design challenges and design decisions and one relation between best practices and design decisions. Undoubtedly, handling data is challenging in several aspects such as management, visualisation and privacy. Some of these challenges are tackled by specific design decisions as reported in Table 24. However, we found that a high number of data-related challenges like data dependencies and accuracy lack both best practices and design decisions.

Model. In the model category, we have four design challenges, four best practices and three design decisions. Considering these, we identified 2 relations between challenges, best practices and decisions. Both the identified relations aim at supporting the resolution of the challenging task of choosing proper models. In this respect, some of the best practices and design decisions suggest letting requirements and the domain type drive the selection of the modelling languages and models. We noticed that some model-related challenges like versioning of models are not tackled by any best practice or design decision.

Cloud technologies together with TensorFlows seem to be the most effective way of setting up the hardware and the platform for ML-enabled systems. In particular, cloud technologies seem to improve the efficiency of cutting tools state recognition systems while cloud technologies seem to provide a natural solution for scaling ML-based systems.

7. Related Work

To the best of our knowledge, this work represents the first mixed-method study assessing current software architecture design challenges, best practices and design decisions for ML systems. However, there is an existing body of literature that systematically reviews complementary aspects of ML systems or architectural aspects of software systems.

Amershi et al. report on a case study on software engineering for machine learning at Microsoft [3]. The study uses prior knowledge of developing AI applications and data science tools documented in the literature and identifies a software engineering development process and a set of development challenges. Eventually, the authors use a questionnaire involving 551 software engineers for validating the identified challenges. Compared to our work, the study by Amershi et al. addresses a much broader scope, which is the software engineering of machine learning systems. We only focus on architectural aspects of machine learning systems being design challenges, best practices and design decisions. Moreover, we use different research methods than those used by Amershi et al., i.e., systematic literature review and expert interviews.

Washizaki et al. perform a multi-vocal systematic literature review on design patterns for ML systems [2]. While the work by Washizaki et al. is relevant to our study, its scope is much narrower as the authors only analyse 10 peer-reviewed publications and 25 grey sources. Washizaki also conducted another study on product quality attributes, model quality attributes prediction, and ML pattern Pi [16]. The work starts with a literature review, which uncovers 15 ML trends. Then, the authors use surveys for understanding the sentiment of practitioners towards these trends. 43% of the respondents declared they reused previous solutions in the form of internal guidelines or trends. Eventually, the study recommends a larger adoption of existing ML patterns. Another systematic study on design patterns for ML systems is the work by Watanabe et al. [25]. In this study, the authors survey the grey literature and practitioners' knowledge and identify a total of 33 design patterns. Compare to our research, the works by Watanabe et al. and Washizaki et al. use similar research methods, but they focus on complementary aspects being design patterns.

Architectural patterns are a core concept in software architecture. In practice, finding and applying the most appropriate architectural patterns largely remains an unsystematic task linked to the architect's knowledge. Avgeriou et al. surveyed architectural patterns and proposed a pattern language that supersets existing architectural pattern collections and categorisations [1]. Other works that shed some light on architectural patterns in a specific domain are those by Apostolos et al. [26], by Growin et al [27], by Washizaki et al. [28] and by Shirwaikar et al [29]. The work by Apostolos et al. researches the state-of-the-art on the Gang of Four (GoF) design patterns by surveying 120 primary studies through a systematic mapping study. The work by Growin et al. focuses on reviewing design patterns for multi-agent application systems (MAS) and indicates a lack of MAS patterns. Washizaki et al. [28] and Shirwaikar et al [29] analyse and report on patterns for the Internet of Things (IoT) and patterns addressing security, respectively. Similar to our work, all the above studies employ a systematic approach to survey the literature. In our work, we do not focus on architectural patterns explicitly although some design decisions and best practices point to patterns. Wang et al. propose a scenario-based architecture for the reliability design of artificial intelligence software, which takes into account two main categories of scenarios, i.e., environmental and structure scenarios [30]. Using the proposed architecture, the quantitative reliability of the software could be evaluated and predicted with the design and scenario-based analysis to allow it to be eligible for safety-sensitive applications.

Mayer et al. performed a comprehensive survey of challenges, methods, and resources for scalable Deep Learning (DL) on distributed facilities [31]. The paper focuses on the scalability of DL systems that must continue to be enhanced to improve DL performance. Some of their findings point to DL infrastructures, parallel DL training approaches, multi-tenant resource planning, and training and model data management. They also reviewed and evaluated 11 existing open-source DL frameworks and concluded that more research efforts on DL systems are needed.

Muccini et al. highlight the different architecting practices that exist for ML-based software systems [32]. They start from their experience in architecting an ML-based software system for solving queuing challenges in one of the largest museums in Italy and identify four key areas of software architecture that need the attention of both ML and software practitioners with the aim of defining a standard set of practices for architecting ML-based software systems. Both our work and the work by Muccini et al. aim at providing concrete guidelines to practitioners approaching the development of ML systems. However, the work by Muccini et al. is a retrospective work based on a single use case. Our work surveys the current body of knowledge and complements it with practitioner knowledge.

8. Conclusions and future Work

In this work, we provided insights into design challenges, best practices, and design decisions when designing software architectures for ML-enabled systems.

The study is based on a systematic literature review and interview questionnaire that help in providing two complementary views being academic and industrial. While these views share most of the identified design challenges, best practices, and design decisions, we found some key differences as well.

Concerning design challenges (RQ1), we observed that the model category is only supported by primary studies, while the architecture challenges are much more popular in peer-reviewd publications than among interviewees. Moreover, even those interviewees mentioning architecture challenges, I4 and I7, are academic experts. Therefore, we can deduce that no architectural challenge has been identified by practitioners. Quality assurance challenges have been identified by both primary studies and interviewees. However, they have slightly different focuses with primary studies touching more on specific techniques for verification and validation and practitioners focusing on highlight explainability and lack of documentation.

Concerning best practices (RQ2), practitioners have not identified any evolution related best practices that were instead identified by some primary studies. On the contrary, data related best practices were only identified by interviewees. When focusing on architecture best practices, there are no common best practices among those proposed by interviews and those extracted from peer reviewed publications. The most popular best practice from interviewees concerns the use of micro-service architectures. These discrepancies may come from the fact that interviewees and papers focus on different domains. This interpretation comes from the design decisions discussed in Section 5, where we highlight which decision is appropriate for which domain; e.g. Siemens four-view is good for ML-based robot navigation. When focusing on QA best practices, interviewees only focus on testing while papers refer also to other techniques, e.g. simulation, time predictability and so on. Regarding SDLC best practices, interviewees recommend best practices covering also organisational aspects while the ones extracted from papers are mainly identifying the main phases or activities when architecting ML-based systems.

Concerning design decisions (RQ3), we identified less discrepancies among those coming from primary studies and those coming from data collected via interviewees. We did not identify quality assurance design decisions from practitioners, while we identified a couple of design decisions in primary studies.

It would be interesting to better investigate whether the identified discrepancies might be confirmed and in case to better understand the reasons for that. However, with the data we have we can not go further in the analysis and we leave this to future works. For instance this can be done by performing an in-depth analysis of the challenges and best practices identified in this study by conducting questionnaires and arranging a validation workshop with practitioners. However, we believe that reading about these differences is already valuable for academics and practitioners and it can stimulate discussion, cross-fertilisation, and analysis.

In addition, we would like to extend this study including the analysis of the so-called grey literature to capture a broader view on this topic. Eventually, we would like to examine how well the findings of this study can be applied to other areas within the broader field of AI. This would provide a more comprehensive understanding of the challenges and best practices in the field of ML. Another direction would be to develop a framework for ML teams. The framework would allow the development team to easily find relevant best design practices and software architecture design decisions for a given challenge.

Acknowledgment

The authors acknowledge the support of the PNRR MUR project VITALITY (ECS00000041), Spoke 2 ASTRA - Advanced Space Technologies and Research Alliance, and of the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI. The involvement of Mälardalen University in this research has been supported by KK-stiftelsen through the project MoDEV and by by the Excellence in Production Research (XPRES) Framework.

Selected Papers

[P1] J. Wang, G. Li, and Y. Pu, "A scenario-based architecture for reliability design of artificial intelligent software," in *International Conference on Computational Intelligence and Security*, 2010.

[P2] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Adoption and effects of software engineering best practices in machine learning," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020.

[P3] J. Schleier-Smith, "An architecture for agile machine learning in real-time applications," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[P4] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?" *IEEE Transactions on Software Engineering*, 2019.

[P5] J. Musil, A. Musil, and S. Biffl, "Introduction and challenges of environment architectures for collective intelligence systems," in *Agent Environments for Multi-Agent Systems IV*, 2015.

[P6] H. Yokoyama, "Machine learning system architectural pattern for improving operational stability," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2019.

[P7] A. Musil, J. Musil, and S. Biffl, "Major variants of the sis architecture pattern for collective intelligence systems," in *Proceedings of the 21st European Conference on Pattern Languages of Programs*, 2016.

[P8] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019.

[P9] H. Washizaki, H. Takeuchi, F. Khomh, N. Natori, T. Doi, and S. Okuda, "Practitioners' insights on machine-learning software engineering design patterns: a preliminary study," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020.

[P10] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Computing Surveys (CSUR)*, 2020.

[P11] H. Muccini and K. Vaidhyanathan, "Software architecture for ml-based systems: what exists and what lies ahead," in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021, pp. 121–128.

[P12] A. Ahmad and M. A. Babar, "Software architectures for robotic systems: A systematic mapping study," *Journal of Systems and Software*, vol. 122, pp. 16–39, 2016.

[P13] H. Washizaki, H. Uchida, F. Khomh, and Y.-G. Guéhéneuc, "Studying software engineering patterns for designing machine learning systems," in *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 2019, pp. 49–495.

[P14] M. Scheerer, J. Klamroth, R. Reussner, and B. Beckert, "Towards classes of architectural dependability assurance for machine-learning-based systems," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020, pp. 31–37.

[P15] C. Castellanos, B. Pérez, D. Correal, and C. A. Varela, "A model-driven architectural design method for big data analytics applications," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2020, pp. 89–94.

[P16] A. Biondi, F. Nesti, G. Cicero, D. Casini, and G. Buttazzo, "A safe, secure, and predictable software architecture for deep learning in safety-critical systems," *IEEE Embedded Systems Letters*, vol. 12, no. 3, pp. 78–82, 2019.

[P17] A. Panousopoulou, S. Farrens, K. Fotiadou, A. Woiselle, G. Tsagkatakis, J.-L. Starck, and P. Tsakalides, "A distributed learning architecture for scientific imaging problems," *arXiv preprint arXiv:1809.05956*, 2018.

[P18] M. Jin, A. Lv, Y. Zhu, Z. Wen, Y. Zhong, Z. Zhao, J. Wu, H. Li, H. He, and F. Chen, "An anomaly detection algorithm for microservice architecture based on robust principal component analysis," *IEEE Access*, vol. 8, pp. 226 397–226 408, 2020.

[P19] I. Alarcon, P. Gomez, M. Campos, J. Aguilar, S. Romero, P. Serrahima, and P. Breuer, "A holistic approach to intelligent automated control," in *International Conference on Information Technology for Balanced Automation Systems*. Springer, 1995, pp. 301–308.

[P20] L. Spalazzi, M. Paolanti, and E. Frontoni, "An offline parallel architecture for forensic multimedia classification," *Multimedia Tools and Applications*, vol. 81, no. 16, pp. 22 715–22 730, 2022.

[P21] A. Berquand, F. Murdaca, A. Riccardi, T. Soares, S. Generé, N. Brauer, and K. Kumar, "Artificial intelligence for the early design phases of space missions," in *2019 IEEE Aerospace Conference*. IEEE, 2019, pp. 1–20.

[P22] A. C. Serban, "Designing safety critical software systems to manage inherent uncertainty," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019, pp. 246–249.

[P23] V. Indhumathi and G. Nasira, "Fault tolerance in job scheduling through fault management framework using soa in grid," *ICTACT Journal on Soft Computing*, vol. 7, no. 2, 2017.

[P24] L. Li, J. Wang, and C. Xu, "Flsim: An extensible and reusable simulation framework for federated learning," in *International Conference on Simulation Tools and Techniques*. Springer, 2020, pp. 350–369.

[P25] O. Fomin and O. Derevianchenko, "Improvement of the quality of cutting tools states recognition using cloud technologies," in *Design, Simulation, Manufacturing: The Innovation Exchange*. Springer, 2020, pp. 243–252.

[P26] E. Kusmenko, S. Nickels, S. Pavlitskaya, B. Rumpe, and T. Timmermanns, "Modeling and training of neural processing systems," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2019, pp. 283–293.

[P27] M. Möstl, J. Schlatow, R. Ernst, N. Dutt, A. Nassar, A. Rahmani, F. J. Kurdahi, T. Wild, A. Sadighi, and A. Herkersdorf, "Platform-centric self-awareness as a key enabler for controlling changes in cps," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1543–1567, 2018.

[P28] A. W. Muzaffar, S. R. Mir, M. Latif, W. H. Butt, and F. Azam, "Software architecture of a mobile robot," in *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2015, pp. 102–107.

[P29] B. Vinayagasundaram and S. Srivatsa, "Software quality in artificial intelligence system," *Information Technology Journal*, vol. 6, no. 6, pp. 835–842, 2007.

[P30] A. Serban, E. Poll, and J. Visser, "Towards using probabilistic models to design software systems with inherent uncertainty," in *European Conference on Software Architecture*. Springer, 2020, pp. 89–97.

[P31] E. Di Buccio, A. Lorenzet, M. Melucci, and F. Neresini, "Unveiling latent states behind social indicators." in *SoGood@ ECML-PKDD*, 2016.

[P32] M. Bhat, K. Shumaiev, K. Koch, U. Hohenstein, A. Biesdorf, and F. Matthes, "An expert recommendation system for design decision making: Who should be involved in making a design decision?" in *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2018, pp. 85–8509.

[P33] B. Venthur, S. Dähne, J. Höhne, H. Heller, and B. Blankertz, "Wyrm: a brain-computer interface toolbox in python," *Neuroinformatics*, vol. 13, no. 4, pp. 471–486, 2015.

[P34] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1387–1395.

[P35] A. Anjos, M. Günther, T. de Freitas Pereira, P. Korshunov, A. Mohammadi, and S. Marcel, "Continuously reproducing toolchains in pattern recognition and machine learning experiments," 2017.

[P36] B. Burns and D. Oppenheimer, "Design patterns for container-based distributed systems," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[P37] P. P. Raut, N. R. Borkar, M. Student, A. Professsor, and S. Kamlatai, "Machine learning algorithms: Trends, perspectives and prospects," *International Journal of Engineering Science*, vol. 4884, 2017.

[P38] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in neural information processing systems*, vol. 28, 2015.

[P39] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On challenges in machine learning model management," 2018.

[P40] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX* *symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[P41] N. I. Chervyakov, P. A. Lyakhov, M. A. Deryabin, N. Nagornov, M. V. Valueva, and G. V. Valuev, "Residue number system-based solution for reducing the hardware cost of a convolutional neural network," *Neurocomputing*, vol. 407, pp. 439–453, 2020.

References

[1] H. Liu, S. Eksmo, J. Risberg, R. Hebig, Emerging and changing tasks in the development process for machine learning systems, in: Proceedings of the international conference on software and system processes, 2020.

[2] H. Washizaki, H. Uchida, F. Khomh, Y.-G. Guéhéneuc, Studying software engineering patterns for designing machine learning systems, in: 2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP), IEEE, 2019, pp. 49–495.

[3] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, T. Zimmermann, Software engineering for machine learning: A case study, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019.

[4] L. Pons, I. Ozkaya, Priority quality attributes for engineering ai-enabled systems, in: Association for the Advancement of Artificial Intelligence AI in Public Sector Workshop. Washington, DC, November 7-9, 2019.

[5] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, D. Dennison, Hidden technical debt in machine learning systems, in: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, NIPS'15, MIT Press, Cambridge, MA, USA, 2015, p. 2503–2511.

[6] I. Ozkaya, What is really different in engineering ai-enabled systems?, IEEE Software 37 (4) (2020) 3–6. doi:10.1109/MS.2020.2993662.

[7] V. R. Basili, G. Caldiera, H. D. Rombach, The Goal Question Metric Approach, in: Encyclopedia of Software Engineering, 1994.

[8] B. Kitchenham, P. Brereton, A systematic review of systematic review process research in software engineering, Information and software technology.

[9] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Computer Science, 2012.

[10] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Procs of EASE, 2014.

[11] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Procs of EASE, 2008.

[12] G. Hackett, Survey research methods, The Personnel and Guidance Journal 59 (9) (1981) 599–604.

[13] C. B. Seaman, Qualitative methods in empirical studies of software engineering, IEEE Transactions on software engineering.

[14] P. P. Roger Nazir, Alessio Bucaioni, Replication package for the paper, https://github.com/nazirroger7/Studying-Software-Architecture-for-ML (January 2023).

[15] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of Systems and Software.

[16] H. Washizaki, H. Takeuchi, F. Khomh, N. Natori, T. Doi, S. Okuda, Practitioners' insights on machine-learning software engineering design patterns: a preliminary study, in: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2020.

[17] N. B. Ali, K. Petersen, Evaluating strategies for study selection in systematic literature studies, in: Procs of ESEM, 2014.

[18] T. Greenhalgh, R. Peacock, Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources, BMJ.

[19] K. Charmaz, L. L. Belgrave, Grounded theory, The Blackwell encyclopedia of sociology.

[20] F. Shull, J. Singer, D. I. Sjøberg, Guide to advanced empirical software engineering, Springer, 2007.

[21] T. Punter, M. Ciolkowski, B. Freimut, I. John, Conducting on-line surveys in software engineering, in: 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings., 2003.

[22] D. S. Cruzes, T. Dyba, Recommended steps for thematic synthesis in software engineering, in: Procs of ESEM, IEEE, 2011, pp. 275–284.

[23] R. Franzosi, Quantitative narrative analysis, 2010.

[24] M. Rodgers, A. Sowden, M. Petticrew, L. Arai, H. Roberts, N. Britten, J. Popay, Testing methodological guidance on the conduct of narrative synthesis in systematic reviews: effectiveness of interventions to promote smoke alarm ownership and function, Evaluation.

[25] Y. Watanabe, H. Washizaki, K. Sakamoto, D. Saito, K. Honda, N. Tsuda, Y. Fukazawa, N. Yoshioka, Preliminary systematic literature review of machine learning system development process, arXiv preprint arXiv:1910.05528.

[26] A. Ampatzoglou, S. Charalampidou, I. Stamelos, Research state of the art on gof design patterns: A mapping study, Journal of Systems and Software.

[27] J. Juziuk, D. Weyns, T. Holvoet, Design patterns for multi-agent systems: A systematic literature review, in: Agent-Oriented Software Engineering, 2014.

[28] H. Washizaki, N. Yoshioka, A. Hazeyama, T. Kato, H. Kaiya, S. Ogata, T. Okubo, E. B. Fernandez, Landscape of iot patterns, in: IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT), 2019.

[29] P. Ponde, S. Shirwaikar, An exploratory study of the security design pattern landscape and their classification, International Journal of Secure Software Engineering (IJSSE).

[30] J. Wang, G. Li, Y. Pu, A scenario-based architecture for reliability design of artificial intelligent software, in: International Conference on Computational Intelligence and Security, 2010.

[31] R. Mayer, H.-A. Jacobsen, Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools, ACM Computing Surveys (CSUR).

[32] H. Muccini, K. Vaidhyanathan, Software architecture for ml-based systems: what exists and what lies ahead, in: IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN), 2021, pp. 121–128.