The Journal of Systems and Software 216 (2024) 112151

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Technical debt in AI-enabled systems: On the prevalence, severity, impact, and management strategies for code and architecture ☆

Gilberto Recupito [a,*], Fabiano Pecorelli [a], Gemma Catolino [a], Valentina Lenarduzzi [b], Davide Taibi [b], Dario Di Nucci [a], Fabio Palomba [a]

[a] Software Engineering (SeSa) Lab — University of Salerno, Salerno, Italy
[b] University of Oulu, Oulu, Finland

ARTICLE INFO

Keywords:
AI technical debt
Software quality
Survey studies
Software engineering for artificial intelligence
Empirical software engineering

ABSTRACT

Context: Artificial Intelligence (AI) is pervasive in several application domains and promises to be even more diffused in the next decades. Developing high-quality AI-enabled systems — software systems embedding one or multiple AI components, algorithms, and models — could introduce critical challenges for mitigating specific risks related to the systems' quality. Such development alone is insufficient to fully address socio-technical consequences and the need for rapid adaptation to evolutionary changes. Recent work proposed the concept of AI technical debt, a potential liability concerned with developing AI-enabled systems whose impact can affect the overall systems' quality. While the problem of AI technical debt is rapidly gaining the attention of the software engineering research community, scientific knowledge that contributes to understanding and managing the matter is still limited.

Objective: In this paper, we leverage the expertise of practitioners to offer useful insights to the research community, aiming to enhance researchers' awareness about the detection and mitigation of AI technical debt. Our ultimate goal is to empower practitioners by providing them with tools and methods. Additionally, our study sheds light on novel aspects that practitioners might not be fully acquainted with, contributing to a deeper understanding of the subject.

Method: We develop a survey study featuring 53 AI practitioners, in which we collect information on the practical prevalence, severity, and impact of AI technical debt issues affecting the code and the architecture other than the strategies applied by practitioners to identify and mitigate them.

Results: The key findings of the study reveal the multiple impacts that AI technical debt issues may have on the quality of AI-enabled systems (*e.g.*, the high negative impact that *Undeclared consumers* has on security, whereas *Jumbled Model Architecture* can induce the code to be hard to maintain) and the little support practitioners have to deal with them, limited to apply manual effort for identification and refactoring.

Conclusion: We conclude the article by distilling lessons learned and actionable insights for researchers.

1. Introduction

Contemporary software systems are more and more empowered by Artificial Intelligence (AI) modules, which companies and individuals use to make informed decisions (Zhou and Chen, 2018) or automate demanding tasks requiring human workload (Rech and Althoff, 2004). The impact of AI on industry and society is pervasive, with a further increase expected in the coming years in multiple domains (Ravì et al., 2016; Zhang et al., 2019). In such a context, researchers, high-tech companies, and government agencies are actively engaging with the definition of novel quality assurance practices that may cope with the continuous evolution of AI algorithms and models (Martínez-Fernández et al., 2022). For instance, the *International Organization for Standardization* (ISO) has recently introduced a new quality standard, i.e., the ISO/IEC 25059:2023 (ISO/IEC, 2023a), that incorporates AI-specific characteristics, *e.g.*, functional correctness of the model. These recent advances are designed to provide practitioners with mechanisms to control for the evolution of AI-specific quality properties (Mikkonen et al., 2021). However, they do not cover the *integration* of AI modules within more complex software systems, representing a key limitation. Indeed, as recently pointed out by Sculley et al. (2015), machine

☆ Editor: Heiko Koziolek.

* Corresponding author.
E-mail addresses: grecupito@unisa.it (G. Recupito), fpecorelli@unisa.it (F. Pecorelli), gcatolino@unisa.it (G. Catolino), valentina.lenarduzzi@oulu.fi (V. Lenarduzzi), davide.taibi@tuni.fi (D. Taibi), ddinucci@unisa.it (D. Di Nucci), fpalomba@unisa.it (F. Palomba).

https://doi.org/10.1016/j.jss.2024.112151
Received 20 September 2023; Received in revised form 10 May 2024; Accepted 25 June 2024
Available online 4 July 2024

learning modules only represent a small fraction of the code contained within machine learning-enabled systems, and their integration is often challenging and a source of quality concerns that may affect multiple maintenance and evolution properties, other than the overall effectiveness of these systems.

AI technical debt[1] (AITD) represents a metaphor to indicate the typical quality concerns of suboptimal solutions integrated into the building process of AI-enabled systems. An example of AITD is the so-called *Pipeline Jungle*, which arises when an AI pipeline becomes overly complex and challenging to maintain over the evolution of the AI-enabled system because of the presence of many interconnected data processing and machine learning components.

In their seminal paper, Sculley et al. (2015) argued that issues due to technical debt are highly diffused in practice and proposed an initial catalog of more than 20 types of technical debt affecting ML-enabled systems under different angles, *e.g.*, issues causing code and architectural concerns. Later, Bogner et al. (2021) extended such the initial catalog by conducting a systematic mapping study, through which 72 additional anti-patterns affecting multiple stages of an AI pipeline were identified. Most research efforts have been conducted to elicit and characterize the properties of AITD (Alahdab and Çalıklı, 2019; Jebnoun et al., 2020; Lwakatare et al., 2019; Munappy et al., 2019; Matthews, 2020), while only a few attempts have been made to identify it (Foidl and Felderer, 2019; Liu et al., 2020; Breck et al., 2017). As such, we argue that the scientific knowledge of AITD is still limited. More specifically, our research points out that key aspects like the prevalence of the problem in practice, the severity and impact of AITD, and the mitigation strategies to deal with it are still unexplored.

> **Objective of the Study**
>
> *In this paper, we aim to enlarge the knowledge of AI technical debt issues that affect the code and architecture, investigating their criticality and how practitioners mitigate those concerns in practice.*

On the one hand, our focus on these AITD issues is driven by our willingness to investigate quality concerns that may lead to ripple effects on the quality of an entire AI-enabled system (Shivashankar and Martini, 2022; Lenarduzzi et al., 2021b): by investigating the debt items defined in the literature, we could therefore provide further insights on actually harmful concerns, assessing how critical they are in practice and what strategies to put in place to limit their adverse effects. On the other hand, no previous work targeted the analysis of those types of AITD (Shivashankar and Martini, 2022): as we intend to extend the current body of knowledge, we deemed these quality concerns as an ideal starting point for our research.

We address the main objective of the study by designing and executing a survey study based on an online questionnaire involving real-world practitioners experienced in developing and maintaining AI-enabled systems. We focus our analysis on 9 AITD issues defined by Sculley et al. (2015), Gesi et al. (2022), and Bogner et al. (2021). We opt for such a research method in light of analyzing the current state of the art. While the software engineering research community is actively engaging with the problem of AITD, scientific knowledge on the matter is still limited. On the contrary, we notice that practitioners seem to perceive the problem of AITD as critical —as proof of that, practitioners released the preliminary catalog of these types of quality issues in ML-enabled systems (Sculley et al., 2015), and other prominent related resources(*e.g.*, (Matthews, 2020; Breck et al., 2017)) have been released by practitioners. As such, our work aims to survey the state of the practice to inform about the next research avenues to pursue to support practitioners better when dealing with AITD issues. Our survey study involves 53 participants who were consulted about the frequency, severity, impact, and mitigation strategies for nine distinct AITD concerns. The key findings of our study revealed that practitioners are relatively unaware of the prevalence of these AITD concerns, yet they consider them severe and harmful.

The key findings of our study revealed that practitioners, while recognizing the low prevalence of AITD issues, consistently recognize the severity and potential harm these issues can inflict on the systems' quality. Furthermore, the study highlights that the primary response to these concerns involves manual inspections and ad-hoc refactoring strategies. The perception of participants on the prevalence and severity of issues suggests a potential gap in the practitioners' ability to identify these issues automatically. This gap likely contributes to underestimating their prevalence, underscoring the need for automatic approaches in detecting and addressing AITD issues. Based on our findings, we eventually come up with a set of challenges that the research community is called to address, other than additional implications that may impact future research actions on AITD.

Structure of the paper. Section 2 discusses the literature on AITD. In Section 3, we describe the research questions driving our investigation and the methodical choices to address them. The analysis of the results is reported in Section 4, while further discussions and implications are discussed in Section 5. Section 6 reports on the limitations of our work and the actions conducted to mitigate them. Finally, Section 7 concludes the paper and defines the steps of our future research agenda.

2. Background and related work

This section first defines the main terms employed throughout the article. Afterward, it provides an overview of technical debt and the software quality concerns affecting AI-enabled systems. In addition, we discuss the most closely related work, pointing out the main differences that let our work advance the state of the art.

2.1. Terminology

Before reporting on the background and the related literature, we provide the basic definitions used throughout the empirical study:

Technical debt. Technical debt is a metaphorical concept introduced by Ward Cunningham in 1992 (Cunningham, 1992). It refers to the compromises made during the design, implementation, or maintenance of software systems to meet immediate goals, often at the expense of the overall system's quality. Technical debt encompasses various factors, including suboptimal coding practices, hurried decision-making, lack of documentation, and architectural shortcuts (Holvitie et al., 2018). While sometimes incurred unavoidably due to factors like tight deadlines or evolving requirements, accumulating technical debt can hinder future development efforts, increase maintenance costs, and impede the evolution of the software.

AI-enabled systems. An AI-enabled system represents "a software system with functionalities enabled by at least one AI component, e.g., for image-, speech-recognition, and autonomous driving" (Martínez-Fernández et al., 2022). In the context of this study, the term refers to systems integrating AI technologies to meet diverse business and technological requirements.

AI components. An AI component is a module or a unit within an AI-enabled system "whose behavior relies on AI code or an AI library that provides an implementation of AI algorithms" (Martínez-Fernández et al., 2022). Within this study, recognizing the definition of AI components is crucial for understanding the interactions of participants engaged in developing, implementing, or maintaining these integral modules.

[1] In this context, we refer to all technical debt issues occurring in AI-enabled systems as AITD.

AI practitioners. As AI practitioners, we intend all practitioners involved in building AI-enabled systems, from data management activities to deploying and serving the system into production. The activities involved in building AI-enabled systems include additional roles that oversee the development and management of the software system, including Data Scientists and Software Engineers (Ozkaya, 2020). This definition allows us to collect different perspectives and priorities on the quality of process and the product of AI-enabled systems.

2.2. Software quality for AI

Bosch et al. (2021) argued that AI engineering expands upon software engineering, incorporating novel technologies and processes essential for developing such systems. Foidl and Felderer (Foidl and Felderer, 2019) defined an ML-based software system as one that employs algorithms to process data, utilizing ML models to autonomously make intelligent decisions based on extracted relationships, patterns, and data-derived knowledge. Based on the terminology defined by Martínez-Fernández et al. (2022), an AI architecture can have traditional and AI software components. Sculley et al. (2015) defined the typical structure of a ML-enabled system, where a tiny fraction of components aimed at the learning and inference process is surrounded by a series of traditional software components aimed at supporting the functionalities of the ML component. Siebert et al. (2020) specified AI software components, particularly those involving ML, as data-driven software entities. A data-driven software component pertains to a software module that addresses a defined task (*e.g.*, image segmentation, sentiment analysis, classification) through data science methodologies, encompassing ML, data mining, natural language processing, signal processing, and statistics. AI components, especially based on supervised ML or DL, differ fundamentally from traditional components because they are data-driven, *i.e.*, their behavior is non-deterministic, statistics-orientated, and evolves in response to the frequent provision of new data (Amershi et al., 2019).

However, due to the complexity of AI components, the effect of applying bad practices on these components is critical and not manageable like for traditional components. One of the main complications of operating with AI components is related to the maintainability of the AI-enabled system. This type of component provides non-deterministic results in which the testability is not assessable with traditional testing approaches (Mikkonen et al., 2021). Moreover, the lack of requirements concerning the specification and robustness of AI models significantly impacts the quality of models designed for conventional systems (Kuwajima et al., 2020). Gezici and Tarhan (2022) conducted a systematic literature review on the software quality attributes used for AI systems. The main findings highlight the adaptation of practitioners of the quality model ISO/IEC 25010 to AI. Since this type of system has a different nature and applying established quality standards could not be a solution for the specific case in which AI is used, the findings highlight the lack of well-defined guidelines, frameworks, roadmaps, and models for measuring the quality of AI-enabled software.

Inadequate process definitions to be followed, quality metrics, attributes, and assurance techniques cause new challenges for academia and industry. A systematic literature review by Martínez-Fernández et al. (2022) concerning the main challenges in the field of SE4AI discovered that besides established quality standards, there are specific quality attributes for machine learning and artificial intelligence-related to safety and ethics. The specificity of these quality aspects highlights the need to define a new quality model that inherits the quality terms from software quality standards and adapts it to AI. The European High-Level Expert Group on AI presented the Assessment List for Trustworthy AI (ALTAI) (Ala-Pietilä et al., 2020). It addresses ethical considerations and data privacy issues while highlighting the significance of reliability and quality. By introducing a series of self-assessment queries, the report facilitates practitioners in identifying prevalent quality issues often encountered in AI-enabled systems. Additionally, starting from the quality standard ISO/IEC 25010 (ISO/IEC, 2023b) for software systems, the International Organization for Standardization (ISO) developed a new quality standard specific for AI systems: ISO/IEC 25059 (ISO/IEC, 2023a). The updated quality standard for AI systems emphasizes comprehensive product quality evaluation, including functional adaptability and correctness. These quality aspects are particularly relevant because models may not always guarantee correctness in all situations, and the ability to utilize past data or outcomes for future predictions is a critical point that can lead to the quality degradation of the model.

However, while quality aspects related to AI-enabled systems have been developed in the last few years, there is still the need to focus the research on several issues that could damage the overall quality of AI-enabled systems. In this regard, a few studies have delved into the potential effects of introducing quality problems. Thung et al. (2012) performed an empirical study to assess the effect of bugs on machine learning systems. Analyzing three open-source machine learning systems, they found that most bugs affect the algorithm used by the system. Sun et al. (2017) discovered seven highly diffused bug types by analyzing three machine learning projects. Golendukhina et al. (2022) interviewed ten companies to extract quality issues in AI-enabled systems. They extracted 12 issues frequently in the development process, identifying how practitioners detect them. These pioneering studies call for intensified research into quality concerns within the AI domain. By highlighting the real-world consequences of quality issues, they emphasize the need for robust quality assurance measures to ensure the effectiveness and dependability of AI-enabled systems.

2.3. Technical debt

Technical debt is the metaphor introduced by Cunningham (1992) to describe suboptimal solutions to gain benefits in the short term that should be repaid with more costs over the medium or long term. As for the metaphor of financial debt, technical debt can cause severe problems to the system's maintainability, increasing maintenance costs and decreasing the quality (Seaman and Guo, 2011; Lenarduzzi et al., 2021a). From the general definition, several studies have been conducted to explore the occurrence of technical debt in depth. Tom et al. (2013) explored the several types of granularity that highlight the level of the effect of each technical debt. In detail, the effect is related to the code (*code debt*), the design, and the architecture (*architectural debt*), in the environment of the application (*environmental debt*), from the relation of the community to the infrastructure (*knowledge distribution debt*) and the activities related to the testing of the system (*testing debt*). Subsequently, Li et al. (2015) extended the classification to other technical debt types (*e.g., requirements debt and infrastructure debt*).

Finally, Alves et al. (2016) conducted a systematic mapping study to overlay a broader overview of technical debt. Among all the outcomes of exploring technical debt management activities found in the study, they identified 15 types of technical debt, encompassing other aspects such as versioning and usability. As an additional contribution, they highlighted the differences between the taxonomies in this field, giving complementary and overlapping definitions. Nonetheless, boundaries among these definitions are difficult to draw. Among the quality issues that can cause *technical debt*, the concept of code smells is prominent, as defined by Fowler (1999). Code smells refer to specific issues within the code that significantly impact the maintainability of a software system. These issues are considered "*symptoms of poor design and implementation choices*", and if left unaddressed, they can deteriorate the overall quality of the system. Consequently, it is crucial to protect the quality of software systems by actively detecting and resolving code smells. To this end, researchers have conducted numerous studies to identify techniques for detecting code smells. Initially, structural metrics and historical metrics were explored, resulting in the proposal of various tools and methodologies (Marinescu, 2004; Moha et al.,

2010; Tufano et al., 2015; Palomba et al., 2015, 2016). However, with AI's advancement and general adoption, machine-learning techniques are proposed to identify code smells (Maiga et al., 2012; Pecorelli et al., 2019). The impact of code smells extends beyond just maintainability. Studies have shown that code smells can directly affect other key aspects of software quality, such as performance (Chen et al., 2014), change-proneness (Khomh et al., 2012; Olbrich et al., 2009), and reliability (Jaafar et al., 2013). Moreover, the occurrence of code smells can affect the development process and the productivity of the development team (Tom et al., 2013), making it imperative to address these issues proactively.

Technical debt has been a significant area of research in software engineering. With the rise of AI systems, researchers have also started exploring technical debt in the context of these systems. One notable study in this direction is the work by Sculley et al. (2015). This study analyzes technical debt in ML systems and highlights the challenges posed by such debt. ML systems have a remarkable capacity for incurring technical debt because they have all the maintenance problems of traditional code plus an additional set of ML-specific issues. They extracted a series of issues that cause technical debt in the several components included in an ML pipeline from the model (*e.g.*, Epsilon Features) to the system's architecture (*e.g.*, Pipeline Jungle). However, while these issues have been identified as challenges for research to find possible solutions to manage this type of technical debt, there is currently limited practical advancement that supports practitioners in effectively identifying, evaluating, and managing these specific issues in real-world AI projects. As AI grows, addressing technical debt in these systems becomes increasingly crucial for maintaining the system's overall quality. Further research and tooling advancements are needed to enable practitioners to proactively handle technical debt in ML systems and ensure their long-term success.

2.4. Related work

Several works analyzed the presence of technical debt in AI-enabled systems. Liu et al. (2020) analyzed comments in the code of seven popular deep-learning projects to investigate the presence of self-admitted technical debt. Their study revealed many technical debt issues reported by practitioners, particularly in design debt, requirement debt, and algorithm debt. Since this work emphasizes the significance of acknowledging and addressing self-admitted technical debt to enhance the maintainability of ML projects, our work extends the investigation to include AITD issues, analyzing their prevalence and collecting data from the perspective of practitioners.

Jebnoun et al. (2020) conducted a comparative analysis of code smell prevalence of deep learning projects. Surprisingly, they found that the prevalence of code smells in projects containing deep learning components was not significantly different from systems not containing them. Furthermore, they observed an increasing trend of code smells during the evolution of deep learning applications, emphasizing the importance of addressing technical debt proactively. The trend discovered in this study is an important finding that highlights that deep learning projects are also affected by technical debt. Our study's goal is to extend the findings to understand whether practitioners also perceive AITD issues in these types of projects.

Additionally, several studies have been conducted to evaluate the data quality inside AI-enabled systems. Foidl et al. (2022) contributed to the literature with a catalog of 36 data smells related to the data format used in AI-enabled systems. From the definition of the smells proposed, Foidl and Felderer (2019) proposed a risk-based data validation approach to assess data quality within an AI pipeline. This approach helps identify and mitigate data-related technical debt by considering data smells, source quality, and pipeline quality. Shome et al. (2022) introduced a novel catalog of data smells by analyzing 25 datasets from Kaggle.[2] Their work identified and analyzed 14 new data smells, further enhancing the understanding of data quality issues in ML projects. All these findings can aid researchers and practitioners in improving data handling and preprocessing procedures.

From the actual contribution that Sculley et al. (2015) give to the definition of technical debt in AI-enabled systems, several studies are conducted to detail the state of knowledge of these issues. Tang et al. (2021) performed an empirical study analyzing 26 ML projects, where they introduced seven new ML-specific technical debt types, primarily concentrating on aspects related to the model code. Their findings shed light on unique challenges and debt types specific to ML projects, helping researchers and practitioners mitigate technical debt effectively. Bogner et al. (2021) conducted a systematic mapping study to identify and analyze technical debt in AI-enabled systems. They identified 72 antipatterns, most related to data and model deficiencies. Their work provides valuable insights into the specific areas requiring attention to effectively manage technical debt in AI projects. While these studies perform empirical analysis of literature and projects, there is still a missing link with the state of the practice, understanding how practitioners perceive the presence of AITD issues and how they manage them. Moreover, the outcome from these studies (from Foidl and Felderer (2019), Foidl et al. (2022), Shome et al. (2022), Bogner et al. (2021) and Tang et al. (2021)) focus on the technical debt issues and antipatterns that are related to data and model. Our work wants to give the same contribution to the practitioners in the context of technical debt management for the code and the architecture of AI-enabled systems.

Gesi et al. (2022) conducted an empirical study on deep learning projects, focusing on the frequency of code smells. In addition to analyzing the existing code smells, they identified five new codes specific to deep learning projects. This study enriches the understanding of code quality issues in the context of deep learning and highlights the importance of tailored approaches to tackle technical debt in ML projects. While the study focused on identifying code smells specific to deep learning projects, it did not thoroughly explore the spread and impact of AITD issues in those projects. Identifying five new code smells is necessary to enrich the understanding of code quality issues. Alahdab and Çalıklı (2019) conducted a case study addressing AITD issues in ML models. Their study extensively examined technical debt in *Glue Code*, *Pipeline Jungle*, and *Multiple Language Smell*. By identifying and addressing these issues, their work enhances ML projects' overall reliability and maintainability. The study conducted by Gesi et al. (2022) and Alahdab and Çalıklı (2019) also focuses on technical debt issues affecting the code and the architecture of those systems. However, there is still a lack of comprehensive assessment of the practitioners' perception of these issues in real projects. A clear and complete analysis that collects information on the relevance, severity, impact on quality aspects, and mitigation strategies from the practitioners' perspective is necessary to understand how practitioners face these issues in the industry to guarantee high quality and prioritize them.

Finally, solutions are proposed to face and prevent the presence of technical debt in AI-enabled systems. Breck et al. (2017) presented 28 monitoring tests to improve readiness and pay down technical debt in ML systems. These monitoring tests serve as practical guidelines for practitioners to assess the quality of the ML development process, enabling them to detect and prevent potential issues during the design of AI-enabled systems. Malakuti et al. (2021) proposed an information meta-model to manage and integrate digital twin models, aiming to prevent *Pipeline Jungle*, *Undeclared Consumers*, and *Data Dependency* issues. This contribution highlights the importance of managing technical debt in the context of integrating multiple models in ML-enabled systems.

In light of the existing research presented, there is a notable gap in AI-enabled systems regarding the practical approaches practitioners

[2] Kaggle platform: https://www.kaggle.com/

adopt to address technical debt. While previous studies have delved into the characteristics of AITD and provided insights into their consequences, there needs to be more research that analyzes the practitioners' perceptions regarding the implications of AITD issues. To bridge this gap, exploring practitioners' perspectives working on AI-enabled projects offers a valuable solution. By gaining insights into how AI practitioners perceive and interact with AITD issues in their work, we can link the theoretical knowledge defined in the literature and the practical realities development teams face. This empirical perspective helps validate the relevance and applicability of AITD issues already defined and provides a nuanced understanding of the challenges practitioners encounter and their strategies.

3. Research method

This section defines research objectives, outlines the rationale, and explains each step we performed.

3.1. Research objective and questions

Our empirical investigation analyzed practitioners' prevalence, effects, and mitigation strategies when dealing with AI Technical Debt (AITD) issues, i.e., suboptimal or compromised design and implementation choices made during the development and deployment of AI-enabled systems. The focus on the set of AITD issues that strictly affect the code and the architecture was motivated by the low research effort conducted on these two aspects and the high negative impact of gaining low quality on the architecture and the code stated by Shivashankar and Martini (2022) and Lenarduzzi et al. (2021b).

To pursue our investigation, we first applied the Goal-Question-Metrics approach (Basili et al., 1994) to extract relevant research questions that could address our objective. More specifically, the goal of our study was the following:

Analyze AITD issues
for the purpose of identifying
their frequency, severity, impact, and management strategies
from the point of view of AI practitioners
in the context of building and maintaining AI-enabled systems.

From the definition of the main goal, we defined the following research questions:

> **RQ$_1$**. *How frequent are AITD issues affecting the code and architecture from the practitioner's perspective?*

The motivation behind studying the frequency of AITD issues from the practitioner's perspective is to gain insights into the prevalence of these issues in software development. While these issues are defined and discussed in the literature, it is still challenging to understand the actual presence in the state of the practice. By understanding how frequently practitioners encounter AITD issues, we can prioritize and address the most significant challenges. **RQ$_1$** helps devise strategies to mitigate risks, allocate resources effectively, and develop proactive measures. Investigating the frequency of the nine aforementioned AITD issues could provide valuable insights into real-world challenges and aid decision-making processes in software development.

> **RQ$_2$**. *How severe are AITD issues affecting the code and architecture from the practitioner's perspective?*

Understanding the severity of AITD issues from the practitioner's perspective is crucial for effectively prioritizing efforts and addressing challenges. **RQ$_2$** helps allocate resources toward resolving the most critical AITD issues and improving the overall quality of the development process. Additionally, assessing the severity of these nine AITD issues fosters a culture of continuous improvement by creating awareness and promoting collaborative efforts to address these challenges proactively.

> **RQ$_3$**. *What is the impact of AITD issues affecting the code and architecture from the practitioner's perspective?*

RQ$_3$ focuses on assessing the impact of introducing AITD issues from the practitioner's perspective. After assessing the severity of each AITD issue, **RQ$_3$** aims to increase the awareness of the practitioners encountering AITD issues, highlighting the specific quality aspects that are influenced or affected when introducing these issues. This analysis is crucial for gaining insights into the potential risks, challenges, and trade-offs associated with AITD.

> **RQ$_4$**. *What strategies do practitioners use to identify and mitigate AITD issues affecting the code and architecture?*

The last research question aims at finding possible strategies to manage the proposed nine AITD issues. The set of discovered strategies could inspire automated techniques for AITD management, creating an ideal set of guidelines that helps practitioners face AITD issues while building AI-enabled systems.

Altogether, the findings from the four **RQ**s would inform the software engineering research community about the AITD types that are more relevant in practice, their impact on multiple properties of AI-enabled systems, and how practitioners currently deal with them. These pieces of information may be exploited by researchers to increase awareness on the matter and to prioritize their research efforts, *e.g.*, by investigating the AITD having the highest impact on practice, other than to devise novel, practitioner-informed techniques and methods that would support the identification and mitigation of AITD.

3.2. Research method

As Easterbrook et al. (2008) suggests, several research methods can be used to gain insights into AITD issues. We opted for a survey study to collect data from a larger sample of participants and understand the behavior of AITD issues in practice. We aimed to collect comprehensive information from a large sample of participants, particularly practitioners with experience in artificial intelligence systems, to improve our understanding of AITD issues and their impact on the development process. Our survey study explored the practitioner's perspective on each AITD issue, analyzing its frequency, severity, impact, and management strategies. We adopted Kitchenham and Pfleeger's survey design principles (Kitchenham and Pfleeger, 2002a) to structure our survey process, comprising multiple steps. We initiated the survey process by extracting the study context (Section 3.3) to identify and select a set of AITD issues for analysis. Subsequently, we designed the survey structure and conducted a pre-screening (Section 3.4). To ensure its reliability, the survey underwent validation using empirical standards and a pilot test (Section 3.5). Finally, we executed and administered the survey (Section 3.6), collected the data, and performed data cleaning and analysis to address our research questions (Section 3.7). Detailed information about the survey is provided in the following sections.

3.3. Context of the study

The *context* of the study was represented by *objects*, i.e., the AI-related issues in the scope of the study, and *subjects*, i.e., the participants of the survey. As for the objects, our research focused on identifying potential AITD issues affecting the code and the system's architecture. We thoroughly analyzed the existing literature on the topic of explicit issues that can introduce technical debt specific to AI-enabled systems, including studies by Sculley et al. (2015), Gesi et al. (2022), and Bogner et al. (2021). From the initial set of 77 AITD issues collected from the three sources, we selected all the issues related to the architecture and the code, identifying 13 issues. We chose issues with the most straightforward definitions, as many of the AITD problems we encountered were quite broad and difficult to pinpoint to a specific

Table 1
Definitions of the collected AITD issues.

AITD Literature Selection	
AITD Issue	Issue Definition
Jumbled Model Architecture (JMA)	Model architecture components are cobbled together and difficult to understand and maintain individually (Gesi et al., 2022).
Pipeline Jungle (PJ)	These pipelines organically evolve as new signals are identified and new information sources are incrementally added. Without care, the resulting system for preparing data in an ML-friendly format may become a jungle of scrapes, joins, and sampling steps, often with intermediate file output. Managing these pipelines, detecting errors, and recovering from failures are all difficult and costly (Sculley et al., 2015; Bogner et al., 2021).
Multiple Language Smells (MLS)	It is often tempting to write a particular piece of a system in a given language, especially when that language has a convenient library or syntax for the task at hand. However, using multiple languages often increases the cost of effective testing and can increase the difficulty of transferring ownership to other individuals (Sculley et al., 2015; Bogner et al., 2021).
Undeclared Consumers (UC)	Often, a machine learning model m_a prediction is made widely accessible at runtime or by writing to files or logs that other systems may consume later. Without access controls, some of these consumers may be undeclared, silently using the output of a given model as an input to another system (Sculley et al., 2015; Bogner et al., 2021).
Correction Cascades (CC)	There are situations in which a model m_a for a problem A exists, but a solution for a slightly different problem A' is required. In this case, learning a model m'_a that takes m_a as input and learns a small correction as a fast way to solve the problem can be tempting. However, this correction model has created a new system dependency on m_a, making it significantly more expensive to analyze improvements in the future. The cost increases when correction models are cascaded, with a model for problem A'' learned on top of m'_a, and so on, for several slightly different test distributions (Sculley et al., 2015; Bogner et al., 2021).
Glue Code (GC)	ML researchers tend to develop general-purpose solutions as self-contained packages. Generic packages often lead to a glue code system, in which a massive amount of supporting code is written to get data into and out of general-purpose packages (Sculley et al., 2015; Bogner et al., 2021).
Deep God File (DG)	A file containing multiple components of an AI system, *e.g.*, model training, validation, and testing (Gesi et al., 2022).
Scattered Use of ML Libraries (SML)	Scattered use of ML API in multiple files. Once ML API needs to be modified, the practitioners must modify places across several files (Gesi et al., 2022).
Unwanted Debugging Code (UDC)	A debugging code fragment, method, or class is no longer used, but it is still part of the source code (Gesi et al., 2022).

scenario. For example, we included *Pipeline Jungle* as it clearly indicates the affected object (*e.g.*, the pipeline), the event (*e.g.*, continuous growth), and the effect on the system (*e.g.*, decreased maintainability of the pipeline). However, we excluded any AITD issues identified in the literature that lacked these specific elements (*e.g.*, "Monolithic Pipeline"). The motivation behind this selection is twofold. On the one hand, the amount of data possible to collect from each participant can be limited by the participant's availability time. It is crucial to strike a balance and avoid making the survey too overwhelming, as this can lead to participants losing interest or abandoning the survey. On the other hand, it is imperative to ensure that any AITD issues in the survey are clearly defined so participants can accurately recognize and respond to them. This is especially critical during the vignette representation step, as unclear issues can result in biased answers based on a misrepresented situation of the vignette. As a result of this analysis, we collected nine AITD issues affecting the code and the architecture, as stated in Table 1.

As for the *subjects*, our research focuses on gathering perceptions about issues that appear in AI-enabled systems, specifically in the code and the architecture. To obtain a comprehensive overview of the perception of AITD, we tried to gather insights from diverse perspectives in our participant selection process. We aimed to gain practical insights into the AI landscape, engaging with professionals actively involved in building AI-enabled systems. Within this process, distinct roles are assigned with unique priorities and goals, and individuals approach the AI component with varying levels of expertise — from model architecture experts to data scientists with limited experience in this field. Consequently, this study's subjects are AI practitioners with experience interacting with AI components. The eligible participants for this study can be involved in different AI components, such as data preparation, model training, model deployment, and application maintenance. Therefore, we defined the subject of this study as practitioners with a baseline understanding of AI, but including different roles and perspectives among the several facets an AI-enabled system includes.

3.4. Survey design

The survey was structured following the principles defined by Kitchenham and Pfleeger (2002b). We also followed the recommendations provided by Reid et al. (2022), which allowed us to increase the quality of the applied method and manage threats related to the participants' knowledge. In particular, we applied the following recommendations:

- We introduced a pre-screening survey to check the skill level of the participants.
- We added a skill and attention check question to check whether participants who self-report the requested skill meet our criteria (labeled in the pre-screening survey as PD12).
- We included examples with vignettes to increase participants' confidence in conducting our study.
- We included a pilot test to check the comprehensibility and clarity of the survey.

The participants' information collection was divided into two parts, as shown in Fig. 1. The first part of the research focused on selecting participants with experience developing AI-enabled systems by pre-screening.

Pre-screening. In this part, we provided each participant with the survey introduction information that describes the role of the participants eligible for the study's context. In detail, we explicitly required that the participants' roles be related to artificial intelligence. Given the diverse interpretations of AI, practitioners may develop their understanding from various contexts, such as different tasks (*e.g.*, classification or regression) and techniques (*e.g.*, Machine Learning or Deep Learning). Therefore, we built a pre-screening survey that involves participants who work in all the AI-related fields, ensuring a baseline understanding. The pre-screening consisted of two essential sections of questions. In the first section, we asked participants about their role in their company and their level of experience in general skills related to building AI-enabled systems: Programming, Artificial Intelligence, and Software Engineering. The questions about the experience are presented on a Likert Scale, where the participant can explain his knowledge level on a 5-point scale (i.e., Novice, Basic, Intermediate, Advanced, Expert) (Nemoto and Beglar, 2014). The second section included questions to understand which part of an AI-enabled system the participant is responsible for. We proposed a sample of the AI pipeline to let the participants recognize the phases in which they are involved. Using the five-point scale, each participant reported their experience with the practices included in an AI application, starting from the data preparation phase to the model deployment phase. Additionally, we asked which activities explain their work. Finally, we included a

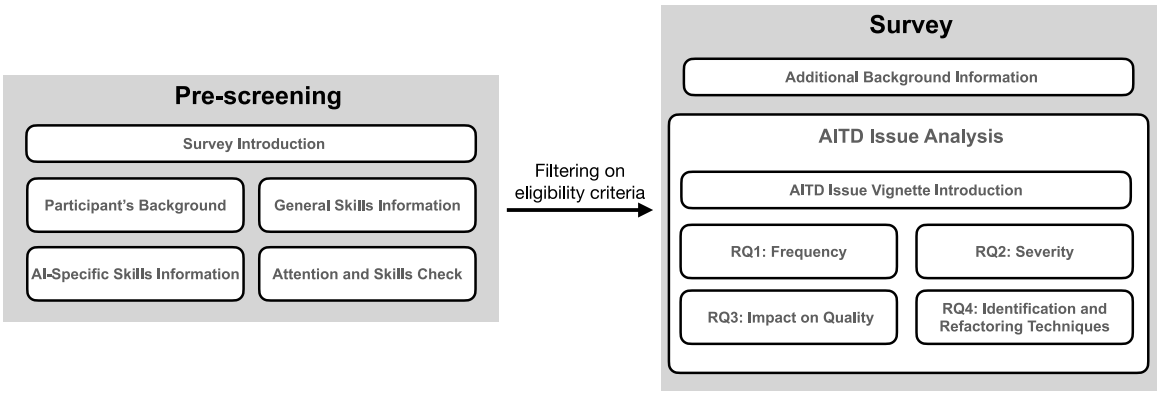


Fig. 1. Structure of the pre-screening and the survey.

competence and attention check question related to the practice used to build an AI-enabled system; In particular, we asked the participant which data balancing techniques should be applied in a specific context. Table 2 gives an overview of the questions delivered to the participants. Afterward, we defined a set of constraints in the pre-screening process to focus on the answers from participants who considered having experience in the domain. The participant, to be considered appropriate for the survey, must respect the following constraints:

1. The participant must correctly answer the competence and attention check question (PD12).
2. The participant must know Programming at least equal to two out of five.
3. The participant must know about artificial intelligence (PD4) or software engineering (PD5) at least equal to three out of five.
4. If the participant respects the previous constraint, the participant must know the other field at least equal to two out of five.
5. The participant must know about monitoring, maintenance, and deployment of AI models (PD9) or knowledge in using machine learning libraries (PD10) at least equal to three out of five.

These criteria prioritize participants with expertise in systems, including software engineering, programming, and artificial intelligence. Moreover, a focus on participants' proficiency in managing various phases of AI-enabled systems has been incorporated. These phases include monitoring, development, and deployment. Proficiency in these areas signifies participants' practical experience in handling the complete lifecycle of AI-enabled systems. By utilizing these criteria, we have identified a group of participants who can effectively analyze issues and technical debt. Their practical experience and knowledge in AI-enabled systems could let the participants understand the situations represented through the vignette and relate experiences to the AITD issues proposed. After selecting the participants, we conducted the main survey, which was structured in two sections.

Main survey. First, we asked two questions related to the dimension of the team and the organization in which practitioners work to have a complete vision of their activities while developing AI-enabled systems. Then, we asked a set of questions to answer the research questions for each instance analyzed by the participants. In detail, the second section started with the representation of the vignette to let the participant recognize the issue of the situation and analyze it with a practical example. We followed the methodology of *Experimental Vignette Study* introduced by Atzmüller and Steiner (2006) to create the vignette in a standardized, concise and precise format and increase the level of understandability of the survey from the practitioner's perspective. This method transforms the formal definition of an AITD issue into a more detailed and intelligible form through a vignette. We created a vignette for each instance to ensure that participants understand the context, activity, and problems associated with AITD entirely. These vignettes, as demonstrated in Fig. 2, depict situations where AITD issues may arise, considering the timing of the situation, the event, and the actions of the practitioners.

By presenting issues related to AITD in the form of *situations*, participants can reason about the circumstances surrounding AITD issues rather than just its definition. This approach allows participants to provide information on the characteristics of AITD issues in a relatable and easy-to-understand way, based on common ground and mitigating the risk of diverse interpretations of an issue.

After understanding the issue through the vignette, each participant answered questions on the characteristics that address our RQs. We designed several questions to analyze the frequency and the severity of AITD issues ($\mathbf{RQ}_1$ and $\mathbf{RQ}_2$). In particular, we asked the participant (i) how often they had encountered an instance of such AITD in their experience, (ii) to what extent and (iii) why this situation is problematic, and (iv) the effort needed to identify and refactor the AITD issue. To understand the impact of AITD issues ($\mathbf{RQ}_3$), we asked several questions based on six AI quality-related aspects: (i) understandability, (ii) performance, (iii) maintainability, (iv) evolution, (v) defect-proneness on other components, and (vi) coupling. The participants provided the agreement level on the statement proposed as a Likert Scale (Strongly Disagree to Strongly Agree). Finally, after analyzing each assigned AITD issue through the vignette proposed, the participant could provide further information on AITDs through the open question and release his email for future investigations. Table 3 illustrates the list of questions delivered to each participant.

3.5. Survey validation

Before starting with the survey study, we conducted a pilot test recruiting five researchers experienced in performing online surveys, as proposed by Reid et al. (2022), to assess the survey in various areas, such as the clarity of the questions and the accuracy of the answers.

We considered the knowledge gap between the researchers in the pilot test and the AI practitioners involved in the survey; therefore, we included an industry participant with experience in AI-enabled systems. Specifically, we included a Backend Software Engineer in AI-enabled systems who works as a practitioner for feedback. The addition of an industry-related participant for the pilot test helped us to understand if the terms used in the survey could be understandable by the participants subject to the investigation since the focus of our study is to collect the perspective of practitioners. Several suggestions helped to increase the quality of the survey study. Regarding the pre-screening questions, participants suggested increasing the detail of scale values of the questions related to the general skills (PD3, PD4, PD5) and AI-specific skills (PD6-PD10). From this feedback, we defined values for general skills to range from Novice level to Expert and experience level in AI-specific skills from Very Poor to Excellent. Regarding the main survey, participants highlighted a lack of clarity for the vignette

Table 2
Prescreening questions.

First Section		
Questions		Question Types
PD1	What is your role in the company?	Multiple Choice
PD2	How many years of experience do you have in your role?	Multiple Choice
PD3	Define your knowledge level in: Programming	Likert Scale
PD4	Define your knowledge level in: Artificial Intelligence	Likert Scale
PD5	Define your knowledge level in: Software Engineering	Likert Scale
Second Section		
Questions		Question Types
	Considering the section figure, I have experience in [...]	
PD6	[...] Data ingestion, data aggregation or other types of data preparation	Likert Scale
PD7	[...] Model selection and model training	Likert Scale
PD8	[...] Model validation, review process and improving model	Likert Scale
PD9	[...] Monitoring, maintenance and deployment plan	Likert Scale
PD10	[...] Using Machine Learning libraries and execution of the model	Likert Scale
PD11	Taking a look at the section figure represented above, Which of these activities explains better your work in AI systems? (One or more of it)	Multiple Choice
PD12	In which part of the dataset do you apply the data balancing techniques (*e.g.*, Oversampling, Undersampling)?	Multiple Choice

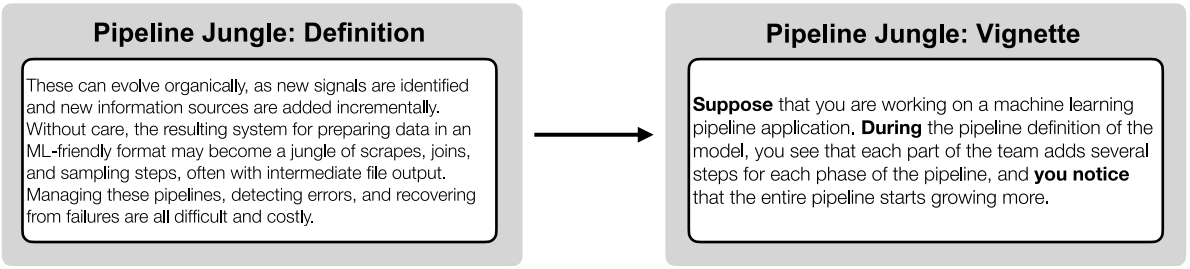


Fig. 2. Application of vignettes to define the Pipeline Jungle issue.

Table 3
Survey questions.

Preliminary Questions		Question Types
S1D1	How many members are in your team?	Multiple Choice
S1D2	How many employers are in your company?	Multiple Choice
RQ1		Question Types
S2D6	How often do you encounter this situation?	Likert Scale
RQ2		Question Types
S2D1	How much do you find this situation problematic?	Likert Scale
RQ3		Question Types
S2D2	Why it is (or not) problematic?	Open Question
	Please vote on the following sentences: I think that this situation ...	
S3D1	...may make other components of the application more defect-prone.	Likert Scale
S3D2	...makes the level of interpretability/explainability of the model more complex.	Likert Scale
S3D3	...makes it difficult to evolve the application.	Likert Scale
S3D4	...may impact and decrease the performance of the model.	Likert Scale
S3D5	...could contribute to create dependencies, increasing the coupling of the application.	Likert Scale
S3D6	...is hard to maintain, increasing the cost and the effort to maintain the application.	Likert Scale
RQ4		Question Types
S2D3	What are your practices of identification for this situation?	Likert Scale
S2D4	Could you tell us something more about identification? (Specify the name of the tool or some info about the activity of the answer you selected in the previous question)	Open Question
S2D5	How much effort do you think is necessary to identify this situation?	Likert Scale
S2D7	How much effort do you think is necessary for refactoring, mitigating, or improving this situation?	Likert Scale
S2D8	How do you refactor this situation?	Open Question
Additional Questions		Question Types
S4D1	Could you give us any additional feedback on your activity or any bad practices you have observed in your experience? *	Open Question
S4D2	If you had like to be considered for future surveys or interviews, please enter your email address.*	Open Question
	*= The proposed question is optional.	

representing AITD issues, particularly for *Jumbled Model Architecture* and *Glue Code*. To address this issue, we rewrote the vignette, adding examples that help the practitioners to recognize the situation. Finally, participants suggested removing questions related to gender. Although gender is an important and relevant factor in various research contexts, our study of AI Technical Debt focuses specifically on technical aspects of AI and its implications. Furthermore, avoiding questions on gender ensures that participant privacy is respected and that the study remains focused on its core research objectives without inadvertently collecting sensitive or unnecessary personal information. For these reasons, we agreed with the participants to exclude gender-related questions that could hinder their privacy.

Table 4
Task assignment.

Participant	AITD Vignette
Group P1	Subsets A+B
Group P2	Subsets B+C
Group P3	Subsets A+C

3.6. Survey administration

One of the key choices was selecting the platform to recruit the survey participants. Social media or a specialized recruitment platform could have been a reliable option. However, it is necessary to be cautious about potential biases in the sample, as social media users may not represent the target population accurately. Opting for a specific recruitment platform like Prolific can be beneficial due to its focus on academic research and access to pre-screened, willing participants who have expressed interest in participating in studies. Furthermore, these platforms often offer features to control demographics and manage participant compensation, simplifying the research process.

We recruited the participants through the Prolific platform,[3] which allowed us to select experts in AI-enabled system development. We could control the recruitment process by defining filters that explicitly address the participants' research on specific domains. Given the absence of specific filters precisely aligned with the role we sought, we used filters focused on participants with expertise in software development techniques and computer programming.

The process of pre-screening analysis began on June 28, 2022, and lasted for nearly four hours to obtain answers from all participants. With these filters, we recruited 500 participants with experience in programming and knowledge of software development techniques. From the initial set, 188 participants correctly answered the pre-screening question, *i.e.*, competence and attention check question, representing 37,8% of the selected population. We excluded those participants who did not meet the defined eligibility criteria and appointed 112 participants as eligible for the experiment. Afterward, some participants declared their unavailability to continue the experiment, leading to a group of *54* participants. This drop in participants could lead to insufficient information extractable from each participant. We had to consider the maximum time and effort a participant could spend on the survey and the amount of information that could be extracted. The between-subject approach, i.e., querying each participant about only a single AITD issue, could result in limited information for each AITD issue. The within-subject approach, i.e., asking participants questions about nine AITD issues, could expose the threat of survey fatigue, leading to careless answers or abandoning the survey. Because of this constraint, the survey is built upon the *mixed-subject design* approach (Atzmüller and Steiner, 2006), asking each group of participants questions about two different subsets of vignettes established beforehand.

In particular, we defined three subsets of vignettes describing AITD and three participant groups. The representation of the subset of the vignette was the following:

- Subset A: Glue Code, Multiple Language Smell, Undeclared Consumers.
- Subset B: Pipeline Jungle, Scattered Use of ML Libraries, Correction Cascades.
- Subset C: Jumbled Model Architecture, Unwanted Debugging Code, Deep God File.

Table 4 illustrates the definition of the subsets and their combination for the survey. Each group participant answered questions about six issues that can cause AITD. Using the mixed-subject design approach, each participant provided answers for six AITD issues, allowing us to obtain more answers for each AITD.

The main survey was conducted on August 3, 2022, and we needed two days to collect answers from all participants. Starting from 54 participants' answers, we collected 38 answers related to the issues of Group A, 36 answers related to Group B, and 33 answers related to Group C.

3.7. Data analysis

Once the data were entirely collected from the participants, we manually filtered the answers from those who did not take the survey seriously; this step excluded one answer. As a result, we collected 53 answers, decreasing the number of answers for Group A to 37. The first type of question included multiple choice questions or in Likert Scale format. Such data in an ordinal scale was analyzed using non-parametric statistics, as proposed by Briand et al. (1996); specifically, we analyzed the distribution and the frequency of each value. The second type of question included open answers, which helped identify other quality-related aspects affected by the introduction of AITD issues and highlighted the solutions practitioners apply to refactor the issues. Such open questions were analyzed following the *iterative content analysis* method (White and Marsh, 2006). This qualitative analysis process, served to extend the set of quality-related aspects that we initially proposed to participants. By embracing the diverse perspectives of AI practitioners, we aimed to uncover a comprehensive spectrum of quality-related facets impacted by AITD issues including factors that analyze the system on a broader perspective, including all the factors related to quality that can be affected ($\mathbf{RQ}_3$). In addition, we use this process to examine useful refactoring techniques ($\mathbf{RQ}_4$). By adopting this approach, we ensure that our analysis includes all innovative and context-specific solutions that practitioners use, enhancing the overall practicality of our findings. In particular, we performed sequentially the following steps:

1. **Micro-Analysis**: The first manually analyzed the answers related to the open questions. The keywords of each answer are extracted and processed to obtain a label that identifies the concept. To avoid the subjectivity introduced by the first author, the second and third authors assessed the quality of the label assignment. They manually analyzed the label of each open answer and provided feedback and suggestions to improve their suitability (*e.g.*, merging two semantically similar labels). This phase ended by defining an agreement status between the label proposed by the first author and the reviewers' feedback.
2. **Categorization**: The analysis continued by categorizing the labels that describe similar concepts and splitting those explaining more than one concept into several categories. This step allowed for isolating information extracted from a single response and was conducted through an iterative process that allowed for a gradual and more precise grouping of labels.
3. **Saturation**: The categorization was performed until reaching saturation, observable when an iteration does not perform any changes in the set of categories.

Finally, we manually analyzed the answers to the question that asks the participants additional feedback, represented in question S4D1 of Table 3.

4. Analysis of the results

This section reports the results of our survey, including the preliminary study and the analyses concerning the prevalence, severity, and impact of AITD, as well as the management strategies to mitigate its effects.

[3] https://www.prolific.co

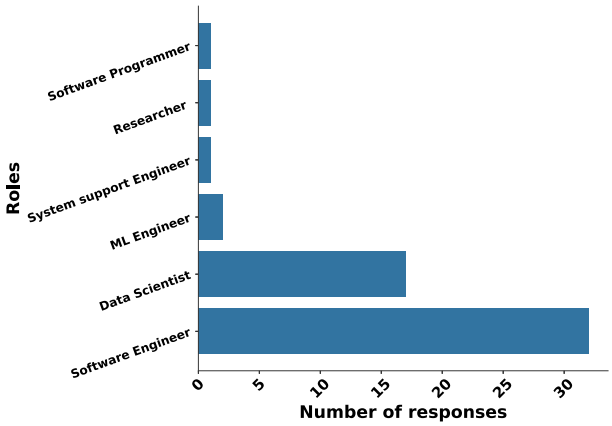


Fig. 3. Distribution of the roles of the participants.

4.1. Preliminary analysis

The study participants were evaluated based on specific criteria to gain insight into their work environment. The selection process of participants was based on their role, which was then analyzed. The results, as shown in Fig. 3, reveal that the majority of the participants were software engineers (59.3%) and data scientists (31.5%). However, the participants also included professionals in related fields, such as AI-enabled system management and development, including researchers, system support engineers, and software programmers. These findings suggest that the study captured perspectives from professionals involved in multiple aspects of AI-enabled systems.

As shown in Fig. 4(a), a significant number of the surveyed participants, 24 out of 53, are employed in small companies with fewer than ten employees. The remaining participants are from medium- or large-scale organizations with more than ten employees. Most participants work in small teams, *i.e.*, 41 in units of less than ten members, as depicted in Fig. 4(b). Despite most participants being from small companies and teams, the collected data includes practitioners' perspectives from large companies and teams, with 13 participants working in a company with more than 100 employers and six working in teams with more than 20 members.

The participants' skill levels were evaluated based on their knowledge of AI-related fields, as shown in Fig. 5(a). 78% of the participants reported having a good understanding of AI, rated with three or higher out of five. The remaining 22% rated their AI knowledge as two out of five had strong Programming and Software Engineering skills. As for Software Engineering skills, 92% of the participants rated them with three or higher out of five. Lastly, 65% of the participants indicated advanced programming skills, with 30 rating them with four or higher out of five.

Finally, we analyzed the ML-Specific related skills for each stage of an ML pipeline by using ML libraries to build an ML application. The results depicted in Fig. 5(b) highlight the notable proficiency of a significant proportion of survey participants in various machine learning-specific competencies. Remarkably, the findings demonstrate that most respondents (87%) exhibited sufficient proficiency in Model Deployment, with a similar trend observed for using ML Libraries. Participants who reported expertise in these two areas of the ML pipeline also demonstrated competencies in Data Preparation, Model Training, and Model Evaluation, with a considerable proportion (91%, 87%, and 89%, respectively) exhibiting adequate proficiency. Notably, most participants exhibited at least intermediate-level skills in these competencies. The competence and attention check question results complemented these outcomes, enabling accurate evaluation of participant eligibility for the survey.

4.2. RQ_1. How frequent are AITD issues from the practitioner's perspective?

We collected the practitioners' answers regarding the prevalence of AITD issues. Fig. 6(a) displays the distribution of answers on a Likert scale, measuring the frequency of AITD issues found by practitioners. The results indicate that the majority of practitioners involved reported "never" encountering specific situations attributable to the AITD issues presented, mainly for *Undeclared Consumers* (14 answers), *Correction Cascades* (14 answers), and *Multiple Language Smell* (14 answers). In addition, most participants reported "rarely" encountering issues of most AITDs, including *Undeclared Consumers* (15 answers) and *Scattered Use of ML Libraries* (17 answers). However, some participants reported encountering AITD issues "sometimes" in their experiences, such as in *Glue Code* (14 answers), *Jumbled Model Architecture* (10 *answers*), *Pipeline Jungle* (nine answers), and *Scattered Use of ML Libraries* (nine answers). Finally, less than 10 participants frequently encountered AITD issues (denoted as "often" or "always"). In detail, the unmanageable growth of the AI pipeline that provokes *Pipeline Jungle* is frequently encountered by nine participants (in which six denoted it as "often" encountered and three as "always" encountered). Other AITD issues are reported to be frequently encountered by seven participants or less. These findings highlight that while most participants reported rarely or never encountering AITD issues, a small portion of respondents have already experienced these challenges in their AI development experiences. Therefore, this study catches information on the properties of AITD issues from two types of AI practitioners, classified as follows:

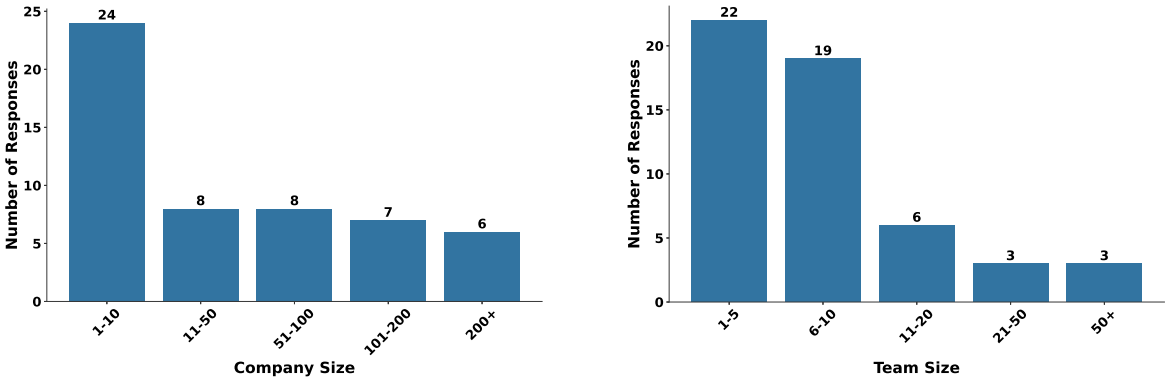
- AITD unfamiliar practitioners: practitioners that never or rarely encountered the specific AITD issue.
- AITD familiar practitioners: practitioners that sometimes, often, or always encounter the specific AITD issue.

The analysis described in the following sections describes the results first from a general perspective of the perception of AITD issues. Subsequently, the results are discussed based on the familiarity level of the participants. It is essential to note that the number of participants in the two groups varies based on the answers that participants gave on the prevalence of each AITD issue. In the article, we reported the global results highlighting the aspects of the AITD issues. Tables and plots grouped by familiarity level are available in our online appendix.[4]

4.3. RQ_2 . How severe are AITD issues from the practitioner's perspective?

Considering the severity levels provided by the participants, six AITD issues out of 9 analyzed are reported to be "very problematic" or "extremely problematic" by participants. Almost all participants (30 out of 37) declared the issue of Undeclared Consumers to be of high severity. The leading cause that emphasizes the severity of *Undeclared Consumers* is free access to the model. The data collected from unknown customers is essential information used to understand the system's architecture and try to exploit it to perform malicious actions. Therefore, this AITD issue inside the system needs to be considered when analyzing possible unknown accesses. *Scattered Use of ML libraries* and *Deep God File* are reported to be highly severe by 27 participants. The uncontrolled increase of the size of the component and the scattered injection of libraries lead to an increase in the complexity of the class, which makes it challenging to maintain and understand the intended responsibility of the module. In detail, analyzing *Scattered Use of ML libraries*, participant (P53) stated: *"Libraries need to be correctly understood. As the team is small and very specific, only particular and few libraries are involved. All team members must be aware and at least have some knowledge of the libraries and dependencies."*. Therefore, the increased complexity of the modules can also reflect damaging socio-technical factors.

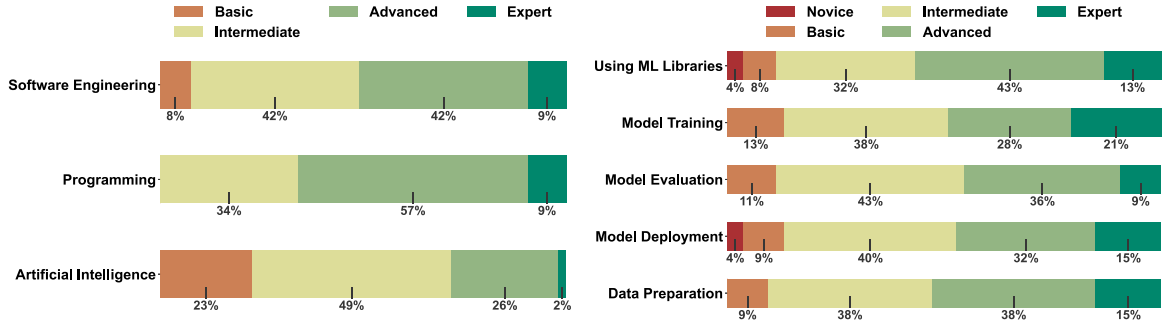
[4] Online Appendix: https://doi.org/10.6084/m9.figshare.24030456



(a) Distribution of the company sizes of the participants

(b) Distribution of the team sizes of participants

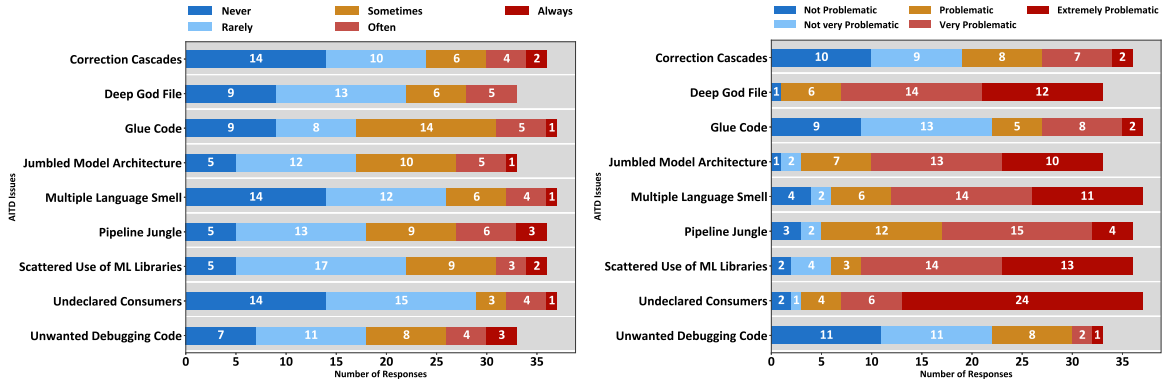
Fig. 4. Distribution of participants' company and team sizes of the participants selected for the questionnaire.



(a) General skills

(b) Machine learning specific skills

Fig. 5. Distribution of the skills of the participants selected for the questionnaire.



(a) Prevalence of AITD issues

(b) Severity of AITD issues

Fig. 6. Prevalence and severity of AITD issues in AI-enabled systems from the practitioner's perspective.

Moreover, *Jumbled Model Architecture* and *Pipeline Jungle* are both reported as highly severe by most part of the participants, in which 23 participants denoted as "very problematic" for *Jumbled Model Architecture* and 19 for *Pipeline Jungle*. The continuous addition of steps and modules is a critical aspect that needs to be managed to avoid a critical explosion of complexity that can affect the whole system. In this context, a participant warned (P51): *"Once multiple steps or processes are added to each pipeline phase, it will require more time and developers to maintain its lifecycle. This increases the chances of a disorganized program and might delay the release date"*.

Finally, *Multiple Language Smell* has been reported to be highly severe by 25 participants. The main problem caused by this AITD issue is mainly related to the human resources involved in the lifecycle of the AI-enabled system. When multiple languages are introduced and used inside the system, only some practitioners have the skill to use a determined language. As the participants identified, operating could be challenging if some components are written in different languages.

On the other hand, most participants reported not all of the AITD issues analyzed were severe. Analyzing the answers that participants delivered on the severity related to *Correction Cascades* and *Glue Code*, answers are equally distributed towards the different levels of severity proposed, leading to a not clearly stated high severity of these AITD issues. However, some of the participants still noted the criticality of introducing these AITD issues. Introducing *Correction Cascades* issue could lead to having uncontrolled dependencies with external applications. This situation is critical if a change in the system could lead to undesired effects on the consumers' applications without the possibility of analyzing the consequences, as stated by Participant P16. Finally,

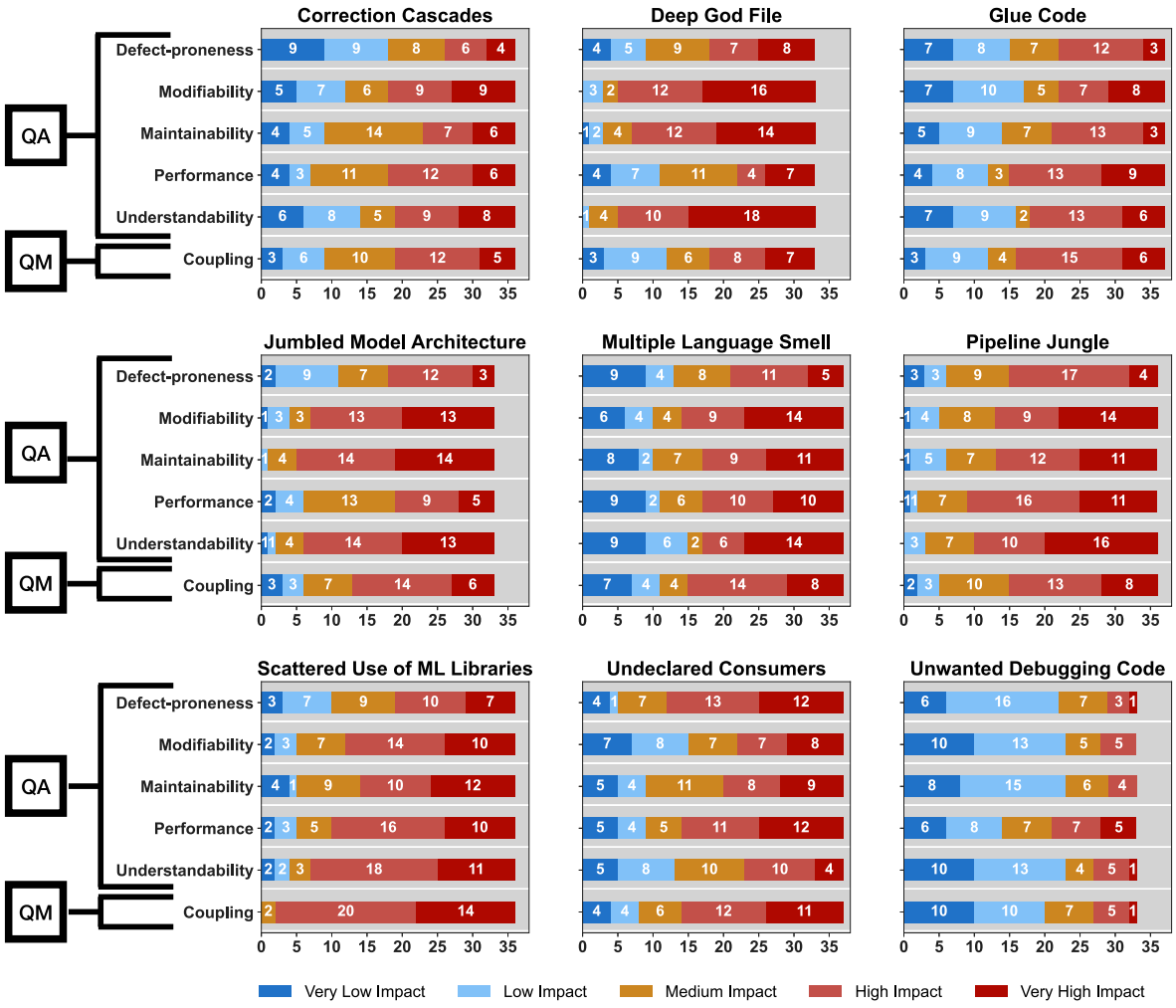


Fig. 7. Effect of AITD issues on the selected quality aspects from the practitioner's perspective. Quality-related aspects are organized into three main categories: Quality Attributes (QA), Quality Metrics (QM), and Socio-Technical Factors (ST).

Unwanted Debugging Code was reported to be low severe by 23 participants. While a few participants highlighted the problem of showing sensitive data through logging, most noted that this AITD would not significantly affect the system.

Subsequently, we analyzed practitioners' perceptions of the severity, considering their familiarity levels. The answers between the two groups do not present remarkably different severity levels for seven out of nine AITD issues, delineating that the severity perception denoted in the results is agreed upon by AITD familiar practitioners and AITD unfamiliar practitioners. However, some differences are engaging in two of the nine AITD issues analyzed. *Pipeline Jungle* has been denoted by both groups to be highly severe but with a particular remark analyzing *AITD familiar practitioners*. In detail, 11 participants out of 18 denoted this AITD issue as "very problematic" and one as "extremely problematic". Therefore, the severity of this AITD issue is strongly remarked on by practitioners who have experienced situations in which they encountered this problem. The situation is similar when analyzing the answers for *Undeclared Consumers*, in which no *AITD familiar practitioners* reported low or medium severity of this AITD issue. Moreover, also *AITD unfamiliar practitioners* reported an extremely high severity for this AITD issue.

4.4. RQ_3. *What is the impact of introducing AITD issues from the practitioners' perspective?*

To answer $\mathbf{RQ}_3$, we analyzed the quality related-aspects impacted by AITD, which could damage AI-enabled systems. First, we collected answers to the quality-related aspects and the distribution of the impact levels, as illustrated in Fig. 7. Second, we performed a content analysis to explore other quality-related aspects practitioners believe AITD could impact.

The content analysis process involved four iterations, each building upon the previous one to extract and categorize quality-related aspects affected by AITD issues. The fourth iteration identified the saturation state and the completion of the content analysis. In the first iteration, we organized 196 labels into 85 categories, providing an initial structure to the analysis. The second iteration narrowed the focus, isolating 31 categories directly related to quality-related aspects. Subsequently, in the third iteration, we identified 18 specific quality-related aspects that participants had highlighted. To illustrate this process, let us consider an example from the analysis of a participant's input (P42) in the context of the impact analysis of *Pipeline Jungle*. The participant mentioned "increased complexity leading to higher running costs and potentially harder debugging", and we extracted the label "Increased complexity and debugging difficulty" from this text. After extracting labels, we proceeded to categorize them. For instance, labels such as "Difficult to maintain" and "Difficulty of future maintenance operations" were grouped under a single category called "Maintainability issues" during this phase. Following these categorizations, we conducted an additional iteration to associate similar quality-related aspects into single categories. For instance, categories related to security, such as "Vulnerabilities", "Privacy Issues", and "Security", were grouped under the category "Security Issues". In

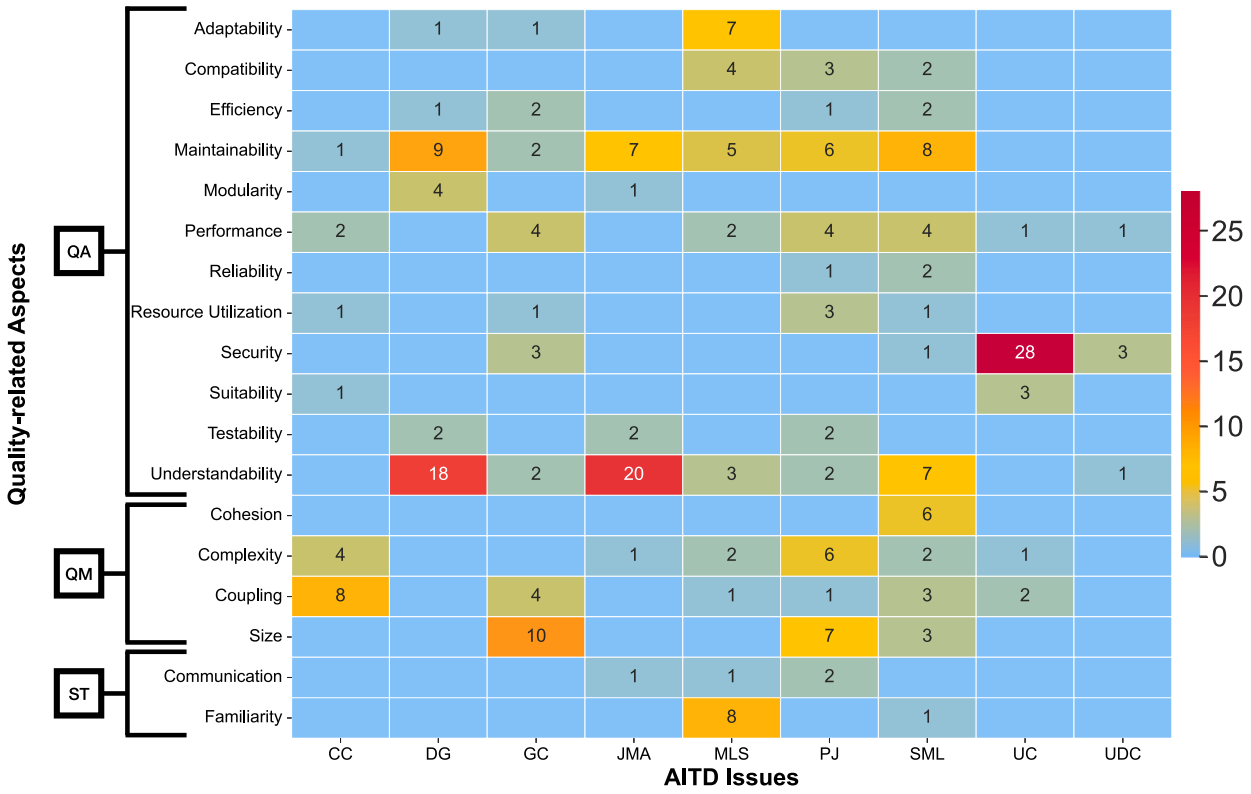


Fig. 8. Frequencies of practitioners' answers on mainly-impacted quality-related aspects by AITD issues, categorized in quality attributes (QA), quality metrics (QM) and socio-technical factors (ST).

summary, the four iterations of the content analysis process allowed us to progressively refine our understanding of quality-related aspects impacted by AITD issues, ultimately resulting in a structured and categorized set of insights.

Successively, we analyzed the relationship between each AITD issue and the quality-related aspects impacted. Fig. 8 summarize all the categorization labels applied to quality-related aspects for each AITD issue. *Understandability* is the most impacted aspect, having a total of 53 answers, and seven out of nine of the AITD issues analyzed impact this aspect. Subsequently, the system's maintainability and security are strongly discussed with 38 and 35 answers, respectively, where security is mainly impacted by *Undeclared Consumers*. Finally, from the extracted aspects, eight of these are impacted by at least four AITD issues. Additionally, Fig. 8 specifies the number of answers of all the quality-related aspects impacted by each AITD issue. The number of answers highlighting the impact of the specified AITD issue is reported for each quality-related aspect collected. In the following, we analyzed deeper the relationship between AITD issues and quality attributes, quality metrics and socio-technical factors.

Impact of AITD issues on quality attributes. Participants noted a significant impact of *defect-proneness* associated with almost all AITD issues, especially notable in the case of *Undeclared Consumers*, where 13 participants reported a "high impact", and 12 reported a "very high impact." Similarly, *Pipeline Jungle* was identified by 17 participants as having a high impact, with an additional four noting a very high impact. Therefore, introducing AITD issues will likely increase the probability of defects. Specifically, participants emphasized this impact on AITD issues closely associated with the system's architecture, which they noted could increase architectural complexity. *Modifiability* is reported to be affected by nearly all AITD issues, particularly by *Deep God File* and *Jumbled Model Architecture*.

Consequently, participants indicated that these issues significantly complicate performing changes and maintenance operations, with 28 participants highlighting challenges with *Jumbled Model Architecture* when handling the *maintainability*. *Performance* is notably impacted by *Pipeline Jungle*, where 27 participants reported a high impact. This suggests that continually adding steps to an AI pipeline in an uncontrolled manner could substantially reduce system performance. *Performance* is also adversely affected by *Scattered Use of ML Libraries*. *Understandability* is impacted by more than half of the proposed AITD issues, indicating that various factors related to these issues limit practitioners' ability to analyze and comprehend the components of the AI-enabled system. Additionally, among the quality attributes reported as significantly impacted by AITD, three of the six proposed quality attributes were confirmed as impacted, along with nine additional attributes. *Understandability* emerged as the most affected attribute, with 53 mentions, and was impacted by seven of the nine AITD issues analyzed. System *maintainability* was also notably affected, identified in seven of the nine AITD issues with 38 mentions, including nine related explicitly to *Deep God File*. System security was another primary concern, denoted in total by 35 participants and particularly impacted by *Undeclared Consumers*, as indicated by 28 participants.

Considering the quality attributes leveraged exclusively by participants, eight of the extracted aspects were affected by at least four AITD issues. Regarding the additional quality attributes reported by participants, *Multiple Language Smell* was noted to impact *adaptability* and *compatibility* by seven participants. Moreover, other quality attributes are included and denoted to be affected by the presence of AITD issues, especially concerning the *testability* of the system and *Resource Utilization*.

Impact of AITD issues on quality metrics. Considering the impact of the first analysis represented in Fig. 7, the coupling is significantly remarked as highly impacted in the presence of *Scattered Use of ML Libraries issue*, where 34 participants out of 36 denoted a high impact. Since this issue strictly creates dependencies between the system component and external libraries, it consequently increases the system's

coupling. Considering the qualitative analysis of the quality-related aspects extracted by the participants, three participants confirmed this outcome as the main factor this AITD issue can cause. Moreover, participants noted an affection for AITD issues regarding the complexity and size of the system. In detail, the presence of *Glue Code* and *Pipeline Jungle* implies the increase in the system's size, as stated by ten participants and seven participants. Finally, a strong connection between the cohesion of the components inside the system is strictly affected by the presence of *Scattered Use of ML libraries*.

Impact of AITD issues on socio-technical factors. Other than the quality attributes and metrics representing technical aspects of the AI-enabled system, participants leveraged the fact that the presence of these AITD issues can also affect socio-technical factors. Firstly, the presence of *Jumbled Model Architecture*, *Multiple Language Smell* and *Pipeline Jungle* are denoted to limit communication between team members. Interestingly, as stated by eight participants, the presence of *Multiple Language Smell* strongly affects the practitioners' familiarity with the system. Participants denoted that when a new component in a different language is added to the system's architecture, each future update and change will be more complicated if the practitioner faces more languages. Every practitioner working on a specific part of the system should have the knowledge to perform operations in two different languages. Specifically, a participant (P4) answered *"Because the two languages, or rather their input and output, may or may not be compatible and/or someone hired to maintain the code may or may not be able to code in both languages."* The familiarity of practitioners in the system could be also affected by *Scattered Use of ML libraries*, as stated by a participant.

Additional analysis on AITD issues impact. Analyses have been extended to consider the level of experience that participants have with AITD issues, collecting perspectives from both *AITD unfamiliar practitioners* and *AITD familiar practitioners*. The main outcome highlights that for three of the proposed AITD issues (*Scattered Use of ML libraries*, *Deep God File*, and *Unwanted Debugging Code*), there are no evident differences in the impact levels on the quality-related aspects as reported by both groups of practitioners. However, with the exception of *Unwanted Debugging Code*, practitioners from both groups recognize a critical impact on the six quality-related aspects we proposed. Interestingly, key differences were identified between these two groups. The high impacts observed in all the proposed quality-related aspects when considering the presence of *Pipeline Jungle* are mainly reported by *AITD familiar practitioners*. Specifically, none of the participants in this group reported a low impact for this AITD issue. Thus, the critical nature of this AITD issue is primarily recognized by practitioners who are aware of its presence and potential effects on the system. Similar patterns were observed when analyzing the impacts of *Deep God File* on the system's *understandability* and *Undeclared Consumers* on *defect-proneness*. However, among the nine proposed AITD issues, *AITD familiar practitioners* tended to report a higher impact compared to *AITD unfamiliar practitioners*, suggesting that the significant impact of these AITD issues is particularly acknowledged by practitioners who are aware of the consequences of these debts in the systems.

Analysis based on the familiarity level is also conducted for the quality-related aspects leveraged by the participants. In a broader context, *AITD unfamiliar practitioners* denoted 17 quality-related aspects impacted. In contrast, *AITD familiar practitioners* reported all the quality-related aspects extracted (18), adding *quality assurance* from the previous set. The main differences between the two groups are notable regarding the number of responses for specific AITD issues. Regarding the quality-related aspects impacted by *Pipeline Jungle*, *size* and *performance* are almost denoted by AITD familiar practitioners. In contrast, *maintainability* is reported to be impacted by the same number of participants in the two groups (three participants for each group). The impact of *Glue Code* on the *performance* of the system is exclusively denoted by *AITD familiar practitioners*. *AITD unfamiliar practitioners* mainly denoted that this AITD issue impacts the *size* of the system (6 participants), also confirmed by *AITD familiar practitioners* (4 participants). *AITD unfamiliar practitioners* exclusively reported the impact of *Scattered Use of ML Libraries* to the *understandability*. AITD unfamiliar practitioners almost exclusively denoted the impact of *Correction Cascades* on the *dependencies* of the system, while only one participant in the AITD familiar practitioners reported it. The situation is analogous for *Deep God File* on *maintainability* and for *Multiple Language Smell* on the *familiarity*. All the other relationships in analyzing the impact of AITD issues on these quality-related aspects are reported in both groups but stated particularly from *AITD unfamiliar practitioners*, given a higher number of participants.

Table 5
Identification strategies proposed by AI practitioners for the AITD issues.

AI TechnicalDebt Issue	M.R.	A.I.	P.T.	N/A
Correction Cascades	**17**	1	5	13
Deep God File	**20**	4	2	7
Glue Code	**16**	4	3	14
Jumbled Model Architecture	**19**	4	4	6
Multiple Language Smell	**16**	2	4	15
Pipeline Jungle	**18**	5	7	6
Scattered Use of ML Libraries	**27**	2	2	5
Undeclared Consumers	**13**	6	9	9
Unwanted Debugging Code	**14**	5	2	12

M.R. = Manual Review
A.I. = Automated Inspection
P.T. = Professional Team
N/A = Do not Identify

4.5. Management strategies

We analyzed the identification strategies leveraging the question that asks the participants to define the set of practices used to identify the specified situations, as stated in question S2D3 of Table 3. Table 5 collects all the strategies for each issue. From the collected results, it is possible to see how manual inspection is the most used strategy for most issues. At the same time, for *Multiple Language Smell*, *Glue Code*, *Correction Cascades*, and *Unwanted Debugging Code*, more than 10 participants do not have an identification strategy for these issues. Some participants highlighted the need for a dedicated professional team to detect these problems inside the system, especially for *Undeclared Consumers* and *Pipeline Jungle*. Additionally, analyses based on the familiarity level of the participants are conducted to understand the identification strategies identified by each group. For AITD familiar practitioners, the adoption of manual inspection is selected from most of the participants of this group for all AITD issues, especially for *Pipeline Jungle* and *Scattered Use of ML Libraries* that are selected from 11 participants. *AITD unfamiliar* group instead is mainly divided into participants who identify manual inspection as the main technique and participants who do not identify any identification technique. Particularly, *Scattered Use of ML Libraries* is identified through manual inspection by 16 *AITD unfamiliar practitioners*, while *Deep God File* from 12 *AITD unfamiliar practitioners*. The engagement of professional teams to identify AITD issues is almost reported by *AITD unfamiliar practitioners*, especially for *Undeclared Consumers*, where seven *AITD unfamiliar practitioners* and two *AITD familiar practitioners* report it. Additionally, the adoption of automated inspection is distributed similarly in the two groups. Finally, considering each issue, only a small number of the participants are supported using automatic tools to identify AITD. Participants use static analysis tools to identify *Pipeline Jungle*, *Scattered Use of ML Libraries*, and *Jumbled Model Architecture* (*e.g.*, SonarQube). Concerning *Pipeline Jungle*, two participants (P39, P40) identify and monitor the continuous growth of the pipeline through the orchestration of MLOps tools (*e.g.*, Kale). In conclusion, while AITD is considered a possible threat to AI-enabled systems' quality, manually checking and reviewing the artifacts is the most used technique.

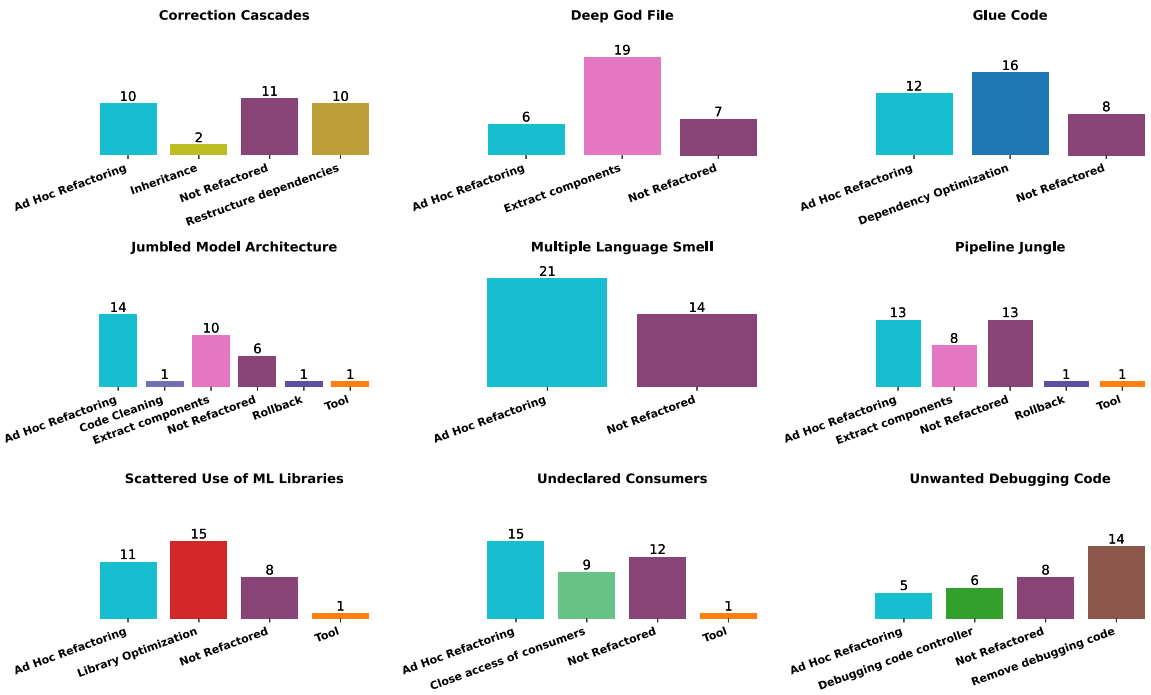
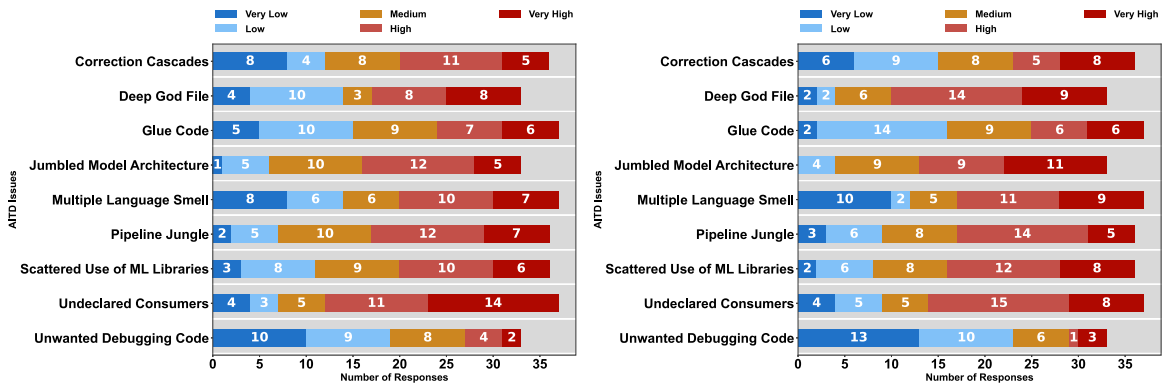


Fig. 9. Frequencies of practitioners' answers of refactoring techniques useful for facing each AITD issue.

The content analysis for the extraction of refactoring techniques was conducted over three iterations, with the third iteration indicating that a saturation point had been reached. First, 150 labels were identified to represent various refactoring techniques during the labeling process. Subsequently, the first categorization phase resulted in the identification of 47 distinct categories. Finally, the second categorization phase led to the identification of 15 strategies that practitioners employ to address AITD issues through refactoring. In the first iteration of this process, we extracted the main keywords that highlighted potential solutions to address the problem. For instance, a participant (P37) proposed a solution for *Undeclared Consumers*, stating: "Well, this is quite difficult because when there are unexpected users, it is very difficult to cut their access, and you need to talk with different departments. If it was OK, you would need a strong access system to avoid having this situation in the future, and only users with appropriate access can consume the info." From the participant's comment, we extracted the solution and labeled it as "Build an Access Control System." In the first categorization step, we simplified labels and grouped those denoting the same solution. For example, labels like "Build an Access Control system", "Restrict access of consumers", and "Close Connection to consumers" were grouped under the label "Close access of consumers". Finally, the second categorization iteration allowed the clustering of all the similar refactoring techniques. For example, techniques related to the splitting of a big component into smaller ones (*e.g.*, "Extract stage", "Extract class", "Extract method") were grouped into a single category named "Extract components". In summary, the three iterations of the content analysis allowed us to systematically extract and categorize refactoring strategies practitioners use to address AITD issues, resulting in a structured set of insights.

Fig. 9 collects all the labels used to represent a specific refactoring technique for each AITD issue. Each participant reported the technique used to face the AITD encountered, classified as *Ad Hoc Refactoring*, if the refactoring techniques involve manual code analysis to find an ad hoc solution to solve the issue. Otherwise, if the participant reported not using any technique to refactor the specified issue, the answer is classified as *Not Refactored*. For almost all AITD issues, most answers report the uses of *Ad Hoc Refactoring* (107 labels) and *Not Refactored* (87 labels). These results highlight the practitioners' criticality when facing these kinds of issues and the need to find an automatic technique to support developers during the refactoring activities. Additionally, a good part of the participants highlighted the use of *Extract Components* to face three AITD issues (*Deep God File*, *Jumbled Model Architecture*, and *Pipeline Jungle*). Therefore, extracting a component could help define the modules used to build an AI model and increase the understandability of the whole pipeline. In this context, a participant (P34) highlights the need to isolate the components of each stage of the AI, specifying that: *"[...] Ensuring the five pipeline stages are written in adequately segregated code for troubleshooting purposes."* Regarding *Glue Code* and *Scattered Use of ML Libraries* issues, most of the participants denoted to perform an optimization of dependencies and libraries. In detail, a participant (P6) claimed: *"I would try to eliminate as many libraries as I could and gain a deeper understanding of the crucial ones that had to stay."*. Moreover, the set of answers for the refactoring techniques of *Undeclared Consumers* includes using a component that allows closing the access of consumers, reported by nine participants. To improve the reliability of the consumers that access the system, participants proposed implementing an access system that tracks the information about the application that accesses the model and restricts access only to registered users. Finally, the most reported technique for *Unwanted Debugging Code* is *Remove Debugging Code*. The ease of applying this refactoring technique is an additive hint that highlights the low severity of this AITD issue. Going deeper, analyzing the familiarity level of the participants that reported the stated refactoring techniques, almost all the two groups reported the same techniques. The most reported specific techniques for addressing AITD issues, such as *Dependency Optimization* for *Glue Code* and *Extract Components* for *Deep God File* are reported by participants of both groups similarly. A key difference between the groups is related to *Correction Cascades* issue. While most of the participants of AITD familiar practitioners proposed to perform ad hoc refactoring to address the issue or apply the use of the inheritance, the technique related to restructuring the dependencies is almost reported by AITD unfamiliar practitioners. Finally, automatic tools for addressing *Jumbled Model Architecture* and *Pipeline Jungle* are exclusively denoted by AITD familiar practitioners.

Figs. 10(a) and 10(b) show the effort participants consider required to identify and refactor AITDs. In general, the participants' assessments



(a) Effort depicted by the participant to identify AITD.

(b) Effort depicted by the participant to refactor AITD.

Fig. 10. Identification and refactoring effort of the AITD issues.

align with the reported severity, with the majority of AITDs demanding substantial effort for both identification and refactoring, excluding *Unwanted Debugging Code*, *Glue Code*, and *Correction Cascades*.

Taking a closer look, the identification of *Undeclared Consumers*, which involves interaction with other applications, proved to be exceptionally challenging (14 participants classified the effort as "very high" and 11 as "high"). Additionally, AITDs closely tied to the architecture and AI pipeline, such as *Jumbled Model Architecture* and *Pipeline Jungle*, exhibited higher levels of effort required for their identification. This finding is especially confirmed by *AITD familiar practitioners*, in which all participants of this group confirmed at least a medium identification effort for *Jumbled Model Architecture* and *Undeclared Consumers* and only two participants from the group of 20 *AITD familiar practitioners* confirmed a low effort for identifying *Pipeline Jungle*.

Regarding the refactoring effort, nearly all AITDs were considered to demand a high level of effort by the majority of participants, particularly the *Deep God File* (with nine participants denoting "very high" effort and 14 participants denoting "high" effort). From the two familiarity level groups, this AITD issue is reported to require high effort, particularly by *AITD familiar practitioners*, where 10 participants denoted at least high effort, and only one denoted "low effort" to refactor it. The *Jumbled Model Architecture* requires high effort to be refactored, as stated by participants (with 11 participants characterizing "very high" effort and nine participants denoting "high" effort). In this case, the refactoring of this AITD issue is reported to require high effort by participants in both familiarity-based groups. Lastly, while the distribution of answers concerning the identification effort generally aligns with the refactoring one, for *Deep God File*, many participants deemed the refactoring process to require more effort than its identification.

4.6. Causes of technical debt from the practitioners' perception

In the final stage of our study, we asked participants for any additional feedback or concerns they had regarding potential issues that were not explored (as stated in S4D1 of Table 3). We received answers from 12 participants who provided valuable insights into common challenges that AI-enabled systems face, including technical debt resulting from failing to adhere to best practices and maintaining tools/libraries (P4, P32), as well as traditional software systems issues like a lack of standardization, privacy concerns, and time constraints (P5, P26, P41, P43). Interestingly, participants also noted that socio-technical factors can contribute to AITD. Specifically, communication issues or conflicts within teams (P18, P33, P34) were identified as potential causes of suboptimal solutions introduced by practitioners. In addition, incorporating new team members into developing AI-powered systems presents a distinct challenge. It requires specialized skills, which may not be easily accessible among less familiar team members (P47). This skill gap can lead to increased maintenance effort (P52) or the creation of convoluted and unorganized code, commonly called "spaghetti code". This complexity of code can obscure accountability for suboptimal solutions (P53).

5. Discussion and implications

5.1. Prevalence vs severity

Figs. 6(a) and 6(b) provide insights into the prevalence and severity levels of AITDs as AI practitioners perceive. The findings indicate a low prevalence of AITDs while highlighting a high severity. Focusing on the prevalence of AITD issues, there are factors to consider strictly correlated to the actual state of the progress of detection methods used to assess the quality of AI-enabled systems.

Intricately linked to the continuous evolution and increase of complexity of AI-enabled systems, AI practitioners need help assessing system quality. However, AI-enabled systems are often characterized by their complexity and non-linear behavior. Sculley et al. (2015) define in this context the *Changing Anything, Changes Everything* principle (CACE), which emphasizes that even minor alterations or adjustments to an AI-enabled system can have substantial and unpredictable consequences, making it challenging to detect AITDs. Successful identification requires a deep understanding of AI algorithms and models, the specific application domain, the associated data, and all the components of the AI-enabled system. Consequently, the low prevalence observed in this experiment may be partly attributed to the inherent difficulty in detecting AITDs, given the required skills.

While our experiment provides valuable insights into the perceived prevalence levels of AITDs, it is essential to acknowledge the limitations of the detection mechanism employed by the developers. The actual prevalence of AITDs may be higher than reported, but the challenges in identification and awareness could mask their visibility. Therefore, investing in how to develop robust tools, methodologies, and practices is imperative to enhance AITD detection and management and understand the prevalence of AITD issues in AI-enabled systems.

Advanced detection techniques, leveraging automated analysis and AI-driven approaches, can aid in uncovering hidden technical debt in AI-enabled systems. Additionally, fostering a culture of awareness and accountability within the AI development community is crucial. Developers can enhance their ability to identify and address AITD issues effectively by promoting knowledge sharing, collaboration, and continuous learning.

In addition to the low prevalence, the experiment revealed a high severity associated with the identified AITDs. The participants consistently rated the severity of AITDs as significantly elevated, indicating the potential detrimental impact of these debt-related issues on AI-enabled systems. The high severity levels highlight the critical need for

proactive identification and effective management of AITDs to mitigate their adverse consequences and ensure the delivery of robust and dependable AI solutions.

In summary, although the participants reported a low prevalence of AITD issues in AI-enabled systems, other factors, such as unclear management practices and the high severity, suggest a lack of awareness among developers. This outcome highlights possible hidden threats that developers cannot recognize but are dangerous and impactful for AI-enabled systems.

Take-Away Message

Participants highlighted a low prevalence of the AITD issues proposed but a high severity. The actual state of practice of TD management for AI-enabled systems is at a preliminary state, leading to the inability of the participants to recognize AITD issues in their systems. This result stresses the need for extensive research to raise awareness among practitioners about the issues posed by AITDs and increase the quality of AI-enabled systems.

5.2. Impact on quality-related aspects

Figs. 7 and 8 illustrate the set of quality-related aspects that participants affirm to be negatively impacted by the presence of AITD issues. Since the analysis started with a focus on the effect of six quality-related aspects, analyzing the open answers of participants through content analysis, we extracted 18 affected aspects related to system quality. In detail, before the experiment, we considered as possible impacted aspects the understandability, performance, maintainability, evolution, defect-proneness, and coupling. By examining the answers provided by each participant, we identified certain underlying factors that contribute to the impact of *Multiple Language Smell*. These factors include the level of expertise required to handle the technology employed (*i.e.*, familiarity) and the capacity of the system to adjust to changing requirements and environments (*i.e.*, adaptability). From a broader viewpoint, excluding *Unwanted Debugging Code*, all AITDs have a significant effect, at least on a quality-related aspect. This outcome implies educating practitioners towards AITD understanding to ensure high system quality.

Specifically, participants identified interesting issues leveraged by some AITDs, like the impact on the understandability of *Deep God File* and *Jumbled Model Architecture*. Designing and mixing the related AI components in a cobbled way can lead to one of the two AITD issues, and, as stated by the participants, recognizing and understanding each component and its respective goal can be challenging. To handle this issue, practitioners should guarantee a modular design and split the different parts of the architecture into more cohesive components.

Undeclared Consumers is strictly related to system security, and 28 participants confirmed the high impact, highlighting the probability of creating security breaches. When consumers or components interacting with an AI-enabled system utilize its outputs or services without proper authentication or authorization mechanisms, unauthorized access and misuse of sensitive information become potential risks. In this way, consumers can use private information that can be exploited to understand the internal state of a model and perform malicious attacks. Mitigating the impact of *Undeclared Consumers* is crucial to ensuring the security and integrity of AI-enabled systems. These AITDs underscore the importance of solid quality assessment processes in developing and deploying AI-enabled systems. AI practitioners must solve these issues, increase quality, and implement robust security measures.

Take-Away Message

The management of AITDs is a critical aspect to ensure a high quality of AI-enabled systems. AITDs are characterized by their complexity and non-linear behavior and pose challenges to detect and address potential issues. By managing the presented issues, AI practitioners can enhance the quality aspects of AI-enabled systems related to understandability, security, and performance.

5.3. Towards the discovery of new TD management techniques

Table 5 and Fig. 9 provide an overview of the techniques to address the AITDs under analysis. The participants predominantly rely on manual techniques involving in-depth examinations of code and system architecture to identify these issues. Additionally, ad hoc refactoring approaches are suggested by participants. These findings, when considered alongside the reported high effort in Figs. 10(a) and 10(b), shed light on the nascent stage of addressing technical debt in AI-enabled systems. The reliance on manual techniques implies a lack of well-defined and automated approaches tailored explicitly to identifying and refactoring AITD issues.

Nevertheless, the participants offer valuable insights for managing AITDs. For instance, in the case of *Undeclared Consumers*, the participants recognize the challenge of tracking applications that interact with the AI-enabled system and propose restricting consumer access to mitigate potential malicious use of the AI model. This outcome underscores the importance of implementing access control mechanisms to regulate system usage and continuously monitor applications that collect information. Regarding other AITD issues such as *Deep God File*, *Jumbled Model Architecture*, and *Pipeline Jungle*, the participants suggest leveraging refactoring techniques inspired by those used in traditional software, as introduced by Fowler (1999). Specifically, employing established methods like *Extract Method* and *Extract Class* to reduce coupling and enhance component cohesion could prove beneficial in mitigating complexity within the pipeline and architecture of the system.

Finally, from the answers collected for the refactoring of *Pipeline Jungle*, two participants (P39, P40) identified the possibility of using MLOps tools to visualize, monitor, and continuously manage the AI pipeline. This answer highlights the potential of MLOps applied to AI software quality assurance.

The challenge of establishing a flexible, customizable, reusable, and fault-tolerant pipeline is a key focus area in the field of MLOps, as confirmed by Steidl et al. (2023). Given the intricate nature of AI pipelines, which often involve multiple stages and components, it becomes imperative to design pipelines that adapt to evolving requirements, accommodate diverse use cases, and seamlessly handle various data formats and sources. Developing a flexible and customizable pipeline facilitates the smooth integration of new components or modifications to existing ones without disrupting the workflow. However, complex and convoluted *Pipeline Jungle* is a severe obstacle to achieving the ultimate goal of a high-quality pipeline. In this context, the set of available MLOps tools in the state-of-the-practice allows obtaining complete monitoring and management of the AI pipeline (*e.g.*, KubeFlow Pipelines), in which stakeholders can comprehend the whole structure, components, and interactions of the pipeline. Therefore, it is possible to prevent *Pipeline Jungle* by leveraging MLOps practices.

These findings imply the need for further research and development efforts to establish standardized techniques and automated tools designed to address AITDs. Such advancements would help reduce dependency on manual inspections and ad hoc refactoring, enabling practitioners to detect, manage, and mitigate AITDs, ultimately enhancing the overall quality and maintainability of AI-enabled systems.

Take-Away Message

Although the current state of the art in identifying and refactoring AITD issues is still in its early stages, the participants have put forth suggestions that contribute to shaping the direction of future research on the automatic refactoring and management of AITD.

5.4. Relationship with socio-technical aspects

The additional analysis results highlighted that issues affecting socio-technical aspects are a possible cause that could lead developers to introduce AITD. During the study, participants identified several key socio-technical factors that can contribute to introducing AITD in AI projects. One such factor highlighted was poor communication within AI development teams, which can result in conflicts during development. Team members must convey their ideas, expectations, and concerns effectively to avoid misunderstandings and misalignment in AI model design and implementation decisions. Failure to do so can result in suboptimal AI models and algorithms, ultimately contributing to AITD. Another crucial factor identified was the expertise level of AI practitioners involved in the project. Participants emphasized how the experience and knowledge of team members could significantly impact the likelihood of introducing AITD. They noted that introducing a new specific-purpose language or a new particular library that requires high expertise from AI practitioners could increase the risk of introducing suboptimal solutions. Lastly, the study found that AITD issues can impact socio-technical aspects, as highlighted by the participants. The presence of the *Multiple Language Smell* could affect the familiarity level of the developers with the system.

Although these findings were only confirmed by some survey participants, they were supported by another study conducted by Mailach and Siegmund (2023). They identified 17 socio-technical anti-patterns in ML-enabled software development, focusing on organizational and management issues. The authors provided recommendations to overcome these problems, ranging from technical solutions to organizational restructuring. We can denote common factors in our study from the set of anti-patterns that the authors discovered. Specifically, Mailach and Siegmund (2023) classified the situation that can arise between team members with different skill sets as anti-pattern, which is consistent with the conflicts and communication challenges reported by participants. Additionally, our study identified a lack of standardization as a socio-technical anti-pattern, which their study referred to as a "Lack of data science process."

The intricate interplay between AITD issues and socio-technical factors suggests a multifaceted relationship, underscoring the need for comprehensive investigations and holistic solutions to manage AI projects effectively.

Take-Away Message

Socio-technical factors play a significant role in the AI-enabled system's quality and are connected with AITD issues. Critical considerations include poor communication within AI development teams, the expertise level of AI practitioners, and the impact of AITD on socio-technical aspects. The relationship between socio-technical factors and the introduction of AITD underscores the need for more research in this area.

5.5. The road ahead for AITD and software quality for AI

Given our results and the state of software quality for AI, it is necessary to discuss the implications of state-of-the-art techniques on quality assessment, focusing on managing AITD issues. Our results suggest the need for techniques and tools to identify and refactor such issues. Identifying different techniques should reduce the developers' need to apply manual analysis, which requires a high effort and whose success depends on the practitioner's expertise concerning the quality assessment of AI-enabled systems.

While literature, starting from the definition of Sculley et al. (2015) of the several problems that can occur when building an AI-enabled system, further work is needed to improve AITD Management.

Firstly, finding the properties and characteristics of AITD issues can improve the investigation of their presence. A recent study conducted by Costal et al. (2023) identifies a preliminary set of metrics to identify some technical debt issues occurring in AI-enabled systems, also considering *Glue Code* and *Scattered Use of ML Libraries*. Further investigation into the tangible properties and qualitative factors of AITD is necessary to understand and effectively address these issues within AI-enabled systems fully. The combination of the metrics proposed and other aspects related to the quality assessment process and architecture analysis can be used to detect AITD issues more accurately. Therefore, these key elements linked with the characteristics of AITD issues will be aimed to define detection techniques. For instance, to identify *Pipeline Jungle*, practitioners could analyze the pipeline configurations of AI-enabled systems and examine the complexity of data flows. They could investigate factors such as the number of stages, the interdependencies between stages, and the presence of redundant or unnecessary processing steps. Following this identification strategy, practitioners could systematically identify instances of *Pipeline Jungle* and assess the associated technical debt. Finally, researchers should provide datasets encompassing various AITD issues to advance the existing body of knowledge. Such datasets would be valuable for developing and evaluating AITD management techniques. They should include real-world examples of AITD issues encountered in different AI-enabled systems and relevant contextual information.

By addressing these steps, we could eventually obtain more reliable, maintainable, and robust AI applications.

5.6. Implications

From a practitioner's perspective, the research implications and the state of software quality for AI significantly impact how AI-enabled systems are developed and maintained. With the identification and refactoring of AITD issues relying heavily on manual analysis by practitioners, it is crucial to invest in training and skill development programs. Organizations should provide resources and opportunities for AI practitioners to enhance their expertise in quality assessment techniques and AI-enabled system management. Given the complex and interdisciplinary nature of AI-enabled systems and collaborative development practices, code reviews become even more critical. Potential AITD issues could be identified and addressed early in development by involving multiple experts from different domains.

Industry bodies, research organizations, and practitioners should collaborate to define and promote guidelines and standards for identifying, measuring, and addressing AITD in AI-enabled systems. These best practices should be regularly updated to keep pace with the evolving technology landscape, allowing for the continuous evolution of the maturity of the AI quality assurance process. Continuous monitoring and maintenance are necessary to ensure new AITD issues do not arise during the system evolution. Regular audits, reviews, and assessments should be conducted to keep the system quality in check and minimize the accumulation of technical debt.

6. Threats to validity

Several design decisions might have threatened the validity of our work. This section discusses our choices and how we attempted to mitigate the corresponding threats to validity.

External validity. One of the primary issues that could potentially threaten external validity is selection bias in the recruitment process. Recruiting participants through social media or specific recruitment platforms may inadvertently result in gathering data from individuals who lack expertise in the domain of interest. To address this concern, we implemented a prescreening process to evaluate the eligibility of each participant before conducting the survey. This prescreening involved applying filters based on the practitioners' experience to ensure that only relevant and knowledgeable participants were included.

However, this design choice introduced another potential threat to the external validity of the study: the small sample size of 53 participants. This limited number of participants might not provide a comprehensive representation of the population of interest, impacting the generalizability of the findings. Nevertheless, opting for a smaller sample size was driven by the inherent difficulty in finding AI practitioners with expertise in the specific domain. Despite starting with a pool of 500 potential participants, eligibility criteria were carefully set to ensure that only individuals with relevant experience in software engineering, artificial intelligence, model development, and model deployment were included. By choosing a smaller sample size, the focus was placed on obtaining higher-quality answers that are more applicable and valuable within the specific context of the survey. Despite the potential limitations in generalizability, this approach aimed to prioritize the quality and relevance of the collected data. It was essential to balance the sample size and the depth of knowledge possessed by participants in the domain of interest.

In conclusion, while the representation of the population might not be fully reached to generalize the findings based on the practitioners' perspective, these findings serve as valuable hints that suggest the need for further investigation into these quality issues. The study outcomes provide insights that hold significance within the specific context of the survey and lay the foundation for future research in this domain.

Internal validity. Another significant concern is the participants' familiarity with AITD issues. Despite deliberately selecting participants with backgrounds in both Software Engineering and Artificial Intelligence, there remains the possibility that they may need more explicit knowledge of AITD issue definitions. To address this potential limitation, we employed the vignette experiment design method, which simplifies the presentation of AITD issues within scenarios commonly encountered during the design and development of AI pipelines. Participants were provided with contextualized situations that helped them recognize and evaluate AITD problems based on their experiential knowledge, even without detailed familiarity with AITD terms. This approach enabled participants to effectively identify and assess AITD issues, drawing upon their experience. Therefore, participants can provide answers based on a common understanding of the issues presented through the vignette.

As for AITD issues, another significant threat could be related to the definition of AI that participants assumed. Recognizing that participants could have a different definition of AI based on their experience, the survey could involve different points of view, including participants unrelated to AI. To mitigate this potential concern, we conducted a prescreening phase to ensure all participants had a common baseline understanding of AI, comprehensive general skills, and AI-specific skills. Filtering the participants based on their answers, combined with a competence and attention check question, ensured that all the participants had a sufficient understanding of AI and were eligible for the survey. While a baseline is ensured among all the participants, it is important to acknowledge that the perceptions collected by the participants can vary depending on their experience in AI and their definitions.

Another potential threat to the internal validity of this study relates to the strategy used for assigning AITD scenarios to each participant. Traditional strategies, such as within-subject or between-subject approaches, can introduce certain limitations and threats. On the one hand, the within-subject approach can significantly increase the time required to complete the survey, leading to participant fatigue and impacting the quality of their answers. On the other hand, the between-subject approach may result in a limited number of questions answered by each participant, potentially reducing the overall depth and amount of the data collected. To address these concerns, we opted for a mixed-design approach, which allows for a balanced trade-off between the survey execution time and the number of questions each participant answers. By adopting the mixed-subject design, we aimed to ensure that participants had a manageable survey completion time while still obtaining sufficient data from each participant. This solution enables us to capture broader insights while mitigating potential threats related to survey duration and participant burden.

In utilizing Prolific as a participant recruitment platform, an internal validity concern is the potential for incentive bias. Incentive bias may arise when participants on Prolific are motivated to complete surveys solely to receive compensation rather than genuinely engaging with the study. The financial incentive offered by Prolific could lead some individuals to rush through the survey or provide answers without thoughtful consideration, jeopardizing the quality and reliability of the data collected. To mitigate incentive bias, we introduced attention check questions during the prescreening phase, limiting the number of careless answers.

Another significant concern often arising when conducting questionnaire surveys is the potential use of external sources, which becomes more critical with the widespread availability of AI language models like ChatGPT.[5] Participants may collect and replicate the suggestions provided by ChatGPT when responding to our survey questions. Despite having conducted participant recruitment and data collection before the availability of ChatGPT, we must acknowledge the possibility that participants could have used other similar models that were accessible at the time of the study or turned to internet sources for information. We have taken several measures to address this threat of external source influence. We provided clear instructions to participants, explicitly stating that using external sources, including AI language models and internet browsing, is prohibited. We aimed to encourage genuine, independent answers that accurately reflect participants' knowledge and opinions. Furthermore, we have implemented validation and attention checks within the survey to assess the authenticity of participants' answers. These checks allow us to identify any potential AI-generated or internet-sourced answers that might affect the integrity of the data.

Construct validity. A possible threat to the construct validity is related to the experience bias. This bias arises due to differences in experience and knowledge levels between researchers conducting the pilot test and industry practitioners who will participate in the main survey. The researchers' expertise and familiarity with AI systems may inadvertently influence the framing of survey questions, leading to potential mismatches with the real-world experiences of industry practitioners. To address this threat, a software engineer with experience in AI system development participated in the pilot test, providing valuable feedback from an industry perspective. This feedback informed refinements to the survey instrument, ensuring its relevance and clarity for industry practitioners.

Another potential threat we encountered was related to hypothesis guessing of research questions due to the comprehensive definition of AITD issues. Participants might have assumed that every issue presented should be regarded as problematic and directly responded in the affirmative. To mitigate this threat, we took proactive steps by broadening the definition of AITD issues to be more encompassing, representing them as vignettes and referring to them as *"situations"*. Additionally, we incorporated open-ended questions, enabling participants to provide detailed answers that allowed us to filter answers based on thoughtful consideration and genuine concerns. This approach ensured that

[5] ChatGPT: https://openai.com/chatgpt

we obtained more nuanced and reliable data, minimizing the impact of hypothesis guessing on the study's internal validity. By adopting these strategies, we aimed to foster a deeper understanding among participants while preserving the integrity of the research findings.

Conclusion validity. A potential threat to the conclusion validity is related to the reliability of measures used to analyze participants' answers to address our research questions. We have implemented a strategy to enhance data collection to address this threat. In addition to the multiple-choice questions, we have incorporated open-ended questions that complement and support the answers provided in the multiple-choice format. By including open-ended questions, we encouraged participants to provide more in-depth and nuanced answers, allowing us to capture additional insights and perspectives that may not be fully represented in the closed-ended options. This approach helped ensure a richer and more comprehensive data set, contributing to a more reliable analysis of participants' attitudes, opinions, and experiences regarding AITD issues.

7. Conclusion

In this paper, we conducted a survey study with 53 practitioners to investigate the prevalence, severity, impact, and mitigation strategies applied to deal with nine distinct AITD issues. The study revealed multiple insights into how practitioners cope with the emergence of AI technical debt. Practitioners perceive those issues as highly severe and impactful despite being unaware of how prevalent these debt issues are in practice; yet, the automated support available to mitigate them is limited and, indeed, they often rely on manual inspections and ad-hoc refactoring approaches. The results of our work let us draw several considerations and implications for future research on AI technical debt, other than presenting a call for novel, automated methods that support practitioners in identifying and refactoring AI technical debt.

The output of this work represents the input of our future research agenda, which will be focused on addressing the practitioners' needs that emerged from our analysis. On the one hand, we aim to design further empirical investigations into how AI technical debt is perceived and managed in practice. On the other hand, we aim to develop novel instruments to automatically detect and possibly refactor the technical debt affecting the source code and architectures. Additionally, this study initiated an exploration into a preliminary set of AITD issues. Our findings, while focused, open avenues for investigation into diverse types of AITD issues, particularly those related to socio-technical factors affecting practitioners' familiarity and communication issues.

CRediT authorship contribution statement

Gilberto Recupito: Writing – original draft, Visualization, Validation, Investigation, Formal analysis, Data curation, Conceptualization. **Fabiano Pecorelli:** Writing – original draft, Visualization, Validation, Investigation, Formal analysis, Data curation, Conceptualization. **Gemma Catolino:** Writing – review & editing, Supervision, Resources, Data curation. **Valentina Lenarduzzi:** Writing – review & editing, Supervision, Resources. **Davide Taibi:** Writing – review & editing, Supervision, Resources. **Dario Di Nucci:** Writing – review & editing, Supervision, Resources, Methodology. **Fabio Palomba:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by the *QualAI* national research project, which have been funded by the MUR under the PRIN 2022 program (Contracts 2022B3BP5S).

Data availability statement

This paper includes data as electronic supplementary material. Datasets generated and analyzed in the context of this study, raw results, and detailed plots, as well as additional resources useful for reproducing our research, are available in the online appendix of this paper: https://doi.org/10.6084/m9.figshare.24030456

References

Ala-Pietilä, P., Bonnet, Y., Bergmann, U., Bielikova, M., Bonefeld-Dahl, C., Bauer, W., Bouarfa, L., Chatila, R., Coeckelbergh, M., Dignum, V., et al., 2020. The Assessment List for Trustworthy Artificial Intelligence (ALTAI). European Commission.

Alahdab, M., Çalıklı, G., 2019. Empirical analysis of hidden technical debt patterns in machine learning software. In: Product-Focused Software Process Improvement: 20th International Conference, PROFES 2019, Barcelona, Spain, November 27–29, 2019, Proceedings 20. Springer, pp. 195–202.

Alves, N.S., Mendes, T.S., De Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C., 2016. Identification and management of technical debt: A systematic mapping study. Inf. Softw. Technol. 70, 100–121.

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T., 2019. Software engineering for machine learning: A case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice. ICSE-SEIP, IEEE, pp. 291–300.

Atzmüller, C., Steiner, P., 2006. Experimental vignette designs for factorial surveys. Kolner Z. Soziol. Sozialpsychol. 58, 117–146+200.

Basili, V.R., Caldiera, G., Rombach, H.D., 1994. The goal question metric approach. In: Encyclopedia of Software Engineering. pp. 528–532.

Bogner, J., Verdecchia, R., Gerostathopoulos, I., 2021. Characterizing technical debt and antipatterns in AI-based systems: A systematic mapping study. In: 2021 IEEE/ACM International Conference on Technical Debt. TechDebt, pp. 64–73.

Bosch, J., Olsson, H.H., Crnkovic, I., 2021. Engineering ai systems: A research agenda. In: Artificial Intelligence Paradigms for Smart Cyber-Physical Systems. IGI global, pp. 1–19.

Breck, E., Cai, S., Nielsen, E., Salib, M., Sculley, D., 2017. The ML test score: A rubric for ML production readiness and technical debt reduction. In: 2017 IEEE International Conference on Big Data. Big Data, IEEE, pp. 1123–1132.

Briand, L., Emam, K., Morasca, S., 1996. On the application of measurement theory in software engineering. Empir. Softw. Eng. 1, 61–88.

Chen, T.-H., Shang, W., Jiang, Z.M., Hassan, A.E., Nasser, M., Flora, P., 2014. Detecting performance anti-patterns for applications developed using object-relational mapping. In: Proceedings of the 36th International Conference on Software Engineering. ICSE 2014, Association for Computing Machinery, New York, NY, USA, pp. 1001–1012.

Costal, D., Gómez, C., Martínez-Fernández, S., 2023. Metrics for code smells of ML pipelines. In: International Conference on Product-Focused Software Process Improvement. Springer, pp. 3–9.

Cunningham, W., 1992. The WyCash portfolio management system. In: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, Vol. Part F129621. OOPSLA, pp. 29–30.

Easterbrook, S., Singer, J., Storey, M.-A., Damian, D., 2008. Selecting empirical methods for software engineering research. In: Guide to Advanced Empirical Software Engineering. Springer, pp. 285–311.

Foidl, H., Felderer, M., 2019. Risk-based data validation in machine learning-based software systems. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation. pp. 13–18.

Foidl, H., Felderer, M., Ramler, R., 2022. Data smells: categories, causes and consequences, and detection of suspicious data in AI-based systems. In: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI. pp. 229–239.

Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA.

Gesi, J., Liu, S., Li, J., Ahmed, I., Nagappan, N., Lo, D., de Almeida, E.S., Kochhar, P.S., Bao, L., 2022. Code smells in machine learning systems.

Gezici, B., Tarhan, A.K., 2022. Systematic literature review on software quality for AI-based software. Empir. Softw. Eng. 27 (3), 66.

Golendukhina, V., Lenarduzzi, V., Felderer, M., 2022. What is software quality for AI engineers? Towards a thinning of the fog. In: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI. pp. 1–9.

Holvitie, J., Licorish, S.A., Spínola, R.O., Hyrynsalmi, S., MacDonell, S.G., Mendes, T.S., Buchan, J., Leppänen, V., 2018. Technical debt and agile software development practices and processes: An industry practitioner survey. Inf. Softw. Technol. 96, 141–160.

ISO/IEC, 2023a. Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality Model for AI systems. International Standard ISO/IEC 25059:2023, International Organization for Standardization and International Electrotechnical Commission.

ISO/IEC, 2023b. Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality model. International Standard ISO/IEC 25010:2011, International Organization for Standardization and International Electrotechnical Commission.

Jaafar, F., Guéhéneuc, Y.-G., Hamel, S., Khomh, F., 2013. Mining the relationship between anti-patterns dependencies and fault-proneness. In: 2013 20th Working Conference on Reverse Engineering. WCRE, pp. 351–360.

Jebnoun, H., Ben Braiek, H., Rahman, M.M., Khomh, F., 2020. The scent of deep learning code: An empirical study. In: Proceedings of the 17th International Conference on Mining Software Repositories. pp. 420–430.

Khomh, F., Penta, M.D., Guéhéneuc, Y.-G., Antoniol, G., 2012. An exploratory study of the impact of antipatterns on class change- and fault-proneness. Empir. Softw. Eng. 17 (3), 243–275.

Kitchenham, B.A., Pfleeger, S.L., 2002a. Principles of survey research part 2: designing a survey. ACM SIGSOFT Softw. Eng. Notes 27 (1), 18–20.

Kitchenham, B., Pfleeger, S., 2002b. Principles of survey research part 3: constructing a survey instrument. ACM SIGSOFT Softw. Eng. Notes 27, 20–24.

Kuwajima, H., Yasuoka, H., Nakae, T., 2020. Engineering problems in machine learning systems. Mach. Learn. 109 (5), 1103–1126.

Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., Arcelli Fontana, F., 2021a. A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. J. Syst. Softw. 171, 110827.

Lenarduzzi, V., Lomio, F., Moreschini, S., Taibi, D., Tamburri, D.A., 2021b. Software quality for AI: Where we are now? In: Software Quality: Future Perspectives on Software Engineering Quality: 13th International Conference, SWQD 2021, Vienna, Austria, January 19–21, 2021, Proceedings 13. Springer, pp. 43–53.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220.

Liu, J., Huang, Q., Xia, X., Shihab, E., Lo, D., Li, S., 2020. Is using deep learning frameworks free? characterizing technical debt in deep learning frameworks. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society. pp. 1–10.

Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H., Crnkovic, I., 2019. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In: Agile Processes in Software Engineering and Extreme Programming: 20th International Conference, XP 2019, MontrÉAl, QC, Canada, May 21–25, 2019, Proceedings 20. Springer International Publishing, pp. 227–243.

Maiga, A., Ali, N., Bhattacharya, N., Sabané, A., Guéhéneuc, Y.-G., Antoniol, G., Aïmeur, E., 2012. Support vector machines for anti-pattern detection. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ASE '12, Association for Computing Machinery, New York, NY, USA, pp. 278–281.

Mailach, A., Siegmund, N., 2023. Socio-technical anti-patterns in building ML-enabled software: Insights from leaders on the forefront. In: 2023 IEEE/ACM 45th International Conference on Software Engineering. ICSE, IEEE, pp. 690–702.

Malakuti, S., Borrison, R., Kotriwala, A., Kloepper, B., Nordlund, E., Ronnberg, K., 2021. An integrated platform for multi-model digital twins. In: Proceedings of the 11th International Conference on the Internet of Things. pp. 9–16.

Marinescu, R., 2004. Detection strategies: metrics-based rules for detecting design flaws. In: 20th IEEE International Conference on Software Maintenance, 2004. Proceedings. pp. 350–359.

Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., Vollmer, A.M., Wagner, S., 2022. Software engineering for AI-based systems: A survey. ACM Trans. Softw. Eng. Methodol. 31 (2).

Matthews, J., 2020. Patterns and anti-patterns, principles and pitfalls: accountability and transparency in AI. AI Mag. 41 (1), 82–89.

Mikkonen, T., Nurminen, J.K., Raatikainen, M., Fronza, I., Mäkitalo, N., Männistö, T., 2021. Is machine learning software just software: A maintainability view. In: Software Quality: Future Perspectives on Software Engineering Quality: 13th International Conference, SWQD 2021, Vienna, Austria, January 19–21, 2021, Proceedings 13. Springer, pp. 94–105.

Moha, N., Gueheneuc, Y.-G., Duchien, L., Le Meur, A.-F., 2010. DECOR: A method for the specification and detection of code and design smells. IEEE Trans. Softw. Eng. 36 (1), 20–36.

Munappy, A., Bosch, J., Olsson, H.H., Arpteg, A., Brinne, B., 2019. Data management challenges for deep learning. In: 2019 45th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 140–147.

Nemoto, T., Beglar, D., 2014. Likert-scale questionnaires. In: JALT 2013 Conference Proceedings. pp. 1–8.

Olbrich, S., Cruzes, D.S., Basili, V., Zazworka, N., 2009. The evolution and impact of code smells: A case study of two open source systems. In: The Evolution and Impact of Code Smells: A Case Study of Two Open Source Systems. pp. 390–400, Cited by: 179; All Open Access, Green Open Access.

Ozkaya, I., 2020. What is really different in engineering AI-enabled systems? IEEE Softw. 37 (4), 3–6.

Palomba, F., Bavota, G., Penta, M.D., Oliveto, R., Poshyvanyk, D., De Lucia, A., 2015. Mining version histories for detecting code smells. IEEE Trans. Softw. Eng. 41 (5), 462–489.

Palomba, F., Panichella, A., De Lucia, A., Oliveto, R., Zaidman, A., 2016. A textual-based technique for smell detection. In: 2016 IEEE 24th International Conference on Program Comprehension. ICPC, pp. 1–10.

Pecorelli, F., Palomba, F., Di Nucci, D., De Lucia, A., 2019. Comparing heuristic and machine learning approaches for metric-based code smell detection. In: 2019 IEEE/ACM 27th International Conference on Program Comprehension. ICPC, IEEE, pp. 93–104.

Ravì, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., Yang, G.-Z., 2016. Deep learning for health informatics. IEEE J. Biomed. Health Inform. 21 (1), 4–21.

Rech, J., Althoff, K.-D., 2004. Artificial intelligence and software engineering: Status and future trends. KI 18 (3), 5–11.

Reid, B., Wagner, M., d'Amorim, M., Treude, C., 2022. Software engineering user study recruitment on prolific: An experience report.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., Dennison, D., 2015. Hidden technical debt in machine learning systems. Adv. Neural Inf. Process. Syst. 28.

Seaman, C., Guo, Y., 2011. Measuring and monitoring technical debt. In: Advances in Computers, vol. 82, Elsevier, pp. 25–46.

Shivashankar, K., Martini, A., 2022. Maintainability challenges in ML: A systematic literature review. In: 2022 48th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, pp. 60–67.

Shome, A., Cruz, L., Van Deursen, A., 2022. Data smells in public datasets. In: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI. pp. 205–216.

Siebert, J., Joeckel, L., Heidrich, J., Nakamichi, K., Ohashi, K., Namba, I., Yamamoto, R., Aoyama, M., 2020. Towards guidelines for assessing qualities of machine learning systems. In: Shepperd, M., Brito e Abreu, F., Rodrigues da Silva, A., Pérez-Castillo, R. (Eds.), Quality of Information and Communications Technology. Springer International Publishing, Cham, pp. 17–31.

Steidl, M., Felderer, M., Ramler, R., 2023. The pipeline for the continuous development of artificial intelligence models—Current state of research and practice. J. Syst. Softw. 199, 111615.

Sun, X., Zhou, T., Li, G., Hu, J., Yang, H., Li, B., 2017. An empirical study on real bugs for machine learning programs. In: 2017 24th Asia-Pacific Software Engineering Conference. APSEC, pp. 348–357.

Tang, Y., Khatchadourian, R., Bagherzadeh, M., Singh, R., Stewart, A., Raja, A., 2021. An empirical study of refactorings and technical debt in machine learning systems. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, pp. 238–250.

Thung, F., Wang, S., Lo, D., Jiang, L., 2012. An empirical study of bugs in machine learning systems. In: 2012 IEEE 23rd International Symposium on Software Reliability Engineering. pp. 271–280.

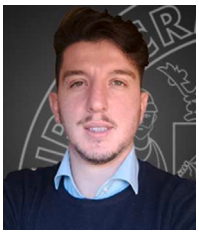
Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. J. Syst. Softw. 86 (6), 1498–1516.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D., 2015. When and why your code starts to smell bad. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1. pp. 403–414.

White, M.D., Marsh, E.E., 2006. Content analysis: A flexible methodology. Lib. Trends 55 (1), 22–45.

Zhang, S., Yao, L., Sun, A., Tay, Y., 2019. Deep learning based recommender system: A survey and new perspectives. ACM Comput. Surv. 52 (1), 1–38.

Zhou, J., Chen, F., 2018. Human and Machine Learning. Springer.



Gilberto Recupito is a Ph.D. Student at the Software Engineering (SeSa) Lab of the University of Salerno, Italy, under the supervision of Professor Fabio Palomba and Professor Dario Di Nucci. He received a bachelor's and master's degree in computer science from the University of Salerno, Italy. His research interests are centered around Software Engineering for Artificial Intelligence (SE4AI). This includes technical debt in AI-enabled systems, Machine Learning Operations (MLOps), Mining Software Repositories (MSR), and Empirical Software Engineering.

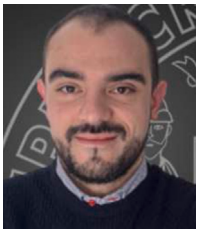
Fabiano Pecorelli, is a Researcher at the Software Engineering (SeSa) Lab of the University of Salerno, Italy. He was a Postdoctoral Researcher at the Jheronimus Academy of Data Engineering (JADE) lab at Jheronimus Academy of Data Science (JADS), The Netherlands He earned a magna cum laude European Ph.D. in Computer Science from the University of Salerno, Italy, supervised by Prof. Andrea De Lucia. His research interests span Software Code and Test Quality, Mining Software Repositories, Software Maintenance, Empirical Software Engineering, and Quantum Software Engineering. He actively referees to esteemed software engineering journals such as TOSEM, EMSE, and JSS.

Gemma Catolino is an Assistant Professor at the Software Engineering (SeSa) Lab (within the Department of Computer Science) of the University of Salerno. In 2020, she received the European Ph.D. Degree from the University of Salerno, advised by Prof. Filomena Ferrucci. She received (magna cum laude) the Master's Degree in Management and Information Technology from the University of Salerno (Italy) in 2016 defending a thesis on Software Quality Metrics, advised by Prof. Filomena Ferrucci. She got a Bachelor's Degree in Computer Science from the University of Molise in 2014 defending a thesis on Software Program Comprehension. Her research interests include human factors in software maintenance and evolution, empirical software engineering, source code quality, and mining software repositories.

Valentina Lenarduzzi is an Associate Professor (tenure track) at the University of Oulu (Finland). Her research activities are related to contemporary software development practices and methodologies, including data analysis in software engineering, software quality, software maintenance, and evolution, focusing on Technical Debt and code and architectural smells. She got a Ph.D. in Computer Science in 2015 and was an Assistant Professor (tenure track) at the University of Oulu (Finland), a postdoctoral researcher at Tampere University (Finland), LUT University (Finland), and the Free University of Bozen-Bolzano, (Italy). Moreover, she was a visiting researcher at the University of Kaiserslautern (TUK) and the Fraunhofer Institute for Experimental Software Engineering IESE (Germany). She served as a program committee member of various international conferences (*e.g.*, ICPC, ICSME, ESEM), and for various international journals (*e.g.*, TSE, EMSE, JSS, IST) in the field of software engineering. She has been the program co-chair of several conferences. Dr. Lenarduzzi is recognized by the Journal of Systems and Software (JSS) as one of the most active SE researchers in top-quality journals from 2013 to 2020.

Davide Taibi is a Full Professor at the University of Oulu, Finland, and head of the M3S Cloud Research Group. His research focuses on empirical software engineering applied to cloud-native systems, focusing on migrating from monolithic to cloud-native applications. He investigates techniques for cloud-native applications and identifies cloud-native-specific patterns and anti-patterns. He has been a member of the International Software Engineering Network (ISERN), since 2018. He was an Assistant Professor at the Free University of Bozen/Bolzano (2015–2017) and a Postdoctoral Research Fellow at the Technical University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering (IESE) (2013–2014).

Dario Di Nucci is an Assistant Professor at the Software Engineering (SeSa) Lab of the University of Salerno, Italy. His research focuses on empirical software engineering, employing machine learning, search-based algorithms, and software repository mining in software maintenance, evolution, and testing. He is an active member of the IEEE Conference Activities Committee, an editor for Elsevier's Journal of Systems and Software (JSS) and Science of Computer Programming (SCICO), and a technical committee member of OASIS TOSCA. He is a program committee member for international conferences and referees for esteemed journals, receiving Distinguished/Outstanding Reviewer Awards.



Fabio Palomba is an Assistant Professor at the Software Engineering (SeSa) Lab of the University of Salerno, Italy, holding a European Ph.D. in Management & Information Technology. His research focuses on software maintenance, evolution, empirical software engineering, source code quality, and mining software repositories. He received prestigious awards for his work, including Best Ph.D. Thesis by IEEE Computer Society and Distinguished Paper Awards at various conferences. He is a ICPC's Steering Committee member and has chaired program committees for several conferences. He serves on editorial boards for prominent journals and has earned 12 Distinguished/Outstanding Reviewer Awards for his reviewing contributions.