A Checklist of Quality Concerns for Architecting ML-Intensive Systems

Alessio Bucaioni[a], Rick Kazman[b], Patrizio Pelliccione[c]

[a] *Malardalen University, Vasteras, Sweden*
[b] *University of Hawaii, Honolulu, Hawaii, USA*
[c] *Gran Sasso Science Institute, L'Aquila, Italy*

Abstract

Background. Machine learning components are being deployed across nearly every business sector and their importance is continually growing. However, the engineering practices for building these systems remain poorly understood compared to those for conventional software systems.
Objective. This work provides practical guidance to support architects in designing and implementing machine learning-intensive systems, and identifies areas where there are gaps in understanding and achievement.
Method. Building on our prior research, we developed a checklist of quality concerns for architects of machine learning-intensive systems. This checklist was iteratively refined through expert interviews and subsequently validated in a workshop with experienced architects.
Results. The main result of this work is a comprehensive list of 40 checks, organized into two main categories and 16 subcategories. Also, we present the results of a workshop where the importance and degree of achievement of each check was assessed by 25 practicing architects of ML-intensive systems.
Conclusion. The findings of this study offer valuable support to architects in addressing the unique challenges of ML-intensive systems and provide guidance to practitioners and researchers in terms of where future work should be focused.

1. Introduction

Machine learning (ML) has become mainstream in software, being deployed in systems in nearly every business sector. However, due to its relative newness and its unique challenges, the state of engineering practice for building these systems continues to lag behind other types of software systems. A number of industry studies have reported that a majority of ML projects fail, typically not progressing beyond the prototype stage [1, 2]. There are several obvious culprits for these failures, which have been pointed out in the literature: the miscommunications between data scientists and software engineers, the relative immaturity of the technologies, the fast pace of technology evolution, and the ever-increasing quality demands placed upon ML-intensive systems. Adding to these challenges is the inherent stochastic nature of ML models, which can lead to inconsistent or unpredictable behavior. This lack of reliability may undermine trust and limit adoption in operational contexts where determinism and explainability are essential. Moreover, problems can be introduced at various stages—including the data science process, system design and deployment, and system evolution—further complicating the successful realization of ML projects.

In this paper, we will describe the challenges and concerns for building an ML-enabled system, by which we mean a software system that integrates one or more ML components. We have collected, analyzed, and validated these challenges and concerns in a three step multi-method empirical process.

We began by building upon a foundation of prior research. Specifically we built upon the results of our previous study [3], which identified key design decisions, challenges and best practices in ML-intensive systems. In addition, we conducted interviews with architects to catalog their quality concerns, resulting in a refined checklist of concerns supported by documentation such as scholarly citations or excerpts from interviews. Finally, we organized a workshop with practitioners to validate the checklist and to gather feedback on the perceived importance and current level of achievement of each check. In this context, we define a check as a guiding question or consideration that architects should assess when designing, implementing, or maintaining ML-intensive systems [4, 5, 6, 7]. Checks are not direct concerns themselves, but prompts aimed at verifying whether critical architectural aspects have been adequately addressed.

ML experts, data scientists, software engineers, and software architects have struggled to find a common language to discuss the concerns for ML-enabled systems. Although ML introduces some requirements and considerations that are not always obvious or well-articulated and documented during system design, in large part the concerns of ML-enabled systems overlap with those of conventional systems. That is to say, in most cases, we want

Email addresses: `alessio.bucaioni@mdu.se` (Alessio Bucaioni), `kazman@hawaii.edu` (Rick Kazman), `patrizio.pelliccione@gssi.it` (Patrizio Pelliccione)

these systems to be secure, to perform well, to be robust, to be easy to modify and extend, and so forth. However, in addition to these universal concerns, there is a set of these concerns that are driven by the nature of this being an ML system. Such systems need to address certain common requirements in a more stringent manner, such as privacy or model modularity. In what follows, we will describe areas where the design drivers of ML-intensive systems may deviate from current software engineering practices or simply be unfamiliar to software engineers, as well as those which are highly overlapping with conventional software.

1.1. Research Questions

To be more precise about our research objectives, we now detail the Research Questions (RQs) that motivated this study.

- **RQ1** - What are the most important design considerations for the architect of an ML-intensive system?
- **RQ2** - What design qualities of ML-intensive systems are difficult for architects to achieve in practice?
- **RQ3** - Where are there gaps between the importance and the achievement of design qualities of ML-intensive systems?

These research questions were posed as a way of focusing our investigation on the process of architecting ML-intensive systems, to identify: how it is similar to the software architecture design process in general [8], how it is different, and areas where it seems to currently be inadequate.

1.2. Contribution

ML-intensive systems are, quite simply, the future of software. This research provides an empirically grounded window into the problems facing architects of such systems. In particular, we have identified the most important areas of architectural concerns, and also the areas where there is the biggest gaps between importance and achievement. Therefore, these results highlight areas where future research and development must focus.

1.3. Roadmap

The paper is organized as follows. Section 2 describes the research methodology we followed to perform this study and discusses the threats to validity of the study. Section 3 presents the checklist for software architects we synthesized in our study. Section 4 details the validation workshop and its qualitative and quantitative results. Section 5 discusses the results of this study. Section 6 compares the work with related works. Finally, Section 7 discusses conclusions and future work directions.

2. Research Method

We designed and conducted this study using complementary research methodologies to address the inevitable limitations of each of the individual approaches. Specifically, we began by building on the results of our prior mixed-method study [3], which employed a systematic literature review [9] and expert interviews [10] to create a knowledge base of quality concerns for architects of ML-intensive systems. This knowledge base was our foundation, but it was subsequently extended with a series of interviews with architects of ML-intensive systems, aiming to catalog their quality concerns [10]. Over the course of nine months, we iteratively refined this catalog. To validate and enrich these findings, we organized a workshop where practitioners——architects of ML-intensive systems——offered critical feedback and additional insights into the quality concerns that we identified. We provide a public replication package to enable independent verification and replication of our methodology and study.[1] The research method we followed is summarized in Figure 1.

2.1. Step 1 - Initial Checklist

Our prior mixed-method study [3] identified 6 main categories of design challenges, namely Architecture, Data, Evolution, Quality Assurance (QA), Model, and Software Development Life Cycle (SDLC). The study also identified (i) 7 main categories of best practices, namely QA, Architecture, Model, Hardware (HW) & Platform, Evolution, SDLC, and Data, and (ii) 7 main categories of design challenges, namely Data, Architecture, Model, Evolution, HW & Platform, QA, and SDLP. Overall, we contributed 35 design challenges, 42 best practices, and 27 design decisions related to the architecting of machine learning systems.

In step 1, we derived 94 potential checks from the challenges, best practices, and design decisions presented in [3].

This step is represented as a circled "1" in Figure 1. Each check was assigned a unique ID for traceability. We then organized these checks into an initial checklist of three distinct lists, corresponding to challenges, best practices, and design decisions.

2.2. Step 2 - Interviews

As a parallel activity, we collaborated with data scientists to leverage their expertise in identifying key challenges and quality concerns associated with designing and constructing ML-enabled systems. The identified categories of concerns were: (i) system-level concerns that aim at influencing the system as a whole, (ii) component-level concerns that involve the ML component, or components, which have significant non-local effects on the system, (iii) system environment concerns that focus on how the system environment needs

[1] https://github.com/AlessioBucaioni/mlchecks

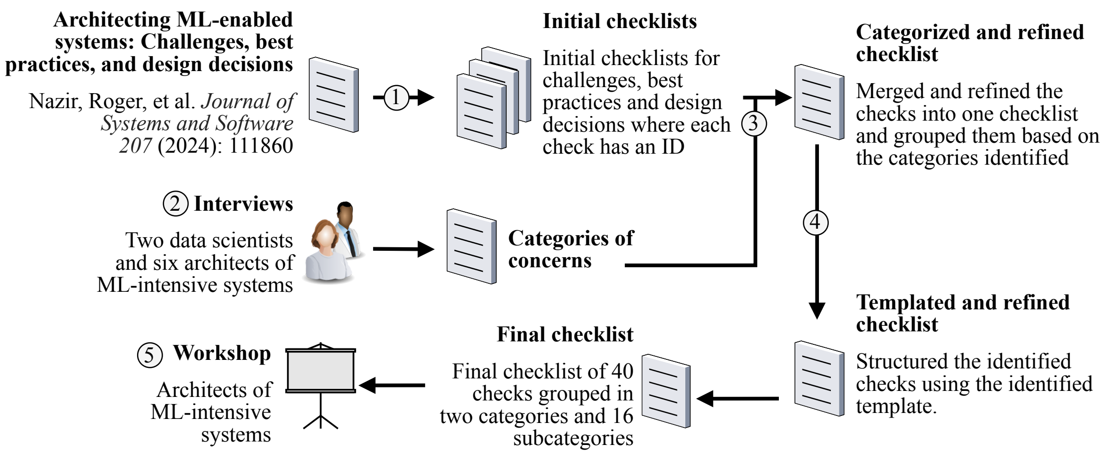
Architecting ML-enabled systems: Challenges, best practices, and design decisions
Nazir, Roger, et al. Journal of Systems and Software 207 (2024): 111860
1
Initial checklists
Initial checklists for challenges, best practices and design decisions where each check has an ID
3
Categorized and refined checklist
Merged and refined the checks into one checklist and grouped them based on the categories identified
2 Interviews
Two data scientists and six architects of ML-intensive systems
Categories of concerns
4
Templated and refined checklist
Structured the identified checks using the identified template.
Final checklist
Final checklist of 40 checks grouped in two categories and 16 subcategories
5 Workshop
Architects of ML-intensive systems

Figure 1: Overview of the adopted research method.

to be appropriately provisioned for, in particular, the ML function, (iv) ML-specific concerns that are important to overall system success, but where the solution to these concerns lives primarily outside the decisions made for the software architecture, and (v) non-architectural concerns that focus on fairness, privacy, or ethics.

To further refine these insights, we conducted semi-structured interviews at the Software Engineering Institute (SEI), involving six experienced system architects from organizations such as PNC Bank, FedEx, and Google (represented as a circled “2” in Figure 1). These were semi-structured interviews where each participant met with three SEI researchers. During the interviews, participants walked through and commented on each of the identified concerns, provided critical feedback, and contributed additional perspectives. The analysis of these interviews and discussions revealed five distinct categories of concerns based on their scope of influence: system environment, system-level, component-level, process, and architecture.

2.3. Step 3 - Categorizing and refining the checklist

In the next step, we revised the initial checklist by organizing the 94 identified checks into the five categories of concerns derived from the interviews. During this process, we eliminated redundant and generic checks (represented as a circled “3” in Figure 1). For example, the check “Does your system require real-time capabilities?” (stemming from a best practice) and “Do you have applicable techniques for ensuring privacy and confidentiality of your data?” (stemming from a design challenge) were removed due to their overly broad nature. This step was conducted iteratively, with the starting set of checks divided equally among the researchers. Each researcher categorized their assigned checks, followed by a collaborative validation and discussion of the categorizations to ensure consistency and accuracy. This refinement process resulted in a categorized and streamlined checklist comprising 60 checks.

2.4. Steps 4 - Templating and refining the checklist

Subsequently, we standardized the representation of all checks using an established template inspired by the work of Caldas et al. [11] (represented as a circled “4” in Figure 1). At the same time, we further refined the set of checks with the aim of minimizing less relevant ones and ensuring optimal categorization (represented as a circled “5” in Figure 1). Key refinements implemented during this stage included the removal of a few checks deemed less significant, the exclusion of concern categories that no longer had associated checks (namely component and system environment), and the reorganization of checks within the system-level category around system -ilities. Similar to the previous step, this process was conducted iteratively over nine months. The initial set of checks was divided equally among the researchers, with each researcher responsible for templating their assigned checks. This was followed by collaborative validation and discussions to ensure consistency and accuracy across all templated checks.

The result of these iterations was a finalized list of 40 checks, grouped into two main categories: System Concerns and Process Concerns. The former category focuses on the “ilities” of ML-intensive systems, such as reliability, scalability, security, and maintainability, while the latter addresses various aspects of the development process, such as documentation and team collaboration. We originally defined five categories based on prior literature and the initial workshop, but three of these categories remained unpopulated after the consolidation process. The absence of checks in three of the initial categories suggested that, in practice, those concerns were either too abstract or already implicitly covered by the other two categories. It is worth noting that during this step, two additional checks were introduced: one addressing Blue/Green and Canary testing, and another focused on Failure Mode and Effects Analysis (FMEA). All the checks are detailed in Section 3.

2.5. Step 5 - Workshop

In the final step, we conducted a workshop with 25 architects of ML-intensive systems to validate the checks and gather their feedback regarding the importance and the level of achievement of each concern (represented as a circled "6" in Figure 1). The workshop was organized as a half-day session as part of the ITARC conference[2], a prominent event for IT architects and professionals interested in IT architecture, organized by Dataföreningen Kompetens and IASA Sweden. The workshop brought together 25 registered participants with substantial expertise across a variety of sectors. On average, participants had 23.7 years of experience in architecting industrial systems, with experience ranging from 10 to 40 years. In contrast, their experience in building ML systems was more recent, averaging 1.85 years, with a range from 1 to 6 years. Participants represented a diverse set of sectors. Government was the most represented, accounting for 16.7% of the attendees. Banking & Finance and Automotive followed, each with 8.3% representation. A number of sectors—IT Infrastructure, Forensic & Legal, Telecom, Healthcare, Public Transport, Public Sector, Smart City, Engineering & Consulting, Medical, Mining, Aviation, and Consulting—were each represented by 4.2% of participants. The remaining participants, coming from various other backgrounds, made up the final 16.7%.

The session began with an introduction to the workshop's objectives, followed by the collection of basic participant information, such as their industry type and years of experience. Each of our identified checks was then presented to the attendees, accompanied by brief explanations and practical examples to provide context to them. Participants were then asked to evaluate the importance and level of achievement of each check using a 5-point Likert scale. Their ratings were collected using Mentimeter, an interactive app that allows presenters to create dynamic presentations with audience engagement features. Using Mentimeter, the participants were able to input their responses in real-time, via their smartphones or other internet-connected devices. Additionally, participants were invited to provide free-text remarks for each check, enabling the collection of qualitative feedback alongside the quantitative ratings. The workshop concluded with an open discussion, focusing on potential additional checks that might have been overlooked and participants' general thoughts on the checklist and the process. The results of the workshop are presented in Section 4.

2.6. *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 [9] and by Wohlin et al. [10]. Hereafter, we describe the main threats to validity according to the scheme by Wohlin et al. [10] and elaborate on mitigation strategies.

Internal Validity. One potential threat to internal validity arises from participant bias. The feedback collected during the workshop may have been influenced by the participants' individual backgrounds or familiarity with ML-intensive systems. For instance, participants with limited experience in ML systems may have underestimated the importance or feasibility of certain checks. To mitigate this, we ensured a diverse group of participants with varying roles and industries, ranging from IT architecture to solution architecture, and spanning sectors such as finance, healthcare, and public governance. Additionally, participants were provided with clear explanations and practical examples of each check to improve understanding and reduce misinterpretation. Another threat stems from the potential difficulty in understanding the context or scope of certain checks, as noted in the qualitative feedback. Some participants expressed challenges in interpreting specific questions, which could have affected the accuracy of their ratings. To address this, we revised the checklist template by detailing the *Context* field to provide more specific guidance and ensure the checks are easier to comprehend in future applications.

External Validity. The generalizability of the results presents a key challenge. While the workshop included participants from a diverse range of industries, it may not fully capture the needs of architects in less-represented sectors. Moreover, the sample size of 25 participants, though diverse, limits the extent to which the findings can be applied to a broader population. This was mitigated by having participants with significant expertise (ranging from 10 to over 40 years of experience) to ensure that the feedback reflected a high level of professional insight. Future iterations of the checklist could benefit from validation sessions with larger and more varied participant groups.

Construct Validity. The comprehensiveness of the checklist itself may pose a threat to construct validity. Although the checklist was derived from a systematic process, there is a possibility that certain niche concerns or emerging trends in ML-intensive systems were overlooked. This could limit the checklist's applicability across all contexts. To mitigate this, the checklist was iteratively refined through expert interviews, collaborative discussions, and participant feedback during the workshop to incorporate as many relevant perspectives as possible. Another construct validity issue relates to the evaluation metrics used. The 5-point Likert scale employed to measure importance and achievement, while standard, may not fully capture the nuanced perspectives of participants regarding the applicability or feasibility of specific checks. To address this, we complemented the quantitative ratings with qualitative feedback, allowing participants to provide additional context and suggestions for each check.

Conclusion Validity. The observed disparity between the importance and achievement ratings of some checks highlights a significant threat to conclusion validity. While this gap is central to RQ3, the reasons behind the

[2]https://itarc.nu/eng/

underachievement of certain checks were not fully explored during the study. This limits our ability to draw definitive conclusions about the root causes of these disparities. To partially address this, we incorporated participant remarks and discussion points into the analysis to provide some context for these gaps. Future studies should delve deeper into these underlying causes through follow-up interviews or focused case studies. The limited timeframe of the workshop also posed challenges. As a half-day session, the workshop may not have allowed participants sufficient time to engage deeply with each check or provide comprehensive feedback. While the session was designed to maximize efficiency, future iterations could include extended workshops or asynchronous feedback mechanisms to collect more detailed responses.

Reliability. The subjectivity of the qualitative feedback introduces another threat. While themes were identified and synthesized from participant remarks, individual differences in expression may have introduced variability in the analysis. This risk was mitigated by involving multiple researchers in the thematic analysis process to ensure consistency and reduce potential biases. Similarly, the iterative refinement of the checklist may have introduced researcher bias. Despite efforts to maintain consistency through collaborative discussions, there is always a possibility of divergence in interpretation. To address this, we implemented a structured process for refining the checklist, with each step documented and validated by multiple researchers to ensure transparency and reliability.

3. Checklist for Software Architects

This section presents the checklist that we created for our study. It begins with a description of the concern categories and a description of the template, followed by an overview of the checklist's composition together with few representative checks. The complete checklist is available on our website [3].

3.1. Concern Categories

We began with four categories of concerns, as identified during our initial interviews. From this starting point we refined the checks and applied a template, to ensure that they were consistently documented. This process of employing a documentation template led to the removal of two of the four concern categories, as these no longer had associated checks. The final concern categories were:

- *System concerns*, which address different quality attribute concerns. These concerns apply to the system as a whole, and do not require special considerations for ML components, as compared to other components.
- *Process concerns*, which pertain to the effective management and execution of the project in which the ML-enabled system is developed and maintained. They also address concerns specific to the process of architecting the ML-enabled system.

[3] https://alessiobucaioni.github.io/mlchecks/webpage/

3.2. Template for the Checks

To present the checks, we defined a simple template inspired by the work of Caldas et al. [11]. It consists of the following fields:

- *ID*: the unique identifier assigned to the check. We followed a systematic naming convention: the capital initial of the main category—S for System and P for Process—followed by the capital initial of the corresponding subcategory (e.g., A for Availability) and an integer. In cases where the combination of category and subcategory initials was not unique or potentially ambiguous, we used a two-letter abbreviation for the subcategory (e.g., SMD for System–Modularity).
- *Name*: a descriptive name summarizing the check.
- *Source*: the origin of the check, as identified in our prior work [3].
- *Context*: contextual information providing additional details about the check.
- *Check*: the specific question or statement that constitutes the actual check.
- *ML Specificity*: indicates the degree to which a check is specific to ML-intensive systems. It is categorized into three levels: low (L), medium (M), and high (H), based on the underlying rationale for the check.

3.3. Checklist

Figure 2 provides an overview of the checklist, organized into the two major concern groups described above: system and process. For clarity and brevity, Table 1 presents a simplified version of the checklist, omitting the source and context fields to enhance readability and save space.

3.3.1. System-Level Concerns

The system concerns category is divided into nine subgroups, each addressing a salient quality attribute. These concerns are: **usability**, **data quality**, **testability**, **modularity**, **monitorability**, **deployability**, **availability**, **reliability**, **safety and security**, and **privacy**.

Usability consists of two checks, primarily focusing on **visualization**. While system usability, as broadly construed, encompasses many aspects beyond visualization, these two checks emerged from the sources of information we utilized, namely scientific papers in the field and expert input.

Data quality is a critical aspect addressed by checks spanning various phases, including **data preparation**, **data**

Table 1: Checklist

System				
	Name	**ID**	**Check**	**ML Specificity**
Usability	Data visualization	SU1	Do you have data visualization techniques in place?	H
	Visualization techniques	SU2	Have you considered visualization techniques to identify or highlight relationships between data and computing tasks?	H
Data quality	Data preparation	SDQ1	Do you have strategies for preparing data and performing statistical analysis?	H
	Data cleaning	SDQ2	Is your dataset clean, of good quality, and free from potential bias?	H
		SDQ3	Are you concerned about data cleaning in NL processing?	H
	Dataset size	SDQ4	Do you have a well-sized dataset for training the ML component?	H
	Conceptual drift	SDQ5	Are you engineering your ML-based system so as to adapt to input data changes, also known as concept drift?	H
Testability	System correctness	ST1	Do you have proper techniques for ensuring system correctness?	M
	Model validation	ST2	Are you performing the validation of the model, e.g., to predict how a learning algorithm will behave on new data? Are you combining model validation with data validation to better detect corrupted training?	H
Modularity	Independent upgradability	SMD1	Are you building a component-based distributed system where parts may need to be upgraded?	L
	High Cohesion and low coupling	SMD2	Are high cohesion and low coupling important?	L
	Microservice	SMD3	If you are interested in maintainability and modifiability, did you consider using a microservice architecture?	L
	Discrete service	SMD4	Can you decompose your system into discrete services?	L
Monitorability	Modeling intrinsic uncertainty	SMN1	Can you explicitly model the intrinsic uncertainty of ML components and assess how it propagates and impacts other elements in the system at the designing stage?	H
	Time predictability	SMN2	Do you have proper mechanisms, like monitoring and a-posteriori analysability for time predictability?	M
	Monitoring drift	SMN3	Do you have tests that monitor changes in input distributions?	H
Deployability	Continuous integration	SDE1	Can you use continuous integration techniques for the development of your system?	L
	Infrastructure as code	SDE2	Do you have a method to manage the entire IT infrastructure, e.g. databases, servers, etc., that is needed to build your ML system?	M
	Blue/Green, Canary testing	SDE3	Are you including Blue / Green or canary testing in your standard MLOps pipelines?	Ö
Availability	Failure recovery strategy	SA1	Did you consider failure recovery strategies to avoid propagation of failures?	L
	Domain knowledge	SA2	Do you have the required level of domain knowledge to take availability decisions?	M
	Layered/tiered architecture	SA3	Can you cleanly split business logic from ML components? If so, did you consider using a layered/tiered architecture?	M
Reliability	Uncertainty	SR1	Do you have complete information on the uncertainty of the ML components used at design time?	H
Safety & Security	Fail safe	SS1	Do you have proper techniques for reaching safe states quickly when needed?	L
	Safety evaluation	SS2	Have you included an evaluation process for architectural safety design choices?	L
	Coding standards	SS3	Do you use strict and certified coding standards when developing safety-critical ML components?	L
	External certification	SS4	Are you having your system safety-certified by an external body?	L
	Design to defend	SS5	Are you explicitly designing and developing your ML system to defend vulnerable sections of the code that may be subject to cyber-attacks?	M
	Safety and fairness	SS6	Do you have a way of systematically ensuring safety and fairness in your system?	H
Privacy	Data loss and privacy	SP1	Have you considered using federated learning to improve data loss and privacy in your ML system?.	H

Process				
	Name	**ID**	**Check**	**ML Specificity**
	Documentation	PD1	Do you have proper documentation or a plan to document your ML system?	L
	Heterogeneous teams	PH1	Do you have heterogeneous teams mixing ML developers, data engineers, architects?	M
	Test-driven	PT1	Do you have a test-driven development strategy for your QA and, overall, is your testing process well defined?	L
	Separate pipelines	PSP1	Do you separate the branches for the training of the pipelines from the training of the model?	H
	Separate models	PSD1	If you have more than one model to develop, did you plan to develop them separately?	H
ML-aware process	Model customization and reuse	SML1	Do you have expertise to customize and reuse models?	H
	Managing and versioning	SML2	Do you have managing and versioning techniques in place?	L
	ML infrastructure for deployment	SML3	Have you defined an ML infrastructure and deployment processes?	L
	Model selection	SML4	Have you applied a systematic and documented approach for selecting the ML model used in your system?	H
	Model training	SML5	Have you evaluated the model's performance ?	H

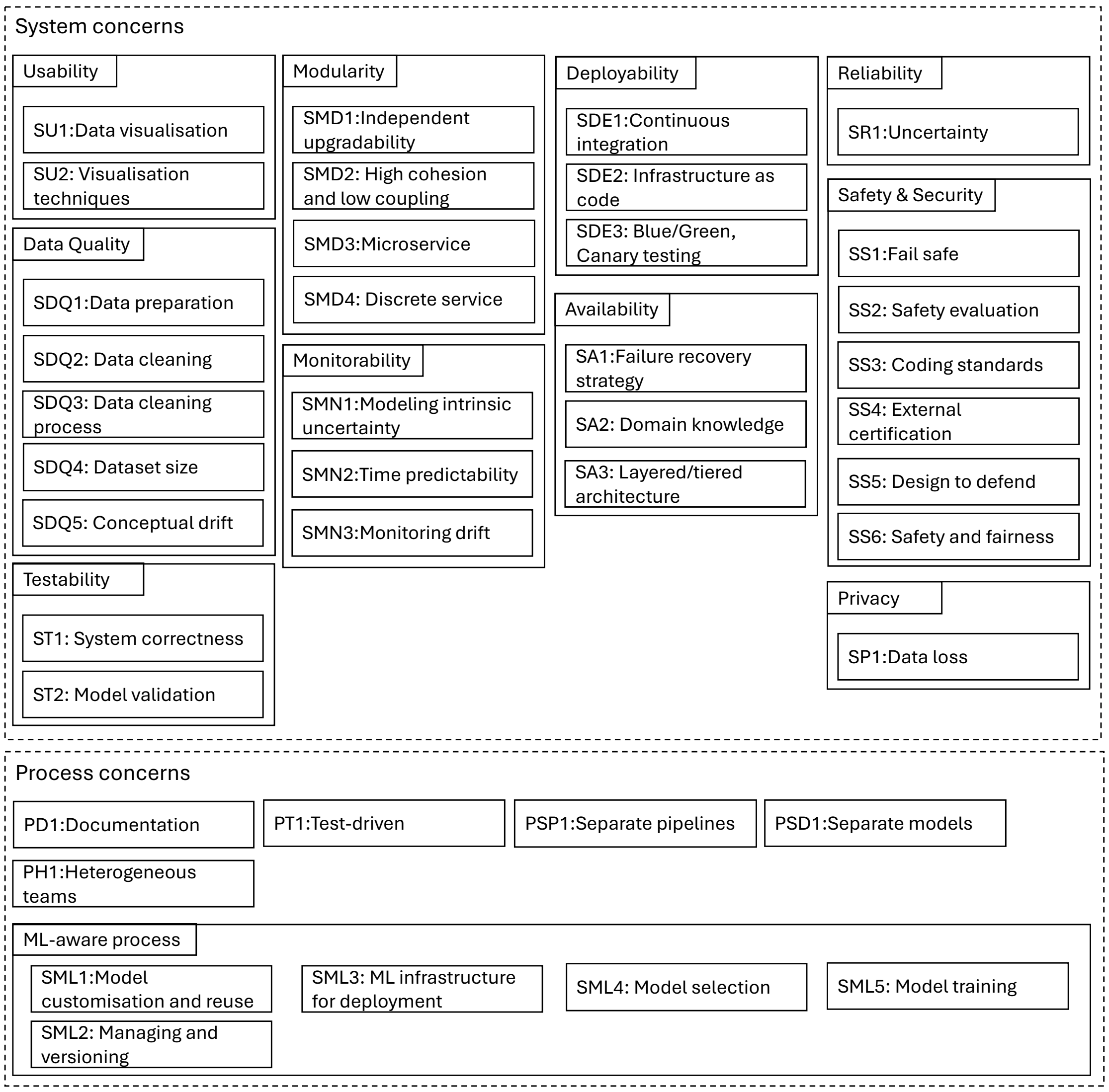

Figure 2: Overview of the checks.

cleaning, and **dataset size**. These checks also highlight the need to consider changes in input data over time, emphasizing that updates in ML-based systems, due to concept drift, can significantly impact system outcomes and performance.

Testability includes two checks, one of which addresses probabilistic **system correctness** by asking architects whether they have appropriate techniques to ensure system correctness. The other check—**model validation**—focuses on verifying both the model and the data used.

Modularity encompasses four checks, focused on how to decompose a complex system into smaller, more manageable parts. The first check concerns the **independent upgradability** of parts of a distributed ML system, and it recommends using a multi-node architecture approach as this allows distributed ML system components to be upgraded independently. The three other checks address **high cohesion and low coupling**. The first check recommends using a client-server architecture when high cohesion, low coupling, and security are important. When maintainability and modifiability are important, and the system can be decomposed into discrete services, the recommendation is to use a use microservice architecture, as recommended by the last two checks.

Monitorability includes three checks that focus on **modeling the intrinsic uncertainty of ML components** and assessing how this uncertainty propagates and impacts other system elements during the design stage. These checks also address incorporating mechanisms such as monitoring and a-posteriori analysability to ensure **time predictability**, and implementing tests to detect changes in the environment that make historical training data no longer representative of current reality, **drift**.

Deployability includes three checks. These address the use of **continuous integration** techniques in system development, the adoption of methods for managing the IT infrastructure needed to build ML systems (e.g., databases and servers) through approaches like **infrastructure as code**

or architecture as code [12], and the use of risk mitigation methods, like **Blue/Green or canary testing**, which may be highly effective in standard MLOps pipelines.

Availability focuses on two checks. The first one concerns the implementation of **failure recovery strategies** to prevent failure propagation. The second emphasizes architectural designs that isolate failures by **separating business logic from ML components**, enabling engineers to perform rollbacks more easily and effectively.

Reliability is represented by a single check, which probes the degree to which information on the **uncertainty** of ML components is available during the design phase.

Safety and Security are addressed by six checks. These ensure the existence of techniques to **reach safe states quickly** when needed, systematic methods for **ensuring safety and fairness** (e.g., FMEA), **evaluation processes for architectural safety design choices**, adherence to strict and certified **coding standards** for safety-critical ML components, an **external certification process** for critical systems, and explicit design and development **techniques to defend vulnerable sections**of ML system code against cyber-attacks.

Privacy focuses on minimizing **data loss and improving data privacy.** Federated learning is recommended as a method to address these concerns.

Example of Checks of the System Concerns. Here after, we provide a detailed example of a system check focusing on the quality attribute of usability.

- *ID*: SU2
- *Name*: Visualization techniques
- *Source*: Panousopoulou et al. [13]
- *Context*: In large ML-based systems, visualisation techniques help express the relationships between data and computing task.
- *Check*: Have you considered visualization techniques to identify or highlight relationships between data and computing tasks?

Note that our goal was not to exhaustively explore the quality concerns surrounding usability. Our goal was to reflect the quality concerns that are specific to ML-intensive systems, as supported by the literature and by our interviews. This check therefore emphasizes the importance of incorporating visualization techniques into the design process of large ML-based systems, particularly to express the relationships between data and computing tasks. Panousopoulou et al. [13] suggest that data visualization techniques enhance the design process by aiding in the representation of these relationships and facilitating data analytics. In large-scale systems, where the complexity of data flows and distributed computing can become challenging, visualization plays a critical role in improving usability and enabling better decision-making during system design. For instance, adopting data visualization techniques in the design process helps identify and communicate data dependencies and computational requirements effectively. These techniques are instrumental in facilitating data management and analytics, ensuring that critical aspects of the system are both accessible and interpretable to stakeholders.

3.3.2. Process Concerns

The process concerns are divided into six sub-groups, and the first five groups each consist of a single check.

The **Documentation** check is included to ensure that appropriate documentation is available or planned. This does not mean that everything must be documented. It focuses on careful consideration of where more thorough documentation might be required, as this can enhance efficiency, reusability, reproducibility, and shareability.

Heterogeneous teamsprobes whether teams include a mix of ML developers and architects. This combination best leverages the practical experience of ML developers alongside the expertise of architects.

Test-driven development strategy examines whether there is a test-driven development strategy in place, for quality assurance (QA) and for the standardization of the testing process.

Scalability checks whether there are **separate branches for pipeline training and model training**. In large-scale ML systems, this separation helps avoid the so-called "pipeline jungle".

Separate development of models attempts to ensure that models, when there are several, are developed separately. This approach simplifies system configuration, reduces the likelihood of costly errors, and helps identify unused or redundant models within the ML-based systems.

The final process concern is **ML-Aware Process**, which is subdivided into five checks. **Model customization and reuse** asks one to consider whether the expertise to customize and reuse models is available. **Managing and versioning** focuses on the presence of effective techniques for managing ML models, and, as they often rapidly evolve, versioning them. **ML infrastructure for deployment** queries the establishment of a proper ML infrastructure, along with suitable training and deployment processes. **Model selection** emphasizes careful selection of models based on the specific requirements of the domain (rather than, say, simply reusing models that a familiar). **Model training** addresses the importance of carefully managing the training process and optimizing model performance to enhance system outcomes.

Example of Checks of the Group process Concerns. Here, we provide an example of process check focusing on documentation.

- *ID*: PD1
- *Name*: Documentation
- *Source*: I10, Wan et al. [14], Anjos et al. [15]

- *Context*: A lack of documentation can affect the quality of ML systems and can negatively affect the reuse of processes. Contrariwise, proper documentation increases the efficiency, reusability, reproductivity, and shareability.
- *Check*: Do you have proper documentation or a plan to document your ML system?

This check focuses on the importance of proper documentation in critical areas of ML-based systems. A lack of appropriate documentation can hinder the use, reuse, and extension of these systems, negatively affecting their quality and long-term viability. Anjos et al. [15] and Wan et al. [14] emphasize that comprehensive documentation enhances efficiency, reusability, reproducibility, and shareability, and supports the overall design process. Proper documentation ensures that processes are well-understood and facilitates smoother collaboration and system evolution.

4. Validation

The validation of the checklist was conducted through a workshop with 25 active participants. The primary objectives of the workshop were: to validate the checklist categories and subcategories, to quantitatively assess the importance and achievement levels of each check, and to gather qualitative feedback to refine the checklist further. These goals directly addressed the overarching RQs of this study: ensuring the checklist's practical applicability, identifying gaps in current practices, and fostering a comprehensive understanding of quality concerns in ML-intensive systems.

The workshop was organized as a half-day session during the ITARC conference, a prominent event for IT architects and professionals in IT architecture, hosted by Dataföreningen Kompetens and IASA Sweden. It brought together 25 registered participants representing a diverse range of industries and roles, including software development, IT architecture, solution architecture, and enterprise architecture. Their expertise spanned sectors such as finance, healthcare, automotive, public governance, and technology, with professional experience ranging from 10 to over 40 years. Participants were asked to evaluate the importance and achievement levels of each check using a 5-point Likert scale. Their responses were collected in real-time through Mentimeter, an interactive app designed for dynamic presentations and audience engagement. This tool enabled participants to input their feedback seamlessly using smartphones or other internet-connected devices. In addition to quantitative ratings, participants were encouraged to provide free-text remarks for each check, offering qualitative insights into its relevance and clarity. The workshop concluded with an open discussion, focusing on potential additional checks that might have been overlooked, as well as general feedback from the participants on the checklist and on the validation process.

4.1. Quantitative feedback

During the workshop, participants evaluated each of the 40 checks using a 5-point Likert scale for both importance and achievement. As mentioned, the responses were collected in real-time using Mentimeter. We now summarize the results of their feedback, which are also graphically represented in Figure 3, showing a scatter plot of the weighted average importance and achievement scores for each check.

Importance. The weighted average score for importance was 3.49 and the standard deviation was 0.46. Scores ranged from a minimum of 2.58 to a maximum of 4.41. Notably, seven checks scored above the average plus the standard deviation, identifying critical design considerations for ML-intensive systems (RQ1) and highlighting areas where proper implementation can bridge gaps between importance and achievement (RQ3). Examples of these checks include: "Did you consider failure recovery strategies to avoid propagation of failures?" (SA1), which highlights the need for robust validation practices, and "Are high cohesion and low coupling important?" (SMD2), emphasizing fundamental principles for maintaining system modularity and maintainability. Conversely, nine checks scored below the average minus the standard deviation, signaling qualities that architects find less relevant in practice (RQ1). For instance, "Are you including Blue/Green or canary testing in your standard MLOps pipelines?" (SDE3) scored lower, suggesting these methods might not yet be widely adopted or deemed relevant in broad contexts. Similarly, "Are you having your system safety-certified by an external body?" (SS4) received lower ratings, possibly reflecting the niche nature of such certifications.

Achievement. The weighted average score for achievement was 2.72, with a standard deviation of 0.63, and scores ranged from a minimum of 1.6 to a maximum of 4. Overall, achievement scores were consistently lower than importance scores by an average factor of 0.77, indicating that many of the identified checks have yet to be widely implemented (RQ3), despite the attendees acknowledging that they are important. Notably, six scores exceeded the weighted average plus one standard deviation. Examples include: "Do you have proper techniques for ensuring system correctness?" (ST1) and "Are you performing the validation of the model, e.g., to predict how a learning algorithm will behave on new data?" (ST2) Conversely, six scores fell below the weighted average minus one standard deviation, highlighting critical areas that are difficult to achieve in practice for the architects. Examples of these checks include: "Have you defined an ML infrastructure and deployment processes?" (SML3) and "Do you separate the branches for the training of the pipelines from the training of the model?" (PSP1).

One notable observation was the identification of 15 checks where the difference between the weighted averages for importance and achievement was significant—exceeding the 0.77 factor. This highlights substantial gaps between the perceived importance and actual achievement of design qualities in ML-intensive systems (RQ3).

4.2. *Qualitative Feedback*

Participants provided valuable free-text remarks that offered deeper insights into the checklist's strengths and areas for improvement. For example, participants emphasized the need for clearer formulations and context for some checks, such as those related to data visualization and input monitoring. One participant noted, "Visualization is a good way to discover trends, but further data processing with other tools is required," highlighting the practical challenges faced. Another participant remarked on the lack of explicit checks for legal and ethical considerations, such as GDPR compliance. Key themes that emerged included:

- Clarity and Context: Several participants emphasized the need for clearer formulations and better contextualization of certain checks to ensure their scope and purpose were easily understood. To address this, we detailed the *Context* field in the check template to provide more specific guidance for each check.

- Practicality: Challenges related to data management, visualization, and resource allocation were frequently mentioned. Comments such as "Visualization is a good way to discover trends, but further data processing with other tools is required" reflected the need for actionable strategies. This theme highlights the need for actionable insights for researchers and practitioners to improve the practical application of the checklist in real-world settings.

- Applicability to ML: Some participants questioned whether specific checks were sufficiently tailored to ML systems or if they were general architectural concerns, underscoring the importance of emphasizing ML-specific contexts in future iterations.

- Ethical and Legal Considerations: The absence of explicit checks for ethical and legal concerns, such as GDPR compliance, was highlighted by a few participants. This theme provides a clear direction for both researchers and practitioners to incorporate these aspects in future iterations of the checklist.

The open discussion at the end of the workshop did not result in new checks or categories, but reinforced the importance of tailoring the checklist to practical, real-world challenges faced by practitioners. Moreover, there was a clear consensus among participants on the urgent need for such a checklist to be made available as soon as possible, reflecting its practicality, relevance, and potential impact in addressing current gaps in the design and development of ML-intensive systems.

4.3. *Impact on the Checklist*

The workshop feedback was instrumental in identifying which checks were perceived as more or less important and highlighting those that were underachieved despite their importance. These aspects are discussed in detail in Section 5. While the checklist itself was not modified, this validation process provided a deeper understanding of its applicability and areas requiring emphasis in future realizations. Moreover, we believe that the insights from the workshop should influence future iterations of the checklist by prioritizing checks that address critical gaps, particularly those areas where there is a disparity between importance and achievement. Additionally, this understanding will help in tailoring the checklist to meet specific industry needs, ensuring its broader adoption and relevance in diverse contexts.

5. Discussion

While software engineers have been designing and architecting systems for decades, it is only in recent years that the majority of engineers and architects have encountered the unique challenges posed by ML-enabled—and especially ML-intensive—systems. As with any emerging engineering challenge, practitioners are actively seeking guidance to navigate these complexities. This need for support was a recurring theme echoed throughout our workshop discussions.

In our preparatory work for the workshop, we uncovered a broad spectrum of concerns that an architect might need to address when designing ML-intensive systems. These concerns spanned system qualities, such as reliability, scalability, and security, as well as process issues, including documentation, team collaboration, and deployment strategies. Our research highlighted the diversity and complexity of challenges faced by architects, ranging from ensuring the robustness of ML models to managing the interplay between data science and traditional software engineering practices. What we learned in the workshop, however, was not just the breadth of these concerns but also the extent to which each one impacts practitioners who grapple with them on a daily basis. Participants consistently emphasized the practical difficulties they face, such as balancing the need for flexibility with the demand for system stability, managing concept drift in real-world deployments, and ensuring compliance with evolving ethical and legal standards. These insights provided invaluable context, helping us better understand which concerns require the most immediate attention and which might benefit from additional research or tooling.

Our first key finding from this study is that the importance of these concerns consistently outpaces their level of achievement. This disparity alone underscores the urgent need for a comprehensive checklist, alongside other forms of guidance such as training courses and best practice frameworks. A well-designed checklist serves as a concise and practical tool, encapsulating the collective expertise and distilled wisdom of the research and practitioner communities. Such tools have proven invaluable in domains where quality is critical, such as medicine and aviation, helping professionals mitigate risks, adhere to standards, and ensure consistency. By adapting this approach to the field of ML-intensive systems, we aim to address the pressing gaps between what architects recognize as important and what they are currently able to achieve.

Figure 3 presents a scatter plot illustrating the weighted average importance and achievement scores for each check. Based on this plot, the remainder of this section will discuss checks positioned above the average, those below the average, and checks exhibiting a substantial discrepancy between their importance and achievement scores.

5.1. Checks above the mean

The checks that are significantly above the mean are discussed below. We categorize them into two groups. The first group refers to checks whose *importance* was ranked above the average. These highlight critical areas of concern for ML-intensive system architects and include: SMD2, SDE1, SA1, SS1, SP1, SML1, PD1, SML2.

These checks focus on foundational principles of software architecture (e.g., cohesion, coupling, and reuse) and address key challenges unique to ML-intensive systems, such as managing privacy, ensuring failure recovery, and maintaining documentation. These areas are critical because they directly impact the maintainability, reliability, and scalability of ML-intensive systems, which are increasingly deployed in safety-critical and high-stakes domains.

The second group refers to checks whose *achievement* was ranked above the average. These represent areas where architects reported better success in implementing the practices and include: SDE1, SMD1, SMD2, SMD3, SU1, SU2.

The checks reflect established practices in software engineering, such as continuous integration, modular system design, and validation techniques. These areas have been widely adopted and adapted for ML-intensive systems due to the availability of supporting tools, frameworks, and methodologies. The relatively high achievement ratings suggest that architects are leveraging these well-understood practices to address challenges in ML-intensive systems effectively, even as they adapt to new requirements introduced by machine learning components.

5.2. Checks below the average

The checks ranked below the mean reflect challenges related to perceived relevance, technical complexity, resource constraints, and expertise gaps. We categorize them into two groups. The first group refers to checks whose importance was ranked below the average. These represent areas that architects perceive as less critical in the context of ML-intensive systems or checks that are less applicable. The group includes: SU1, SDQ4, SDQ5, SU2, SMN2, SMN3, SDE3, SS4, PSD1.

These checks often address advanced or specialized practices that may not yet be widely adopted or deemed essential by architects of ML-intensive systems. For example, practices such as monitoring input distributions or engineering for concept drift are relatively new challenges specific to ML-intensive systems. Similarly, techniques like Blue/Green testing or safety certification may only apply in certain industries or use cases, leading to their lower perceived relevance. These results suggest that, while these checks may be crucial in niche or high-stakes applications, they are not universally prioritized across all ML-intensive systems.

The second group refers to checks whose achievement was ranked below the average. These highlight practices that architects find difficult to implement, despite their admitted relevance to ML-intensive systems. The group includes: SML1, SML3, PSP1, SS4, SDE3.

These checks highlight areas that require significant resources, technical expertise, or organizational commitment to implement. For example, defining robust ML infrastructure, customizing models, and employing advanced deployment strategies like Blue/Green testing are practices that often involve steep learning curves, specialized tooling, and cross-disciplinary collaboration. These challenges can make implementation difficult, even when the practices are recognized as relevant. The findings point to gaps in the availability of tools, frameworks, and training needed to support architects in adopting these practices effectively. Note that there are some checks, such as safety-certification and Blue/Green testing that were not ranked particularly high by our participants, but where the achievement of these was ranked even lower. That is, although not all architects need these techniques, the ones who do need them are struggling.

5.3. Checks with substantial discrepancy between average importance and achievement

In addition, we observed a substantial number of checks with a large discrepancy between their importance and achievement (where we consider the difference to be “large” if it exceeded the average difference between importance and achievement). These checks highlight areas where architects recognize the significance of specific practices, but struggle to implement them effectively. These areas need to be highlighted, as they can serve as valuable inspiration for future research directions. The checks include: SDE3, SA1, SA2, SA3, SR1, SS1, SS2, SS3, SS4, SS5, SS6, SP1, PD1, PH1, PT1, PSP1, PSD1, SML1, SML2, SML3, SML4, SML5.

The significant gaps between the importance and achievement of these checks often point to challenges associated with education, awareness, technical complexity, resource constraints, and the evolving nature of ML-intensive systems. Many of these practices, such as federated learning, safety certifications, and advanced MLOps strategies like Blue/Green testing, require specialized knowledge, tools, and processes that may not yet be widely available or adopted. Additionally, some checks, such as maintaining

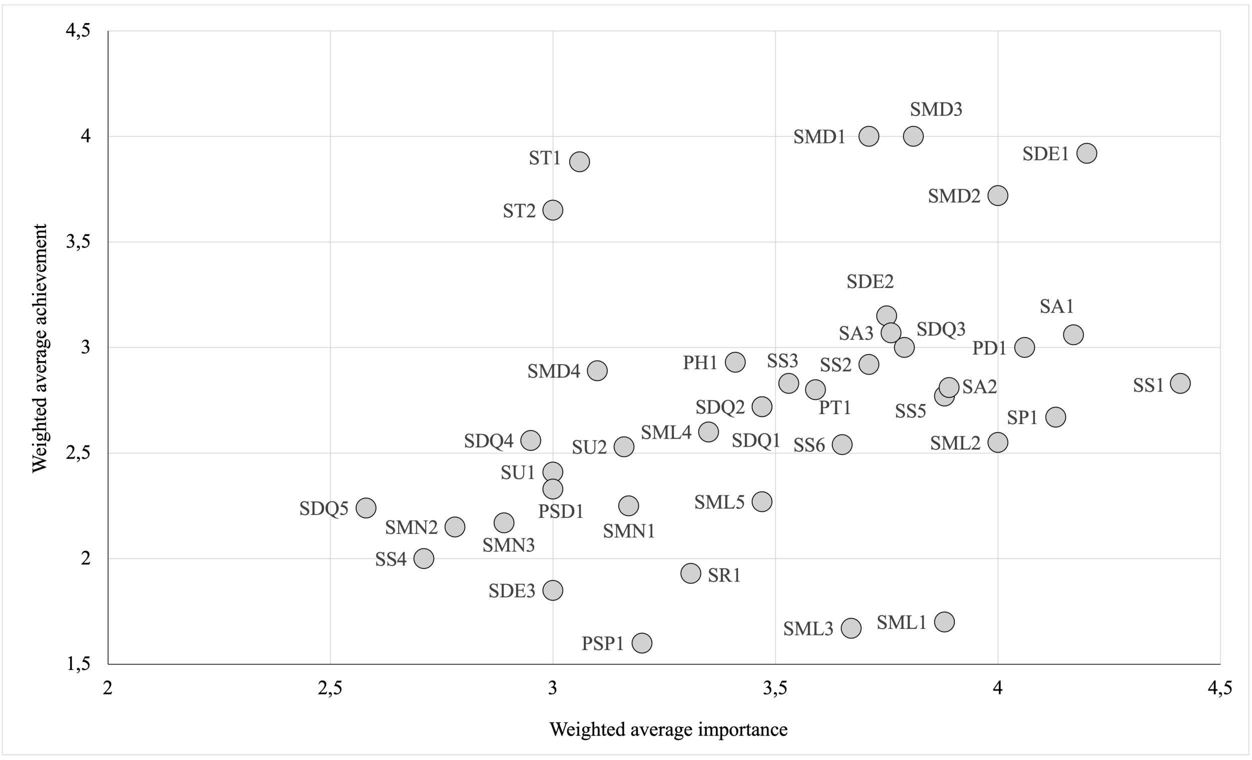

Figure 3: Scatter plot featuring the weighted average importance and achievement for each check.

heterogeneous teams or managing versioning techniques, highlight organizational and interdisciplinary challenges, reflecting the need for better collaboration between data science, engineering, and architecture teams. These gaps suggest that while architects understand the criticality of such practices, they may lack the necessary resources, tools, or expertise to implement them effectively. This presents a clear opportunity for future research and development: to focus on supporting and operationalizing these practices, creating better tools, frameworks, and training programs to bridge the gap between importance and achievement.

6. Related Work

There have been several attempts to create guidance for software architects over the past few decades. The IEEE 1471 standard, created in 2000, attempted to enumerate best practices for architectural descriptions, including rationale. It was superseded by ISO/IEC/IEEE 42010 [16]. A number of studies also attempted to catalog architectural design primitives [4], techniques for architecture review [5, 6], and guidance for architecture decision-making [7].

While we did not find any prior work that explicitly provides a checklist to guide software architects in designing ML-intensive systems, several related studies in the literature merit discussion.

First, the study in by Nazir et al. [3], which forms the foundation of our work, identified challenges, best practices, and design decisions for ML-intensive systems. As detailed in previous sections, we build on these findings and extend them by contributing a ready-to-use checklist specifically tailored for architects. Similarly, the study by Gorton et al. identifies fundamental engineering challenges in developing AI systems, with a focus on data collection, integration, inference, and continuous model updates and validation [17]. Our checklist aims to address and mitigate some of these challenges.

The empirical study reported by Zhang et al. explores architectural decisions in AI-based system development, providing valuable insights that align with the concerns addressed in our checklist [18]. Meanwhile, the work by Lu et al. adopts a different approach by proposing a pattern-oriented reference architecture for designing responsible AI systems, particularly those based on foundation models [19]. This work primarily focuses on responsible AI, offering a complementary perspective to our checklist-driven methodology.

Several works by Serban have addressed best practices in ML engineering and the architectural challenges associated with ML systems. In one study, the authors mined both academic and grey literature to identify 29 engineering best practices for ML applications [20]. Their findings indicated, for instance, that larger teams tend to adopt more practices, and that traditional software engineering practices generally exhibit lower adoption rates compared to ML-specific practices. Furthermore, the authors developed statistical models that could accurately predict perceived effects—such as agility, software quality, and traceability—based on the degree of adoption of specific practice sets. By combining adoption rates with the importance of practices (as determined through these models), they identified both underutilized yet important practices and widely adopted practices that have limited impact on the studied outcomes. In a subsequent study, Serban and colleagues conducted a mixed-methods empirical investigation into architectural challenges in ML systems [21]. This included: a systematic literature review to identify architectural challenges and their proposed solutions, semi-structured interviews with practitioners to qualitatively enrich the findings; and a survey to quantitatively validate them. The results revealed that traditional software architecture challenges continue to play a significant role when integrating ML components, while new ML-specific challenges also emerge. Interestingly, the study found that traditional decision drivers—such as scalability—remain dominant, whereas ML-heightened drivers, such as privacy, play a relatively marginal role in architectural decision-making.

Similar to Serban's work on ML engineering best practices, Nahar et al. conducted a meta-summary study to synthesize knowledge on practical challenges in ML engineering [22]. Using systematic literature review guidelines, they collected 50 relevant papers that, collectively, reported input from over 4,758 practitioners. From these sources, the authors extracted, grouped, and organized more than 500 distinct mentions of challenges. These challenges were then categorized into the following areas: Requirements Engineering, Architecture, Design, and Implementation—with special attention to Model Development and Data Engineering—Quality Assurance, and two crosscutting categories, Process Challenges and Team Challenges.

While Serban et al. and Nahar et al. focused on identifying ML engineering practices and architectural challenges through literature reviews and practitioner studies, our work complements theirs by operationalizing these insights into a structured checklist of quality concerns, prioritized and validated by experienced software architects, to support concrete decision-making in ML-intensive system design.

Finally, the study by Heyn et al. contributes a compositional approach to creating architecture frameworks, with a specific application to distributed AI systems [23]. While its focus is within the domain of software architecture, its ambition and scope differ significantly from our objective of creating a practical, actionable checklist for architects of ML-intensive systems.

7. Conclusion and Future Work

This study provides an empirically grounded investigation into the quality concerns faced by real-world architects of ML-intensive systems. By employing a multi-method research approach, we identified, refined, and validated a comprehensive checklist of design considerations that address both system and process concerns. The study revealed critical gaps between the importance and achievement of these design qualities, emphasizing the need for targeted efforts to enhance the practices surrounding the development and deployment of ML-enabled systems.

The key findings of this research highlight areas of high importance, such as ensuring system correctness, improving documentation, and adopting modern architectural practices like microservices, which architects may perceive as important, and yet they frequently remain underachieved. Conversely, certain practices, such as safety certification and advanced deployment strategies like Blue/Green testing, were deemed less critical, or perhaps were difficult to implement given current practices. These insights, aligned with the research questions, underscore the challenges architects face and offer actionable directions for both practitioners and researchers.

While this study has advanced the understanding of architectural concerns in ML-intensive systems, there remain opportunities for further refinement and application of the checklist. Future efforts should focus on expanding validation across diverse industries and audiences, enhancing tool support to facilitate practical implementation, and addressing gaps related to ethical and legal considerations.

As discussed in the validation section, Section 4, an interesting research direction identified by the practitioners, concerns ethical and legal considerations, such as GDPR or AI Act [24] compliance. Additionally, adapting the checklist for domain-specific needs and integrating emerging trends in ML technologies will further ensure its relevance and impact. Longitudinal studies evaluating its effectiveness over time could provide deeper insights into its practical benefits and inform ongoing improvements.

Finally, the checklist that we proposed ought to be treated as a living artifact, and hence should be periodically updated. Thus the entire community of researchers and practitioners who are focused on the design of ML-intensive systems can contribute by refining existing checks, proposing new ones, and further validating the list.

Acknowledgements

This work has been partially funded by (a) the National Science Foundation under Grant No. 2232721 (b) the MUR (Italy) Department of Excellence 2023 - 2027, (c) the European HORIZON-KDT-JU research project MATISSE "Model-based engineering of Digital Twins for early verification and validation of Industrial Systems", HORIZON-KDT-JU-2023-2-RIA, Proposal number: 101140216-2, KDT232RIA_00017, (d) the PRIN project P2022RSW5W - RoboChor: Robot Choreography, (e) the PRIN project 2022JKA4SL - HALO: etHical-aware AdjustabLe autOnomous systems, and (f) the Swedish Agency for Innovation Systems through the project "iSecure: Developing Predictable and Secure IoT for Autonomous Systems" (2023-01899).

References

[1] P. Rohit, T. Saleh, M. Szybowski, R. Whiteman Forbes, Operationalizing machine learning in processes, https://www.mckinsey.com/capabilities/operations/our-insights/operationalizing-machine-learning-in-processes, accessed: December 11, 2024 (2021).

[2] U. F. Forbes, Why most machine learning applications fail to deploy, https://www.forbes.com/councils/forbestechcouncil/2023/04/10/why-most-machine-learning-applications-fail-to-deploy/, accessed: December 11, 2024 (2023).

[3] R. Nazir, A. Bucaioni, P. Pelliccione, Architecting ml-enabled systems: Challenges, best practices, and design decisions, Journal of Systems and Software 207 (2024) 111860.

[4] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison-Wesley Professional, 2022.

[5] J. Maranzano, S. Rozsypal, G. Zimmerman, G. Warnken, P. Wirth, D. Weiss, Architecture reviews: practice and experience, IEEE Software 22 (2) (2005) 34–43.

[6] R. Kazman, L. Bass, Making architecture reviews work in the real world, IEEE Software 19 (1) (2002) 67–73. doi:10.1109/52.976943.

[7] A. Tang, R. Kazman, Decision-making principles for better software design decisions, IEEE Software 38 (6) (2021) 98–102. doi:10.1109/MS.2021.3102358.

[8] H. Cervantes, R. Kazman, Designing Software Architectures: A Practical Approach, Addison-Wesley, 2024.

[9] B. Kitchenham, P. Brereton, A systematic review of systematic review process research in software engineering, Information and software technology (2013).

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

[11] R. Caldas, J. A. P. García, M. Schiopu, P. Pelliccione, G. Rodrigues, T. Berger, Runtime verification and field-based testing for ros-based robotic systems, IEEE Transactions on Software Engineering (2024).

[12] A. Bucaioni, A. D. Salle, L. Iovino, P. Pelliccione, F. Raimondi, Architecture as code, in: 22nd International Conference on Software Architecture (ICSA), 2025.

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

[14] Z. Wan, X. Xia, D. Lo, G. C. Murphy, How does machine learning change software development practices?, IEEE Transactions on Software Engineering 47 (9) (2019) 1857–1871.

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

[16] ISO/IEC/IEEE 42010:2022: Software, systems and enterprise — Architecture description, https://www.iso.org/standard/74393.html (2022).

[17] I. Gorton, F. Khomh, V. Lenarduzzi, C. Menghi, D. Roman, Software Architectures for AI Systems: State of Practice and Challenges, Springer Nature Switzerland, Cham, 2023, pp. 25–39. doi:10.1007/978-3-031-36847-9_2.
URL https://doi.org/10.1007/978-3-031-36847-9_2

[18] B. Zhang, T. Liu, P. Liang, C. Wang, M. Shahin, J. Yu, Architecture decisions in ai-based systems development: An empirical study, in: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2023, pp. 616–626. doi:10.1109/SANER56733.2023.00063.

[19] Q. Lu, L. Zhu, X. Xu, Z. Xing, J. Whittle, Toward responsible ai in the era of generative ai: A reference architecture for designing foundation model-based systems, IEEE Software 41 (6) (2024) 91–100. doi:10.1109/MS.2024.3406333.

[20] A. Serban, K. Van der Blom, H. Hoos, 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, pp. 1–12.

[21] A. Serban, J. Visser, Adapting software architectures to machine learning challenges, in: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2022, pp. 152–163.

[22] N. Nahar, H. Zhang, G. Lewis, S. Zhou, C. Kästner, A meta-summary of challenges in building products with ml components–collecting experiences from 4758+ practitioners, in: 2023 IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN), IEEE, 2023, pp. 171–183.

[23] H.-M. Heyn, E. Knauss, P. Pelliccione, A compositional approach to creating architecture frameworks with an application to distributed ai systems, Journal of Systems and Software 198 (2023) 111604. doi:https://doi.org/10.1016/j.jss.2022.111604.
URL https://www.sciencedirect.com/science/article/pii/S0164121222002801

[24] Madiega Tambiama André, Artificial intelligence act (2024).
URL https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf