Graphical Abstract

Algorithm Debt in Machine and Deep Learning: Insights from a Mixed-Methods Study of Practitioners

Emmanuel Iko-Ojo Simon, Chirath Hettiarachchi, Fatemeh Fard, Alex Potanin, Hanna Suominen

Highlights

Algorithm Debt in Machine and Deep Learning: Insights from a Mixed-Methods Study of Practitioners

Emmanuel Iko-Ojo Simon, Chirath Hettiarachchi, Fatemeh Fard, Alex Potanin, Hanna Suominen

- Characterised AD in ML/DL systems with lack of developer knowledge as a major cause of AD.
- Identified scalability as major effect of AD.
- Proposed mitigation strategies to AD.

Algorithm Debt in Machine and Deep Learning: Insights from a Mixed-Methods Study of Practitioners

Emmanuel Iko-Ojo Simon[a], Chirath Hettiarachchi[a], Fatemeh Fard[b], Alex Potanin[a], Hanna Suominen[a,c,d]

[a] *Australian National University (ANU), ANU School of Computing, Australia*
[b] *British Columbia University, Department of Computer Science, Canada*
[c] *ANU School of Psychology and Medicine,*
[d] *University of Turku, Department of Computing, Finland*

Abstract

Background. Algorithm Debt (AD), is a Technical Debt type that arises from suboptimal algorithmic choices, impacting system reliability. Despite the impact of AD affecting scalability and model degradation and the rapid evolution of Machine and Deep Learning (ML/DL) systems, it remains underexplored.

Objective. To bridge this gap, in this study we investigated the causes, effects, and mitigation strategies of AD in ML/DL systems.

Methods. Using concurrent mixed-methods, we analysed 65 questionnaires and conducted 21 semi-structured interviews with ML/DL practitioners.

Results. We identified 13 causes, 8 effects, and 15 mitigation strategies of AD in ML/DL systems. Notably we found a limited awareness of AD among practitioners with causes varying by roles. We notably identified the lack of developer knowledge of ML/DL algorithms (72.3%) as the major cause of AD. Poor model scalability (73.8%) was the dominant effect, with systemic testing (70.8%) as key mitigation strategy.

Conclusion. This study confirmed AD as a distinct TD type, and offered practical guide to practitioners on how to mitigate AD in practice. Future research should focus on enhancing practitioner expertise and creation of automated tools for the detection of AD to enhance the long term reliability of ML/DL systems.

Keywords: Technical Debt, Algorithm Debt, Software Maintenance, ML, DL, Mixed-methods

1. Introduction

Machine Learning (ML) and Deep Learning (DL) systems (i.e., systems incorporating any ML/DL algorithms) have become integral to a wide range of applications and domains. These applications range from Natural Language Processing (NLP) [55] to autonomous vehicles [20], driving innovation across different industries such as healthcare and industrial processes [45]. However, the adoption of these ML/DL applications have introduced challenges in maintaining their reliability (e.g., performance consistency) and sustainability (e.g., resource efficiency) over time.

One of such challenges in the maintainability of ML/DL systems is Technical Debt (TD) [15]. TD refers to expedient design decisions that are beneficial in the short term but introduce inefficiencies in the long term [26]. While TD has been extensively studied in traditional software systems, its occurrence in ML/DL systems is unique [43]. This is due to the additional learning components and development cycle of ML/DL systems (e.g., model training, data pre-processing).

This unique nature of ML/DL makes developers incur TD types such as Algorithm Debt (AD). AD is a form of TD that arises from suboptimal algorithmic choices in the design and implementation of systems leading to degradation of systems and poor model scalability [28, 44]. For example, inefficient algorithmic choices in ML/DL systems development especially during model training can result in prolonged training times and limited scalability, ultimately impacting operational performance as the system evolves.

Several studies published in the 2020s have explored different aspects of TD in ML/DL systems. For example, Pepe et al. [38] analysed Self-admitted TD in DL systems and categorised those specific to DL into categories including DL models, technology (hardware and API), and suboptimal DL processes such as model usage or configuration. Also, Ximenes [53] identified a list of potential factors contributing to TD in the source code of ML/DL systems. They identified data processing as the most critical factor. They also identified issues related to model creation and training as issues leading to TD in ML/DL systems. Sklavenitis and Kalles [46], in their work to identify TD types in ML systems, conducted a survey and categorised

TD types in ML/DL systems into AD, architecture, Code and Configuration Debt. Recupito et al. [39] further investigated ML-specific TD to address issues related to managing ML-code and Architecture Debt in ML/DL systems through the use of a survey. Despite these studies on TD focusing on code and data, the causes, effects, and mitigation strategies of AD in ML/DL systems remain underexplored.

To address this gap, our aim is to characterise AD in ML/DL systems by identifying its causes, effects, and mitigation strategies. We used a concurrent mixed-methods, combining 65 questionnaire responses and 21 semi-structured interviews with ML/DL practitioners. This enabled us to address three research questions (RQs): RQ1) What causes practitioners to accumulate AD in ML/DL projects? RQ2) What are the effects of AD in ML/DL systems? RQ3) What are the mitigation strategies used by practitioners when they encounter AD in ML/DL systems?

We found that the complexity of ML/DL algorithms and lack of developer knowledge are the primary causes of AD, with variations across roles. Poor model scalability emerged as the primary effect, while systemic testing and continuous monitoring were key mitigation strategies. The main contributions of our study addressing the management of AD is that we:

- Identified 13 causes, 8 effects, and 15 mitigation strategies of AD in ML/DL systems derived from practitioner insights.
- Identified and characterised AD in ML/DL systems, with a theoretical framework that synthesised its causes, effects, and mitigation strategies.
- Offered practical guidance for practitioners, such as training and the need for scalable design for ML/DL models, to manage AD effectively; and
- Identified the need for AD-aware tools serving as a guide for future research on AD in ML/DL systems.

The remainder of our paper is structured as follows: Section 2 provides related work on TD and AD. Section 3 details the methodology and Sections 4, 5, and 6 presents our findings on the causes, effects, and mitigation strategies of AD. This is followed by a discussion of the results in Section 7. Section 8 presents future work and Section 9 addresses threats to validity. We conclude our study in Section 10, with a summary of our findings.

2. Related Work

2.1. Technical Debt in Software Engineering

TD has been a foundational concept in software engineering since its introduction by Cunningham [15] as a metaphor. TD is described as the long-term cost of short-term development decisions, such as priotising speed over quality in delivering software. TD encompasses various types, including Code Debt (e.g., poorly structured code), Design Debt (e.g., architectural inefficiencies), and Documentation Debt (e.g., outdated manuals) [6].

Kruchten et al. [26] further formalised TD as a framework for understanding trade-offs in software projects, categorising issues like performance bottlenecks and code smells under broader categories such as code quality. Traditional mitigation strategies for TD often involve refactoring and improved documentation practices.

However, these studies were primarily focused on conventional software systems, where development cycles are linear and less dependent on data-driven iterative processes, limiting their applicability to the dynamic nature of ML/DL systems.

2.2. AD in ML/DL Systems

The rise of ML/DL systems has introduced a specialised TD type known as AD [28]. They defined AD as the suboptimal implementation of algorithm logic that negatively impacts system performance in DL frameworks. Their work identified the complexity of ML/DL algorithms and the iterative nature of model training as the cause of AD. Sculley et al. [43]'s work on ML systems explained ML-specific TD challenges, noting that ML systems often incur "hidden debt" through data dependencies, feedback loops, and the difficulty of maintaining models in production environments. They highlighted effects like degraded model performance and increased computational costs, as the effect of TD in ML systems.

Despite the identification of AD in ML/DL domain, the causes, effects and mitigation strategies remain underexplored with limited studies available.

2.3. Mixed-Methods Studies in Software Engineering

Mixed-methods approaches have gained traction in software engineering research providing a more holistic understanding of complex phenomena like TD. Onwuegbuzie and Johnson [35] advocated for combining quantitative

and qualitative methods to leverage the strengths of both (i.e., quantitative insights from questionnaires and contextual depth from interviews [24]).

Previous TD studies have investigated various aspects using mixed methods. One of the most influencial efforts is the InsighTD survey [41], a globally distributed family of industrial surveys on TD. The InsighTD was Launched in 2018, as a global project planned collaboratively with many researchers and developed to be executed incrementally based on continuous and independent replications in different countries. After the pilot study on InsighTD, TD researchers adopted the InsighTD survey and used it to investigate TD in various contexts.

For instance, Rios et al. [40] surveyed software practitioners to identify how they perceived the occurrence of Documentation Debt in their projects. They analysed results from two complementary studies: a survey (InsighTD) and an interview-based case study. Their work provided a list of causes and effects of Documentation Debt, along with practices that can be used to address it during software development projects. Similarly, Souza et al. [47] focused on Test Debt, investigating its causes and effects among software testing practitioners in Brazil.

Building on these works, Barbosa et al. [10] explored the state of the practice of Requirements and Requirements Documentation Debt (R2DD), revealing its causes, effects, and practices as well as the practice avoidance reasons (PARs) considered for its prevention and repayment. They analysed responses from a survey with software practitioners on R2DD and its elements, identifying 33 causes, 33 effects, and 26 prevention practices, which were then organised into a conceptual map.

More recently, Freire et al. [19] investigated the mitigation strategies on how to monitor TD in software systems. They analysed data from the InsighTD survey and found 46 practices for monitoring TD and 35 mitigation strategies. Also, Aldaeej and Alshayeb [5], investigated the causes of TD in the software industry. Likewise, Wang et al. [51] investigated the effects of TD on software systems. Ahmad et al. [3], used mixed methods to investigate Organisation Debt in three software industries to investigate the effects, causes and mitigation strategies of Organisation Debt.

Research Gap. Unlike prior research focused on general TD, Documentation Debt, Test Debt, or R2DD, our study applies a concurrent mixed-methods approach to investigate the causes, effects, and mitigation strategies of AD in ML/DL systems. This work fills a gap by identifying the causes of AD,

how it impacts ML/DL development, and what strategies practitioners use to manage it.

3. Methodology

3.1. Research Questions (RQs)

To achieve the goals of this research, we formulated the following three RQs to guide our investigation into AD in ML/DL systems:

- *RQ 1:What causes practitioners to accumulate AD in ML/DL projects?* The aim of this RQ was to identify the factors that lead to the accumulation of AD during the development of ML/DL projects. We asked practitioners about the reasons they incur AD, focusing on causes such as ML knowledge gaps and the complex nature of algorithms.

- *RQ 2: What are the effects of AD in ML/DL systems?* This RQ investigated the impacts of AD on ML/DL systems as they evolve over time. This was critical to pose as a way to understanding how AD affects system performance. We addressed this by asking practitioners to describe the consequences they experienced due to AD in their projects.

- *RQ 3: What are the mitigation strategies used by practitioners when they encounter AD in ML/DL systems?* The aim of this RQ was to identify the strategies that practitioners employ to manage and reduce the impact of AD in ML/DL systems. To answer this, we asked practitioners about the practices and techniques they have used to manage and address AD.

3.2. Research Design

We employed a concurrent mixed-methods design to explore the causes, effects, and mitigation strategies of AD in ML/DL systems. We deployed a questionnaire and conducted semi-structured interviews with ML/DL practitioners simultaneously. This concurrent approach combined the quantitative breadth of questionnaire data with the qualitative depth of interviews, to maximise data collection efficiency across a diverse participant pool. This method also allowed for a separate and parallel exploration of AD, with integration occurring after the data collection.

Unlike real-time triangulation, the data from the questionnaire and interviews were collected independently, ensuring that responses from one method did not influence the other during the collection phase, thus maintaining methodological integrity and time efficiency.

Before we conducted this study, we obtained ethical approval from the Australian National University (ANU) Human Ethics Committee (Protocol: H/2024/0982) and the University of British Columbia (UBC Behavioural Research Ethics Board Number: H25-01879). This ensured that we adhered to ethical guidelines, protecting the rights of participants.

3.3. Interview Process

3.3.1. Participant Selection

We interviewed 21 practitioners from the ML/DL domain to explore their experiences with AD. This is because their insights are valuable for understanding the causes, effects, and mitigation strategies of AD in practice. This sample size was determined based on the principle of data saturation, a widely accepted criterion in qualitative research [37], where additional data collection no longer yielded new insights. Our thematic analysis revealed that by the 18th interview, recurring themes related to the causes (e.g., ML knowledge gaps), effects (e.g., reduced model accuracy), and mitigation strategies (e.g., education and training) were consistently emerging, with minimal new information identified in the subsequent three interviews.

We identified our participants through personal contacts and recommendations from supervisors, This ensured we had access to qualified ML/DL practitioners with relevant experience in real-world challenges. We also posted on LinkedIn and X for possible participants, but did not interview any participant given that we did not receive volunteers within the time frame of conducting the interviews. The recruitment was conducted via email, targeting practitioners and researchers with expertise in widely used ML/DL frameworks such as TensorFlow, Keras, PyTorch, DL4J, and CNTK [28]. This focus ensured that participants possessed practical knowledge of commonly adopted tools, enhancing the relevance of their insights to the research questions. While this approach ensured expertise, it may limit generalisability beyond our network, however this was mitigated by the diversity of the participants.

We recruited the participants from five regions: Oceania, Africa, Europe, south America, and Asia, reflecting varied cultural contexts (Figure 1).

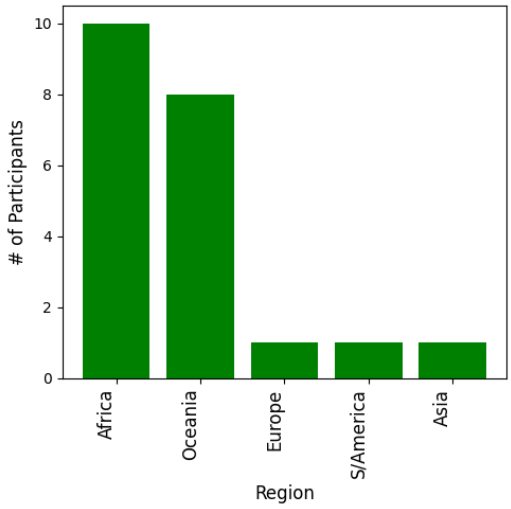

Figure 1: Distribution of interview participants by region. Note that S/America on the x-aixs represents South America.

The 21 participants represented a diverse range of experience, with years in ML/DL development or software engineering research ranging from two to over 15 years. This range enriched our study with perspectives from early-career practitioners to seasoned experts, depicted in Figure 2.

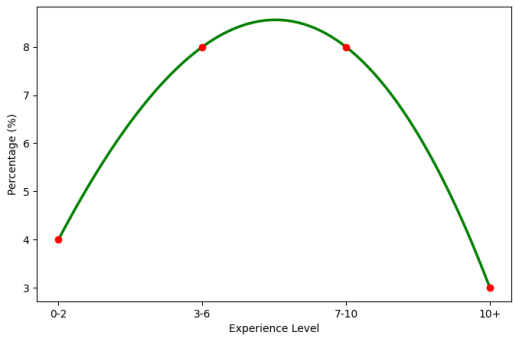

Figure 2: Distribution of Interview participants by years of experience

3.3.2. Interview Protocol

We conducted semi-structured interviews with the 21 ML/DL practitioners following the protocol outlined in Figure 3 (a flow of events from the pilot study to data analysis). Before the main interviews, we conducted a pilot

interview with an expert who has both experience in ML and research. This pilot study was crucial in refining the interview protocol, ensuring clarity and effectiveness (i.e., based on the expert's feedback), the questions were restructured to be more open-ended, enhancing the depth of responses. For instance, we removed questions like "What is TD to you?", and rather explained what TD means. We also focused on questions regarding AD to ensure the conversation remained centered on AD. The semi-structured approach provided a balance between consistency and flexibility, ensuring that all participants were asked a specific set of questions aligned with the RQs while allowing for in-depth exploration of their experiences.

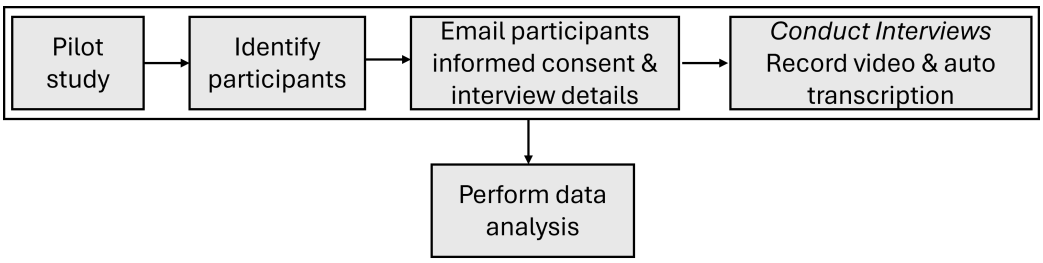

Figure 3: Interview Protocol. This outlines the protocol followed during the interview phase of the study. Note: Some participants may have also completed the survey, indicating a possible overlap between interviewees and survey respondents.

Table 1 summarises the key questions, such as "Have you ever needed to implement a suboptimal algorithm in any past or current ML/DL projects?" (RQ1) and "Do you think there are best practices in the industry that could be implemented to reduce AD?" (RQ3). We asked follow-up questions to clarify responses and explore insights in greater depth. This ensured a comprehensive understanding of participants' perspectives on AD causes, effects, and mitigation strategies.

We conducted the interviews online in English using Microsoft Teams[1] or Zoom[2], based on participants' preference. We selected English as the medium to eliminate the need for translation and ensure clarity, given its accessibility across the recruited regions (Australia, Nigeria, India, Canada, and the UK). Prior to each interview, we informed participants about the purpose of the study via email and provided written consent. The email also

[1] `https://www.microsoft.com/en-au/microsoft-teams/group-chat-software`
[2] `https://www.zoom.com/`

Table 1: RQ's and their key questions

RQ	Key Questions
RQ1	Have you ever needed to implement a suboptimal algorithm in any past or current ML/DL projects?
RQ1a	What specific cause or constraints led to the use of the suboptimal algorithm (i.e., what were the causes of incurring AD)?
RQ2	Did the use of this sub-optimal algorithm or the implementation lead to any long-term consequences or impacts on the system (e.g., performance degradation, scalability issues)?
RQ2a	Could team dynamics, like miscommunication, contribute to AD via algorithmic decisions in ML/DL systems or projects?
RQ3	Do you think there are best practices in the industry that could be implemented to reduce AD?
RQ3a	Reflecting on your past experiences, what do you think could help to prevent or mitigate AD?

contained a brief description of TD and AD, so that the practitioners could have an idea of the overall conversation. We also shared an interview guide containing the main questions in advance to facilitate preparation.

After obtaining consent from the participants, we recorded the interviews. We also used the automated transcription services provided by Zoom and Microsoft Teams to record the interviews. This ensured the accuracy and completeness of the auto-generated transcript. We then cross-verified the generated transcripts against the video recordings to mitigate errors from the automated transcription process. The interviews lasted between 15 and 45 minutes, depending on the depth of responses and the willingness of participants to elaborate on a given topic. To ensure clarity, we reiterated the concept of AD and the purpose of the study at the start of each session before proceeding to specific questions about their experiences with AD.

3.3.3. Qualitative Data Analysis

To analyse the interview data from the 21 ML/DL practitioners, we employed thematic analysis following the approach proposed by Braun and Clarke [14]. We selected thematic analysis for its suitability in identifying, organising, and interpreting patterns within qualitative data, making it well-suited to addressing our RQs on the causes, effects, and mitigation strategies of AD in ML/DL systems.

We started our analysis by cross verifying the automated transcripts generated by Zoom and Microsoft Teams against the video recordings to ensure accuracy and completeness. After the transcription, we familiarised ourselves with the data by reviewing the transcript from each interview. We noted initial observations and patterns to develop a deep understanding of the responses from the participants' related to AD.

Next, we assigned initial codes to the data, guided by our RQs. We applied descriptive codes to interview excerpts i.e., identifying instances of ML knowledge gaps under RQ1 (causes), model degradation under RQ2 (effects), and improved stakeholder communication practices under RQ3 (mitigation). We iteratively refined and grouped these codes into broader themes of causes of AD, effects of AD, and mitigation Strategies . We used NVivo 14[3] to help organise the codes and for efficient retrieval of relevant excepts.

The first author performed the thematic analysis to generate the themes. To validate the themes, the remaining four authors reviewed and refined them through iterative discussions during weekly meetings. This collaborative process involved two rounds: the first focused on themes related to causes, and the second addressed effects and mitigation strategies. Themes that were too broad or overlapping were redefined or split into sub-themes for clarity e.g., "team culture" as a sub-theme under "team factors". Clear definitions and naming ensured the themes represented participants insights and aligned with the objectives of our study.

Due to the open and iterative nature of the discussions while creating themes, inter-rater reliability was not calculated, as the interactive process was deemed sufficient for this qualitative study [14]. This approach ensured a flexible analysis, capturing insights from the interview data to inform the findings of our study on AD.

[3] `https://lumivero.com/product/nvivo/`

3.4. Questionnaire Design and Distribution

3.4.1. Questionnaire Design

To complement the qualitative insights from our 21 interviews, we deployed a structured questionnaire designed to collect data across a broader sample of 65 ML/DL practitioners. This was to enhance the generalisability of our findings on AD. While the interviews provided in-depth individual experiences, the questionnaire enabled us to quantify patterns across diverse roles and technical domains, addressing the RQs on the causes, effects, and mitigation strategies for AD. The anonymous nature of the questionnaire encouraged respondents to share information about technical challenges such as barriers to addressing AD that might not have been disclosed in face-to-face interviews, enriching the scope of our study.

We designed the questionnaire through a three-step process. First, we reviewed the InsighTD survey [41] as a guide, drafting the initial questions tailored to AD in ML/DL systems. Next, we refined the questions through collaborative sessions with domain experts in ML/DL and software engineering researchers. Third, we conducted three rounds of refinements: i) an internal review by the authors to assess question clarity and alignment with the RQs, ii) expert validation by a senior ML practitioner to ensure relevance, and iii) a pilot test with five potential respondents (for initial feedback) to identify ambiguous questions and terminology.

The iterative process of designing the questionnaire improved the question wording, added contextual definitions for technical terms (e.g., AD), and restructured the question flow to enhance completion rates. For instance, we received feedback on the need to group questions by theme (i.e., causes, effects, and mitigation strategies). We also shared an example of AD via a code snippet for clarity and to reduce the risk of wrong interpretation. The final questionnaire, implemented online via Qualtrics platform, was deployed concurrently with the interviews from October 2024 to February 2025, with data collected independently to maintain methodological integrity.

3.4.2. Sampling and Distribution

We targeted industry professionals actively involved in the development, deployment, and maintenance of ML/DL systems to participate in the questionnaire, aiming to capture diverse perspectives on AD. The sample included ML/DL engineers responsible for designing, deploying, and maintaining ML/DL systems in production environments, ML/DL developers focused

on integrating existing models into applications and building ML-driven prototypes. In addition, the sample included ML/DL researchers who have advanced theoretical understanding of ML/DL techniques, and also ML/DL analysts who are involved in interpreting model outputs, and evaluating performance. Participants were required to have at least one year of hands-on experience with frameworks such as TensorFlow, Keras, PyTorch [28], ensuring sufficient exposure to AD-related challenges.

We employed a multi-sampling technique to ensure the diversity and representativeness of the 65 valid respondents. This approach combined purposive sampling through professional networks and snowball sampling, where the respondents were encouraged to share the questionnaire with colleagues in similar roles, reaching both widely connected professionals and niche communities.

We distributed the questionnaire via multiple channels including professional platforms like LinkedIn and X (formerly Twitter), shared within specialised ML/DL forums (e.g., PyTorch community, TensorFlow developers on LinkedIn). We also sent emails to 21 contacts from the interview pool, and promoted the outreach of the questionnaire through QR codes at two major workshops where ML/DL practitioners were encouraged to participate. We note that the sampling could risk bias, however, we mitigated this by our broad platform outreach.

The distribution of the questionnaire spanned a four-month period from October 2024 to February 2025, overlapping with the period when we conducted the interviews in September and October 2024. This concurrency ensured broad engagement while maintaining independent data collection, supporting the objectives to quantify AD patterns across the RQs (i.e., RQ1 for causes, RQ2 for effects, and RQ3 for mitigation).

3.4.3. Data Collection

The questionnaire consisted of 16 questions that we designed to address the RQs on AD in ML/DL systems, collecting data from 65 valid respondents from a total of 69. For example, one of the removed responses had only the demographic section completed, with all subsequent questions left unanswered.

The questions were grouped into six sections, aligning with the objectives of our study to explore the causes (RQ1), effects (RQ2), and mitigation strategies (RQ3) of AD. The first section (Q1 — Q4) included demographic questions to characterise respondents, providing context to their responses.

The second section (Q5 — Q6) assessed the familiarity of the respondents with AD. The third section (Q7 – Q8) explored the experiences of respondents with AD and factors contributing to its occurrence (RQ1), such as ML knowledge gaps. The fourth section (Q9 — Q11) focused on identifying the effects of AD effects (RQ2), by asking respondents to reflect on consequences like reduced accuracy. The fifth section (Q12 — Q13) examined mitigation strategies (RQ3), seeking insights into practices like improved data management. The sixth section (Q14 — Q16) further probed the broader impacts of AD across projects. The complete list of questions are found at our online repository[4].

Unlike the original InsighTD survey, we excluded the section on practitioners' responses to TD awareness, as it fell outside the scope of our study which was focused solely on AD.

3.4.4. Characterisation of Participants

We received a total of 69 responses to the questionnaire, with 65 that were valid for analysis after excluding incomplete entries. The participants were distributed across diverse regions (Figure 4 i.e., a bar chart of regional distribution), with the highest engagement from Australia (32%), reflecting strong local engagement, followed by Europe (29%) and Africa (23%), indicating growing engagement in these regions. Lower participation from North America (5%) and Asia (11%) suggests opportunities for targeted outreach in future studies. This geographic diversity ensured a broad range of cultural and professional perspectives on AD in ML/DL systems.

The expertise levels of participants (Figure 5) followed a roughly bell-shaped distribution, peaking at Competent (35%), with a decline toward Novice (9%) and Expert (8%). Most respondents (83%) were categorised as Beginner (18%), Competent (35%), or Proficient (22%), indicating practical experience with varying depths of mastery, while only 17% were Novices or Experts, reflecting a balanced sample with sufficient exposure to AD challenges. Notably, 50% of respondents were either not familiar (28%) or slightly familiar (22%) with AD, highlighting limited awareness in the broader community, while only 12% were very familiar, underscoring the need for educational resources on AD.

The role distribution (Figure 6) showed a high representation of Re-

[4]`https://github.com/ikoojos/Practitioner-Insights-on-Algorithm-Debt/`

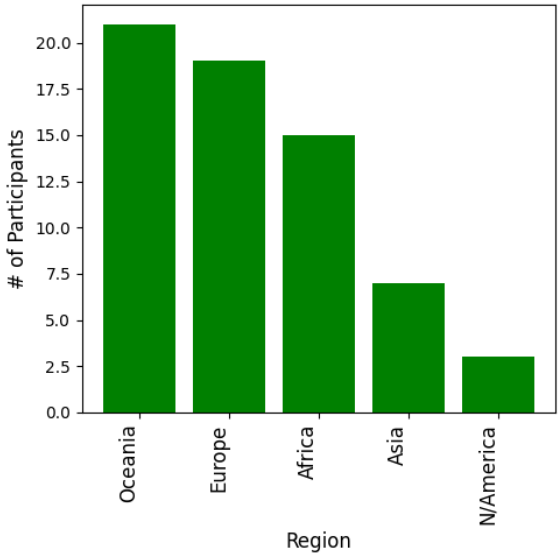

Figure 4: Distribution of questionnaire participants by region. N/America represents North America.

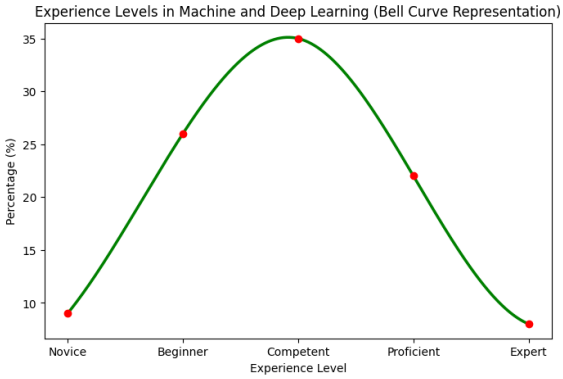

Figure 5: Distribution of questionnaire participants by experience

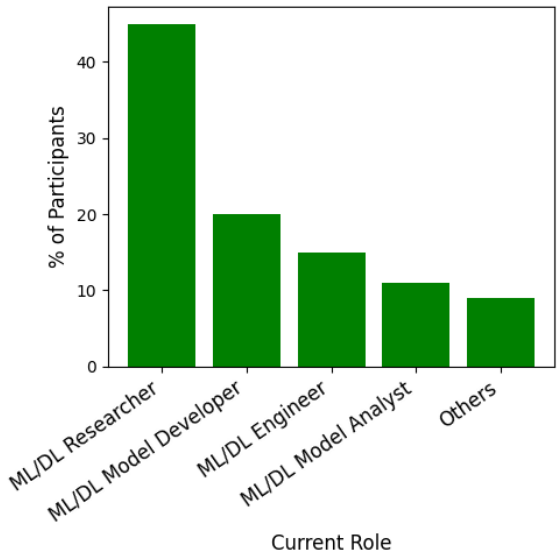

Figure 6: Distribution of current professional roles of questionnaire participants

searchers (45%), suggesting AD's interest in academia and exploratory ML/DL projects. Developers (20%) and Engineers (15%) comprised 35% of the sample, ensuring practical perspectives, while Analysts (11%) and Others (9%) were less represented, indicating AD's prominence in development and engineering stages. Over half of the respondents (51%) answered based on their current role, 31% drew on both current and previous roles, and 18% relied solely on past roles (Figure 7).

This distribution (82% including current experience) strengthens the relevance of our findings to the present state of the field, while the 31% reflecting on both roles provides deeper insights into AD.

4. Causes of AD (RQ1)

4.1. Interview Findings

We identified three broad categories from the thematic analysis of the 21 semi-structured interviews as factors contributing to AD in ML/DL systems: i) Organisational Factors, ii) Software Engineering Principles, and iii) ML Pipeline. These categories, provided nuanced insights into the causes of AD, addressing RQ1: What causes practitioners to accumulate AD in ML/DL projects?

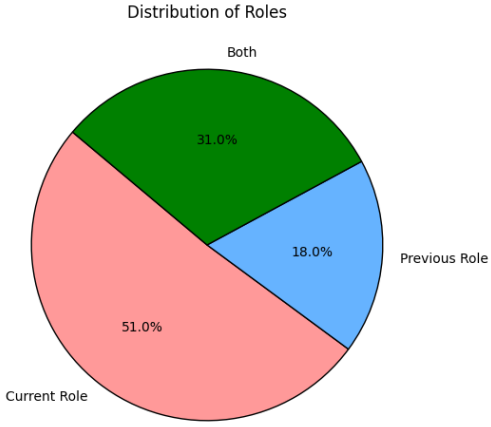

Figure 7: Role of Participants

4.1.1. Organisational Factors

Team Culture. Developers mentioned team culture as a factor, noting a tendency to avoid complex (possibly efficient) solutions due to perceived risks or fear of conflict. A senior ML expert explained, *"There can be a kind of culture that occurs among development teams which prides itself on stupidity, where if you do implement something sophisticated and clever, even though it might dramatically improve an algorithm, it might well be rejected by the rest of the team. And that could be a clear up front reject."* Another developer added, *"This cultural dynamics can result in a 'stupidity pride' mentality, where simplicity is valued over innovation, and team members are discouraged from exploring new ideas"*, reflecting groupthink dynamics [22]. This aversion to complexity stifles innovation, increasing AD by hindering potentially scalable solutions.

Communication Among Team Members. Given the iterative and experimental nature of ML/DL workflows, continuous and structured communication is important to ensure alignment across critical stages of the ML development lifecycle, including model design, algorithm implementation, and integration into production pipelines [42]. A ML developer with production experience explained: *"The second issue is that we are lacking communication with other developers. Improved communication would have allowed us to gain more information regarding the packaging, implementing of the ML/DL models, and conducting integration tests for our models."* This quote highlighted how

communication gaps can negatively influence implementation of ML models or neural network integration workflows. For example, variations on how team members interpret design specifications can lead to conflicting implementations [4]. This could lead to inconsistent model implementations and fragmented training workflows.

Experience in Using ML/DL Frameworks. Developers identified expertise in ML/DL as a possible cause of AD. One participant stated, *"But lack of knowledge will allow you to only use what you know and what you know might not be enough to address the situation. We did not know how to use some of the frameworks, so could not fully utilise them."* This quote highlighted that lack of expertise in ML/DL frameworks restricts developers to basic functionalities, limiting their ability to implement efficient algorithms for distributed training, automatic differentiation, or dynamic computation graphs [29, 52]. This lack of knowledge can result in missed opportunities to optimise system performance leading to AD.

4.1.2. Software Engineering Principles

Simplicity Over Optimisation. Choosing simplicity rather than optimising systems was highlighted, where participants noted, *"In that case, people would tend to use something simpler, and only when they find out that this actually causes problem, they will start changing that."* While simpler solutions can expedite development and reduce short-term complexity, they may lead to significant long-term challenges, such as reduced efficiency, scalability issues [50], leading to the accrual of AD.

Outdated Assumptions. Participants highlighted outdated assumptions as possible causes of AD. A participant remarked, *"It's probably dating back to like 20 years ago, something like that. So a lot of assumption of what is good and what is bad at that point no longer true at this point, right? So some part of the code we thought was not performance critical, but now it's causing issues for us."* This quote highlights legacy issues that can arise and become problematic as system contexts evolve over time. Although the participant did not explicitly mention hardware, drawing on this, advances in computational capabilities such as NVIDIA A100 GPUs and TPUs can exacerbate inefficiencies of legacy algorithmic patterns [16]. This could necessitate revisiting the original hardware and software implementations.

Poor Documentation of Pretrained Models. The absence of detailed documentation, or implementation guidelines can disrupt workflows and impede efficient model reuse [2], leading to either inefficient model reuse or wrong use of components. One participant stated: *"Say they have pretrained some models. They put it online so that others can just fork it and use it for their applications, but in most of the cases it is that there is some particular detail which is missing, a particular very important file which is missing or some kind of documentation which is missing which stops us from using their models efficiently."*

From this statement, unlike Documentation Debt, which affects maintainability, missing documentation could force developers to implement suboptimal algorithmic solutions that incur computational interest over time. With missing information such as hyperparameter configurations, or model initialisation strategies, practitioners must reverse-engineer or approximate these algorithmic components, creating implementations that become increasingly inefficient as the system scales. Without clear guidelines, practitioners may face challenges in adapting pretrained models to domain-specific tasks, such as transfer learning for specialised datasets or model interpretability for compliance requirements.

4.1.3. ML Pipeline

Data Drift. Data issues such as data drift was a concern, with participants stating, *"One of the main issues there is about the data that you are going to use. Let's say you develop your ML model. You need to train your model. You need to evaluate, validate, and test your model. And all of that depends on the data that you have and maybe at that point you have one data set. But at some point you may have more data sets available that has different characteristics and then your model needs to be a little bit more general. So this change will cause you to re-evaluate and re-train your model."* This misalignment in data [13], could result in degraded performance when exposed to such data with altered distributions.

Feature Extraction. Feature extraction processes were identified as issues that could contribute to AD. A participant noted, *"We were in a hurry and so what we ended up doing was instead of working out what the process should be to get the right data, I just added an extra prompt and an extra tool for the language model. We did not follow the right steps for data preprocessing and management."* Proper feature selection and engineering create algorith-

mic efficiency by reducing dimensionality, but inefficiency in this process i.e., in high-dimensional spaces with spurious correlations, become problematic [17]. This decreases prediction accuracy of ML models which contributes to AD. Conversely, focusing on an ideal set of features through proper data preprocessing can yield optimal results.

Hyperparameter Optimisation. Failure to fine tune hyperparameters could lead to AD, with a participant saying, *"As an ML developer instead for you to optimise your parameters, you will decide not to optimise them at that particular time"*. During the development of ML/DL models, hyperparameters such as learning rate, batch size, and batch normalisation may not be fine tuned possibly due to the computational resources and the time required to search for the optimal parameters [54]. While not using the optimal hyperparameters may meet short-term objectives, such as adhering to tight project deadlines, this could introduce AD that manifests in the form of degraded model performance after deployment [30, 1].

GPU constraints. Limitations in computational power could significantly constrain training of DL models [25]. One developer noted: *"Yes, it could have been avoided if I maybe had access to some GPUs. We required extensive resources to actually resolve the issues (and train our models)."* This illustrated how resource availability impacts computationally intensive tasks like training large DL models [25]. The resulting AD is caused by training inefficiencies such as reducing the training epochs, that could compromise model accuracy.

Time Limitations. Project deadlines could force ML/DL practitioners to implement temporary solutions that incur AD debt through expedient decisions. One participant stated: *"So you have a deadline and you have already built most of the pipeline and there is just one part which needs some kind of tuning to the DL architecture. So you just make temporary inefficient adjustments to that part and you at that point you do get the results."* This illustrated that AD could accumulate when practitioners make expedient rather than optimal algorithmic decisions. Deadlines may force teams to implement non-modular architectures with hard-coded layers that satisfy immediate needs but create rigid designs that resist future adaptation to new requirements or datasets. Also, skipping proper model development steps under time constraints can significantly reduce classification accuracy [23].

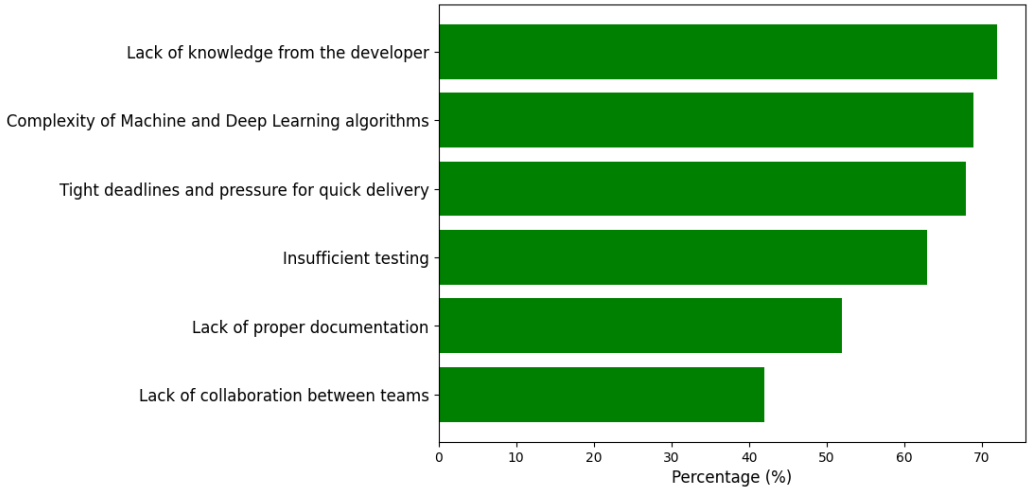

Figure 8: Human Causes of AD

4.2. Insights from Questionnaire

The questionnaire quantified causes of AD in ML/DL systems, complimenting the insights from the interviews addressing RQ1. We categorised the causes into human and technical factors shown in Figure 8.

4.2.1. Human Factors

Lack of knowledge and proficiency in the use of ML/DL frameworks by practitioners was the most highlighted cause of AD, selected by 72% of respondents (47 respondents). The *complexity of ML/DL algorithms* was next at 69% (45 respondents), reflecting challenges posed by ML/DL algorithms. *Tight deadlines and pressure for quick delivery* ranked third at 68% (44 respondents), which highlighted that project constraints could lead to AD.

Insufficient testing was noted by 63% (41 respondents), indicating that inadequate verification and validation of algorithms could contribute to the accumulation of AD. *Lack of proper documentation* was reported by 52.3% (34 respondents), this indicated that insufficient documentation on how to integrate models could lead to AD. *Lack of collaboration* between teams (41.5%, 27 respondents), indicated that organisational factors also contributed to AD, though it was the least cited among the human factors.

Technical Factors. Figure 9 assessed the frequency of causes of AD based on technical factors. *Hardware issues* were the most frequent causes, with 39.1%

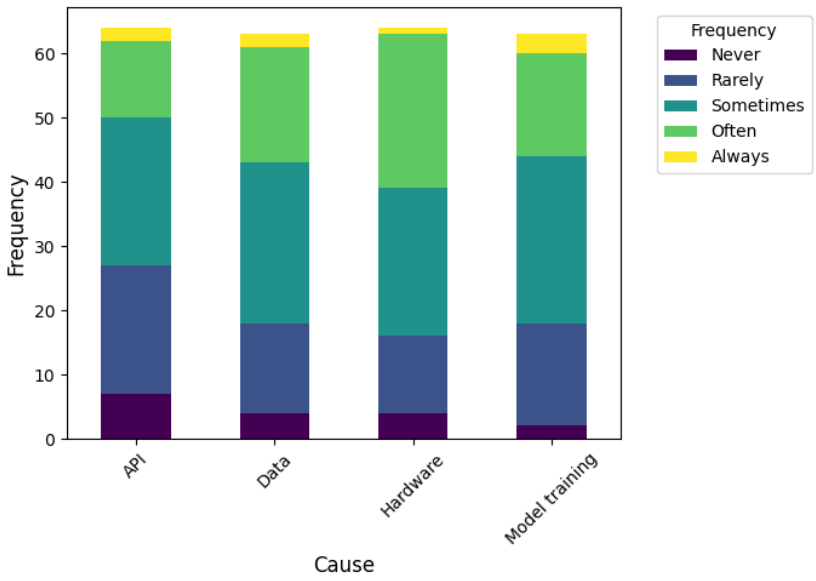

Figure 9: Technical Causes of AD

of respondents reporting that hardware often or always led to them incurring AD. This highlighted the effect of resource constraints leading to incurring AD. *Data issues* followed at 31.8% (often/always), aligning with challenges like data drift. *Model training issues* were selected by 41.3% citing them as sometimes occurring, suggesting variability in issues related to training leading to AD. *API-related issues* were the least frequent, with 42.2% reporting they rarely occurred, indicating that API challenges are less pervasive in contributing to AD.

4.2.2. Causes by Role

To further explore the causes of AD identified, we analysed the questionnaire responses to examine how the causes of AD varied across different roles within the ML/DL community (Figure 10).

ML/DL Developers reported the highest concern for lack of developer knowledge (90%, 9/10), while Model Analysts were least concerned about this cause (50%, 3/6) and Tight deadlines (33.3%, 2/6). Researchers and Model Developers both identified Complexity of ML/DL algorithms at 79.3% and 78.5% (23/29 and 11/14, respectively), but Researchers were more concerned about Lack of documentation (58.6%, 17/29) compared to Model Developers (28.5%, 4/14). Builders/Engineers reported higher concerns for Lack of team

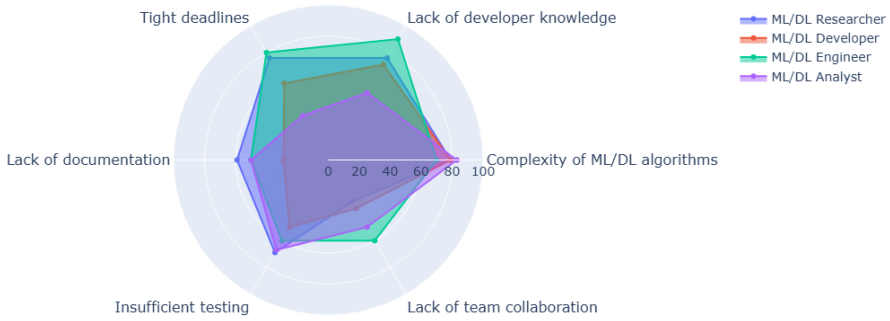

Figure 10: Radar chart displaying the causes of AD categorised by role type

collaboration (60% , 6/10) compared to Researchers (31%, 9/29).

4.3. Integrated Insights

The questionnaire identified lack of developer knowledge as the predominant human factor (72.3%, 47 respondents), a finding that was reinforced by the role-specific analysis, where this cause was the most cited by Builders/ Engineers (90%, 9/10). The interviews provided deeper context, with participants noting, *"Lack of knowledge will allow you to only use what you know and what you know might not be enough,"* highlighting how knowledge gaps lead to suboptimal implementations. Factors like hardware constraints (questionnaire, 39.1% often/always) and data drift (interviews) further underscored the interplay between human and technical challenges.

The role-specific analysis also revealed that complexity of ML/DL algorithms (78.1%, 50/64) is a universal concern across roles, particularly for Model Analysts (83.3%, 5/6) and Researchers (79.3%, 23/29), suggesting that the inherent complexity of ML/DL systems is a pervasive driver of AD. Additionally, the interview theme of team culture (e.g., "stupidity pride" mentality) aligned with the role-specific finding that lack of team collaboration is a significant concern for Builders/Engineers (60%, 6/10), indicating that organisational dynamics introduce AD in roles requiring cross-functional teamwork. These findings suggested that AD arises from a combination of knowledge gaps, technical constraints, and role-specific organisational challenges, necessitating targeted mitigation strategies. A detailed view of the

causes from both the interviews and surveys is shown available in our supplementary materials[4].

5. Effects of AD (RQ2)

5.1. Interview Findings

We identified two broad categories of effects of AD in ML/DL systems, addressing RQ2: What are the effects of AD in ML/DL systems? These categories e.g., system performance issues, human and resources, highlighted the multifaceted impacts of AD on both technical and human aspects of ML/DL systems development.

5.1.1. System Performance Issues

Lack of Scalability. In ML/DL system development, AD could arise when models are designed without considering future scalability needs, leading to performance bottlenecks as computational demands increase [9]. A participant highlighted this challenge: *"But when it comes to implementing the same algorithm on a large-scale dataset, it might not really work because the underlying data structures and algorithm that you're using are not optimal, not time-efficient."* This statement underscored how suboptimal design choices (i.e., inefficient data structures) can lead to significant computational overhead. For example, training pipelines optimised only for small datasets may suffer from excessive memory consumption or out-of-memory errors when scaled up [32]. Similarly, the choice of suboptimal tensor representations can slow down gradient computations and increase memory allocation overhead in frameworks like TensorFlow and PyTorch. Moreover, scalability issues can emerge if model architectures and computational infrastructure are not designed to handle increasing data volumes, leaving practitioners refactoring code under tight time constraints, leading to further technical compromises.

Increased Computational Cost. Inefficiency in ML/DL systems could manifest if these systems implement suboptimal querying patterns in LLMs. This could be due to prioritising immediate development gains over long-term operational efficiency. As one participant described: *"It's gonna cost much more to run over the long term. Because instead of saying, here's all the information, you ask them to give an answer. The LLM is gonna be queried multiple times, so it's gonna take more time to complete execution."* This

statement illustrated how algorithmic implementation choices (i.e., the decision to make multiple iterative API calls rather than designing more comprehensive, batched queries) increases the computational cost of the system. Each redundant LLM query not only increases API costs proportionally but compounds latency issues in production environments.

Also, this increased computational costs could extend beyond API expenses. Repeated model inference incurs higher energy consumption particularly in large-scale deployments. As these inefficiencies scale, they may necessitate redesigns to optimise resource utilisation.

Performance Degradation. In real-world ML/DL applications, the dynamic nature of data can lead to data drift. This 'shift' in data characteristics contributes to AD (i.e., Section 4.1.3 where we discussed data drift as a cause of AD). Models trained on historical data may become misaligned with the current data distribution, resulting degraded model performance. One participant described this challenge saying *"Initially, our implementation was perfect and working well; however, with time, we noticed prediction errors in our model due to a change in the overall distribution of the data."* Data drift typically arises in production environments where the model encounters real-time input data that differs from the training dataset. For instance, changes in feature distributions (covariate shift) or shifts in the relationships between input features and target labels (concept drift) can lead to degraded models [31]. These deviations introduce differences between the training and production data characteristics, causing classification or regression outputs to that are unreliable.

5.1.2. Human and Resources

Reduced Developer Productivity. The use of unmaintained ML/DL libraries or frameworks in system development was identifies as a contributor to AD. As frameworks become outdated, they often fail to integrate seamlessly with modern environments, forcing practitioners to allocate valuable time patching issues rather than focusing on building solutions, leading to delays in carrying out important tasks. This was described by a developer: *"In the first few months, I wasn't doing my best work because I was spending time working on these unmaintained ML/DL packages fixing this debt."* This reduces the overall productivity of practitioners, as substantial effort is required to refactor, debug, and update dependencies rather than advancing core model improvements. According to Besker et al. [12], developers spend

approximately 23% of their time managing TD.

Increased cost of Maintenance. Practitioners highlighted that failing to manage AD early led to higher resource expenditures in their projects. This effect is pronounced in ML/DL contexts, where the iterative nature of model development and dependency on complex algorithms exacerbate the challenge. One developer stated that *"Bad thing from this TD is that they accumulate at that some point and it's a big cost to fix."* This highlighted how accumulated AD, makes fixes expensive. For instancel, as models evolve, systems initially designed with inefficient optimisation may require modifications to meet new performance benchmarks. This may involve retraining with updated datasets, re-tuning hyperparameters, or redesigning model architectures to address inefficiencies [11]. Addressing these accumulated AD could lead to demands for additional resources, including extended developer time.

5.2. Questionnaire Findings

The questionnaire provided quantitative insights into the effects of AD in ML/DL systems, addressing RQ2. Of the 65 respondents who answered this question (out of 66 total survey participants), 29 (44.6%) rated the impact of AD in their projects as moderate, indicating meaningful challenges to system performance. Approximately 26.1% (17 individuals) reported a significant impact, suggesting severe consequences are relatively common, while 18.5% (12 respondents) categorised the impact as minor, reflecting variability in the impact of AD across projects.

Specific effects of AD (Figure 11) were highlighted, with *poor model scalability* as the dominant issue, and was selected by 73.8% of respondents (48 out of 65). This underscored AD's role in hindering the ability of ML/DL systems to handle increased data or computational demands. *Decreased model accuracy* and *increased maintenance costs* were the second most common effects, each reported by 60% (39 respondents). This demonstrated the dual impact of AD on functional performance and resource demands.

Delayed project timelines was reported by 52.3% (34 respondents), which indicated the influence of AD on project management. Increase in model complexity was stated by 44% (29 respondents), suggesting a compounding effect where systems become harder to maintain. *Model bias*, was reported by 36.7% (24 respondents), which was less prevalent, possibly indicating that practitioners view bias as distinct from AD. *Loss of stakeholder confidence*,

at 30.7% (20 respondents), was the least common effect, suggesting external impacts are less visible than technical consequences. Other reported effects included poor replicability in research.

5.3. Integrated Findings

The questionnaire and interview findings identified *poor model scalability* as a major effect, with the questionnaire reporting it at 73.4% (48 respondents) and interviews elaborating on its causes, such as a participant noting, *"The underlying data structures and algorithm that you're using are not optimal, not time-efficient,"* linking to issues like excessive memory consumption. Decreased model accuracy, cited by 60% in the questionnaire (39 respondents), aligned with the interview theme of performance degradation due to issues like data drift, where a participant stated, *"We noticed prediction errors in our model due to a change in the overall distribution of the data"*.

Increased maintenance costs, also at 60% in the questionnaire, were echoed in the interviews, with a developer explaining, *"It's a big cost to fix,"* emphasising the resource strain from accumulated AD. The questionnaire's *increased model complexity* (44%) and interview findings on increased computational cost (e.g., *"The LLM is gonna be queried multiple times"*) both point to long-term operational challenges, that AD could cause.

However, the interviews uniquely identified *reduced developer productivity* (e.g., *"I wasn't doing my best work because I was spending time working on these unmaintained ML/DL packages"*), which the questionnaire did not capture. While the *loss of stakeholder confidence* (30.7%) added external dimensions that were absent from the interviews.

6. Mitigation Strategies

6.1. Interview Findings

Based on the responses from the ML/DL practitioners that we interviewed, we categorised the mitigation strategies into three main groups: i) Team-related approaches, ii) Software engineering approaches, and iii) ML pipeline strategies.

6.1.1. Team

Communication Among Developers. We identified poor communication among development teams, which can result in wrong assumptions, building incompatible models, and integration failures as a contributor to AD. To mitigate

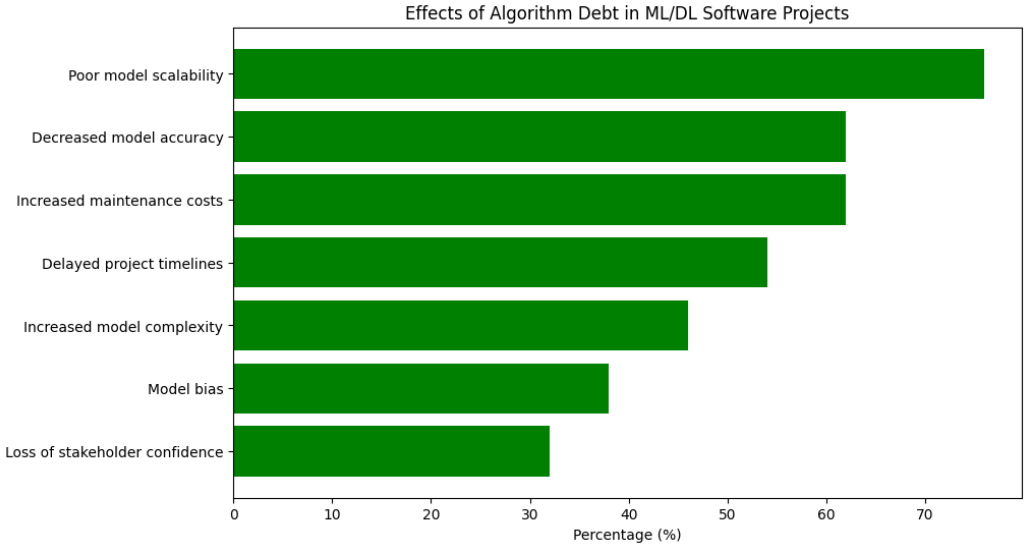

Figure 11: Effects of AD

AD, one developer emphasised the importance of communication in reducing AD: *"If we could have the communication with other developers, we can have more experience in the packaging also in the implementation of the algorithm and also in the integration test for our models."* This suggested that open communication can help development teams to resolve potential AD issues early by ensuring that MLOps are aligned across teams. When development teams openly share design decisions, potential risks, and integration concerns, they can proactively address hidden dependencies and prevent unexpected failures, thereby reducing AD accumulation.

Empirical studies support this approach, showing that effective collaboration reduces integration risks, hence minimising the accumulation of TD [33]. Therefore, fostering a culture of open, structured, and continuous communication is a key strategy for preventing AD in large-scale ML/DL systems.

Awareness of Team Capabilities. Self-awareness among teams was linked to better project planning and decision-making. For instance, a participant stated: *"Now, if you know very well your internal conditions or your internal characteristics, what are your strengths, and what are your weaknesses, you will know exactly what you can do and what you cannot do under a certain time."* This perspective highlighted that understanding team strengths and limitations allows for better alignment between project expectations and ac-

tual execution capacity, thereby reducing AD risk. In the context of ML/DL systems, this translates to recognising internal strengths, such as access to high-quality and diverse datasets, skilled team members proficient in advanced frameworks like TensorFlow or PyTorch, and robust computational resources (e.g., GPUs or TPUs). Conversely, weaknesses might include technical limitations, such as inadequate infrastructure or gaps in domain-specific expertise. Identifying these strengths and weaknesses allows teams to set realistic goals to align project timelines with available resources and technical capabilities. This can ultimately help teams avoid over-promising deliverables, reducing the likelihood of accruing AD.

Collaboration with Experts. Effective collaboration among domain experts with diverse perspectives was identified as a strategy for mitigating AD in ML/DL systems in large projects. A participant emphasised the importance of diverse expertise in problem-solving: *"You know why I think so is because in ML/DL there are domain experts. And since all of you have different viewpoints, you know that will also help because when you collaborate, you come with different viewpoints which will help you guys to brainstorm and then come up with an optimal solution."* The multidisciplinary nature of ML/DL projects often requires input from software engineers and subject-matter experts, each bringing unique insights into a model's design, implementation, and deployment. While collaboration is general good practice, the complex nature of ML/DL development means that expertise in areas such as data processing ensures efficient handling of large datasets, addressing issues such as data variability and feature extraction. Experts in algorithm design such as ML engineers and researchers bring insights into model architecture design and hyperparameter tuning, which is critical for optimising system performance, reducing the likelihood of accumulating AD over time.

Knowledge of ML/DL Frameworks. The complexity of ML/DL development often requires using different ML/DL libraries, frameworks, and tools, each with specific capabilities and limitations. Adequate knowledge of ML/DL frameworks can help teams in designing their systems to avoid mistakes such as choosing outdated or unsupported packages, which can contribute to AD. One participant said *"I think, first of all, the supervisor should be very experienced in using these DL packages like PyTorch."* Supervisors with extensive knowledge of ML/DL libraries can guide teams to select the best frameworks, ensuring compatibility and maintainability. For example, us-

ing modern frameworks like PyTorch or TensorFlow instead of unmaintained packages reduces the risk of TD accumulation [36].

6.1.2. Software Engineering Approaches

Clear Understanding of Project Goals. Without a clear understanding of the desired outcomes, projects risk becoming misaligned, leading to inefficiencies, incomplete implementations, or the accumulation of technical and algorithmic compromises. One participant noted that clear understanding of project goals could mitigate this cause, saying: *"We need to have clear goals about the ML system that we are building."* This clarity in project goals serves as a mitigation strategy against AD by enabling more informed algorithmic choices from the project outset. Well-defined goals allow teams to select appropriate model architectures and optimisation strategies that align with specific performance requirements, preventing the need for costly refactoring later. For example, understanding whether a computer vision system requires real-time inference on edge devices or can tolerate higher latency in cloud environments directly informs architecture selection between lightweight MobileNets or more complex EfficientNets [48].

Identifying possible externalities. Externalities (i.e, factors beyond the direct control of developers such as changes in data sources, shifts in computational environments, or unforeseen user behaviors) present challenges in ML contributing to AD. Recognising these factors is critical to ensuring the long-term performance and reliability of ML/DL models hence mitigating the risk of accumulating AD. One participant stated: *"You don't know the situations... that of course is like you don't control them. So that's why they are externalities. But you need to be aware of them and and it's about the conditions and the environment. In order to mitigate this, you need to identify all these externalities and try to tackle them as soon as you can to reduce the impact of the debt they can cause"*. This strategy supports the implementation of adaptive algorithmic approaches rather than static implementations that quickly become outdated when externalities change. For example, instead of hard-coding feature transformations based on current data distributions, teams can implement adaptive normalisation techniques that can adjust to distribution shifts. This proactive approach prevents the accumulation of AD that would otherwise require expensive retraining or model refactoring when externalities inevitably change.

Education and Training. In ML/DL domain, where technology evolves rapidly, the ability to adapt and stay current is essential for mitigating AD. Unlike traditional TD that primarily arises from code quality issues, AD could stem from knowledge gaps regarding algorithmic efficiency. One participant stated: *"So now let's say that as a ML/DL developer you must constantly study. You must constantly know what are the new trends in maybe a particular programming language. What are the upgrades that have been done to this programming languages..."* This highlighted the importance of continuous learning for ML/DL developers as mitigation strategy for AD. For instance, education on optimisation techniques enables developers to make implementation choices that avoid common inefficiencies such as in tensor operations, gradient calculation, and memory management (i.e., preventing the introduction of performance-related AD from the onset). Also, a developer trained on modern DL optimisation techniques may implement adaptive learning rate scheduling rather than static approaches, thus avoiding the algorithmic inefficiencies that would otherwise require costly refactoring when scaling to production.

6.1.3. ML/DL Pipeline

Continuous monitoring and model retraining. AD in ML/DL systems uniquely accumulates over time due to the dynamic nature of real-world data distributions and operational environments. Continuous monitoring and systemic model retraining can serve as critical mitigation strategies by remediation of performance degradation before it compounds into significant debt. As one practitioner with experience maintaining production ML systems emphasised: *"You need to continuously monitor the performance criteria so that you know how this ML or DL model is scaling or accommodating the service it performs... you continue testing and monitoring..."* Unlike traditional software systems where functionality remains consistent unless code changes, ML/DL systems can accumulate AD as data distributions evolve, making continuous monitoring an essential mitigation strategy rather than merely an operational best practice [49, 8]. For example, detecting gradual decline in model performance for specific data segments can reveal algorithm-environment misalignment that would otherwise remain hidden until creating significant AD.

Iterative Optimisation. Iterative optimisation through benchmarking was highlighted as a mitigation strategy for AD as it provides an empirical foundation

for identifying and resolving algorithmic inefficiencies before they compound into significant TD. One participant explained: *"So we have a quick implementation to get the model working, then we can run like benchmarks or workloads to collect some data. Then we know where the bottleneck is, then we optimise on the bottleneck."* This strategy addresses AD by creating a structured approach to identifying areas with potential AD rather than speculation. AD mitigation through benchmarking leverages quantitative performance data to guide optimisation efforts precisely where they will have the greatest impact. This approach enables teams to allocate computational and human resources effectively, prioritising efforts on components with the most significant performance impact. This focused approach avoids the common pitfall optimisation across all system components, which itself can create new forms of debt through unnecessary complexity.

Use of Automated Tools. Automated tools could play a vital role in mitigating AD in ML/DL systems development, similar to their role in managing TD in traditional software engineering. Participants in this study highlighted integrated development environments and specialised tools as possible mechanisms for reducing AD. For instance, one participant noted: *"Moreover, text editors like VSCode can come up with algorithms to detect these violations and suggest edits. Tools like docstring generators and code formatters can help senior developers adhere to best practices without significant effort on their part."* These tools, while not directly eliminating AD, streamline ML/DL development processes in several ways. First, they promote consistent adherence to best practices without requiring constant effort from developers. Code formatters (e.g., Black, YAPF) enforce uniform coding styles, which reduces cognitive overhead when dealing with complex algorithmic implementations in large, collaborative ML/DL projects. Hence, ML/DL-specific tools could be designed to address unique aspects of AD that traditional software tools cannot. For example, model validation frameworks can detect potential algorithmic inefficiencies while automated hyperparameter optimisation tools can prevent the accumulation of AD from suboptimal configurations.

6.2. Questionnaire Findings

The questionnaire helped to quantify several mitigation strategies for addressing AD in ML/DL systems. The findings, summarised in Figure 12,

Table 2: Categorisation of Mitigation Strategies for AD

Team Factors	**Software Engineering Factors**	**ML/DL Pipeline**
Communication among team members	Design with dynamism	Continuous monitoring/training
Awareness of team capabilities	Clear understanding of project goals	Iterative optimisation
Collaboration with experts	Education and training	Use of automated tools
Domain-specific expertise	Identifying possible externalities	
Knowledge of ML/DL frameworks		

highlighted a range of technical and organisational approaches that are employed by practitioners to address AD.

Systemic testing practices emerged as the most widely suggested strategy by 70.7% of respondents (46 out of 65). This strong preference underscores the belief that rigorous testing (i.e, unit tests for model components, integration tests for pipelines, and validation against diverse datasets) is fundamental to identifying and mitigating AD early in the development lifecycle. Efficient resource allocation ranked second, with 61.5% (40 respondents) recognising that many AD issues stem from resource constraints, such as limited access to GPUs or insufficient computational infrastructure (RQ1).

Education and training was selected by 60% (39 respondents), emphasising the importance of educating practitioners early on given the complexity of ML/DL algorithms. Improved team communication was noted by 52.3% (34 respondents), indicating that practitioners view AD as not only a technical challenge but also an organisational one, where enhanced collaboration can lead to misalignments in requirements or implementation. Finally, algorithm-specific interventions, such as refactoring algorithms and code reviews, were suggested by 44.6% (29 respondents), reflecting support for directly addressing AD through targeted improvements to algorithmic design, such as optimising hyperparameters or restructuring neural network architectures.

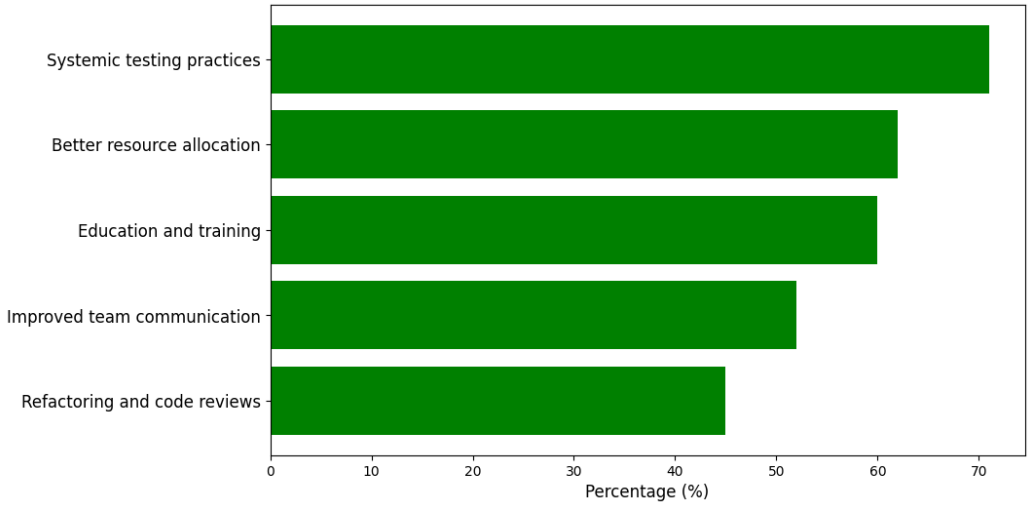

Figure 12: Mitigation Strategies for AD

6.3. Integrated Insights

Both the questionnaire and interviews emphasised improved team communication, with the questionnaire reporting it at 52.3% (34 respondents) and interviews elaborating on its role in preventing integration failures, as a participant noted, *"If we could have the communication with other developers, we can have more experience in the integration test for our models."* This alignment underscores communication's dual role as a technical and organisational strategy [33].

Systemic testing practices, the top questionnaire strategy at 70.1% (46 respondents), aligned with the interview theme of continuous monitoring and model retraining, where a participant stated, "You need to continuously monitor the performance criteria you continue testing and monitoring." Both highlighted the importance of proactive evaluation to identify AD early [49]. Efficient resource allocation (questionnaire, 62%) aligned with the interview discussions on awareness of team capabilities, with a participant noting, "If you know your strengths, and weaknesses, you will know exactly what you can do". This highlighted that with realistic planning, developers can know what constraints they will face while developing their systems [21].

Interviews provided deeper context, such as collaboration with experts (*"When you collaborate, you come with different viewpoints"*) and use of automated tools ("*Text editors like VSCode can suggest edits*") was also sug-

gested by the participants in the questionnaire, emphasising the need for tool-based approaches. Conversely, the questionnaire's suggestion for algorithm refactoring and code reviews (45%) added specific practices not detailed in interviews, highlighting their importance.

Table 3: Categorisation of Mitigation Strategies for AD

Team Factors	**Software Engineering**	**ML/DL Pipeline**
Improved communication	Design with dynamism	Continuous monitoring/training
Awareness of team capabilities	Clear understanding of project goals	Iterative optimisation
Collaboration with experts	Education and training	Use of automated tools
Domain-specific expertise	Identifying possible externalities	Adhering to industry standards
Knowledge of ML/DL frameworks	Systemic testing	
	Refactoring algorithms	

7. Discussion

7.1. Interpretation of Findings

Causes of AD. For RQ1 (causes of AD), from the questionnaire we identified lack of knowledge as the predominant cause of AD (72.3%, 47 respondents). This was further explained in the interviews where participants noted, "*Lack of knowledge will allow you to only use what you know and what you know might not be enough.*" This highlighted a gap in expertise, particularly in leveraging advanced ML/DL frameworks like PyTorch or TensorFlow, which can lead to suboptimal implementations and long-term inefficiencies. Technical factors like hardware constraints (39.1% often/always) and interview themes such as data drift further underscored the interplay between human and technical challenges, where resource limitations and evolving data distributions contribute to AD.

A deeper analysis of the questionnaire data by role (Section 4.2.2) revealed how causes of AD are different across roles, adding nuance to the overall findings for RQ1. Lack of developer knowledge was predominant, with Builders/Engineers reporting the highest concern (90%, 9/10), possibly due to their role requiring practical implementation of complex models without deep theoretical understanding, highlighted in the interviews.

Tight deadline was also a significant cause for Builders/Engineers (80%, 8/10) and Researchers (75.9%, 22/29), but less for Model Analysts (33.3%, 2/6), suggesting that time pressures are more important in roles involving development and research rather than analysis. These role-specific differences highlight the need for tailored strategies to address AD, as the challenges faced by Researchers differ from those of Model Developers.

Effects of AD. For RQ2 (effects of AD), poor model scalability was the dominant effect (73.8%, 48 respondents), with interviews providing context: *"But when it comes to implementing the same algorithm on a large-scale dataset, it might not really work because the underlying data structures and algorithm that you're using are not optimal, not time-efficient."* This suggested that AD could hinder the ability of ML/DL systems to handle increased computational demands, a critical issue in production environments.

The effects such as decreased model accuracy (60%) and performance degradation due to data drift (interviews) indicated that AD not only affects scalability but also undermines the reliability of ML/DL systems when deployed. AD issues related to performance degradation is pronounced in DL domains such as NLP, where external factors like evolving user behavior, seasonal trends, or shifts in domain-specific terminology could lead to a change in data distributions. For example, a sentiment analysis model trained on a dataset of product reviews from 2019 may perform poorly on reviews from 2023 if the language used by customers has significantly evolved.

Mitigation strategies for AD. For RQ3 (mitigation strategies), systematic testing practices was the most adopted mitigation strategy (questionnaire, 70.7%, 46 respondents), aligning with the interview emphasis on continuous monitoring: *"You need to continuously monitor the performance criteria."* The convergence between testing and monitoring reflected a shared recognition that proactive evaluation is essential to detect and address AD early, preventing its compounding effects. Also, improved team communication (questionnaire, 52.3% and the interview insights) further highlighted the dual

technical-organisational nature of AD mitigation strategies, where continuous model monitoring through testing and improved team communication could mitigate the effects of AD.

Lack of knowledge and expertise was among top causes (i.e, 60% respondents), and practitioners suggested the need for education and training. To address expertise gaps, one participant said, *"Regular training on frameworks like TensorFlow or PyTorch helps us stay updated and reduces AD by enabling better algorithmic choices."* Training enhances proficiency in advanced ML/DL tools, equipping developers to implement efficient solutions such as distributed training and optimised hyperparameter tuning. For example, leveraging PyTorch's DistributedDataParallel (i.e., a more efficient approach for scaling training across multiple GPUs) or TensorFlow's TPUStrategy (i.e., an API for distributing training across multiple GPUs, machines, or TPUs) can improve computational efficiency and scalability.

Also, educating and training practitioners on modern optimisation techniques (e.g., AdamW over SGD) can prevent the long-term inefficiencies noted in RQ2, directly addressing effects like increased maintenance costs and reduced developer productivity while addressing AD. This strategy is beneficial for ML/DL Builders/Engineers, who encounter implementation challenges such as managing complex data pipelines, optimising hyperparameters which are contributors to AD. By improving proficiency in modern frameworks and techniques, developers can reduce suboptimal implementations that contribute to AD, such as redundant model re-training, inefficient resource allocation, or technical overhead in scaling models.

Interrelationships among AD factors. Our study highlighted complex interrelationships among the causes, effects, and mitigation strategies for, which future research should explore to inform strategic planning. Resource availability interacts with organisational priorities, as budget constraints could force trade-offs between rapid deployment and AD mitigation, amplifying scalability limitations. Tool support also mediates the impact of organisational priorities on developer expertise: without automated tools, developers under time pressure resort to quick fixes (e.g., temporary data patches), which exacerbate model degradation. These interrelationships suggest that addressing AD requires a holistic strategy balancing expertise, resource allocation, and tool investment

Comparison with Previous Works. Our findings both aligned and extended prior research on TD. Prior studies [15, 26], focused on code (e.g., poor de-

sign) and its management via refactoring. In contrast, our work emphasised algorithmic choices, with complexity of ML/DL algorithms as the primary cause (RQ1). Likewise, while Sculley et al. [43] identified ML-specific TD (i.e., data dependencies and glue code) AD is focused on algorithmic issues such as hyperparameter optimisation. This distinction underscored the emphasis of AD on algorithmic efficiency rather than code structure.

Furthermore, prior works Recupito et al. [39], have examined TD in ML systems, primarily addressing Architecture and Code Debt. Our findings contribute to this discourse by demonstrating that AD impacts model scalability. Moreover, while previous research has suggested mitigation strategies, such as technical reviews and documentation [27], our study identified AD-specific mitigation techniques, including systemic testing and continuous monitoring (RQ3), tailored to the iterative nature of ML/DL development.

Regarding the causes of AD, our study introduced novel insights not previously identified in Documentation or Requirements Debt studies [10]. For instance, complexity of ML/DL algorithms (78%), model training issues (e.g., suboptimal feature selection), and GPU/hardware constraints (39.1% often/always) emerged as contributors to AD (RQ1). These factors were not highlighted in Rios et al. [40] or Barbosa et al. [10]'s focus on Documentation and Requirements Debt respectively, further distinguishing AD from broader TD categories. However, AD shared causes like tight deadline that was identified in previous TD studies [5].

Similarly, studies on Documentation Debt [40], identified reduced developer productivity as a shared effect with AD, reflecting how knowledge gaps (RQ1) could hinder efficiency in both contexts. However, while they noted loss of low external quality and lack of market competitiveness, as some of the effects of Documentation Debt, this was not a major issue for AD in our findings. Instead, the effects of AD (RQ2) primarily included challenges like poor model scalability (73.8%, 48/65), model bias, and performance degradation due to data drift (interviews). These AD-specific effects underscore AD as a unique TD type, impacting system reliability rather than stakeholder trust and code quality alone.

For mitigation strategies (RQ3), systemic testing (70.7%, 46/65) and continuous monitoring aligned with Amershi et al. [7]'s ML practices. However, our finding that education and training (60%, 39/65) are important to address knowledge gaps diverged from Rios et al. [40]'s documentation-centric solutions. This suggested that AD mitigation requires a broader skill-development approach beyond traditional TD management techniques.

The view of AD as potentially beneficial builds on the definition of TD [15] as a trade-off between short-term gains and long-term costs. For instance, suboptimal algorithm choices might accelerate initial development, accepting that it might introduce scalability challenges over time. Future research should explore how organisations balance the intentional accumulation of AD with long-term technical sustainability, in production ML/DL systems.

Surprisingly, one developer offered a counterintuitive perspective, stating that AD is not entirely a bad thing as it can help produce a better system. This suggested that incurring AD, could lead to iterative improvements of ML/DL systems by allowing teams to deploy initial models quickly, gather real-world feedback, and refining the systems over time. This aligns with the concept of intentional TD, as described in Fowler [18]'s TD quadrant, where teams knowingly take on debt with an intention to address it later. Additionally, [26] highlighted that TD is not inherently negative but required careful management to mitigate its long-term consequences. For example, an initial suboptimal implementation might reveal performance bottlenecks that, once addressed, lead to a more robust system. This perspective underscored the importance of managing AD effectively through strategies like continuous monitoring and iterative optimisation to ensure the potential benefits of AD are realised.

Limitations. We acknowledge several limitations in our work to guide future research. First, our participant sample, consisting of 65 questionnaire respondents and 21 interviewees, may not fully represent the diversity of ML/DL practitioners. The predominance of researchers (44.6%, 29 respondents), raised concerns about potential bias, as researchers may prioritise theoretical aspects of AD (e.g., model accuracy) over production-focused issues (e.g., scalability), potentially under representing the challenges faced by industry practitioners in large-scale production environments. Future studies should address this by including a more balanced sample of participants from both academic and industry backgrounds.

Second, the decision to send a link to the participants we interviewed, directing them to answer the questionnaire, introduced a limitation that may have compromised the independence of the data sources in the mixed-methods design. Since the interviews were conducted first, the 21 interviewees who possibly later completed the questionnaire may have been influenced by discussions we had, potentially biasing the quantitative findings. Future studies should be conducted and ensure stricter separation between

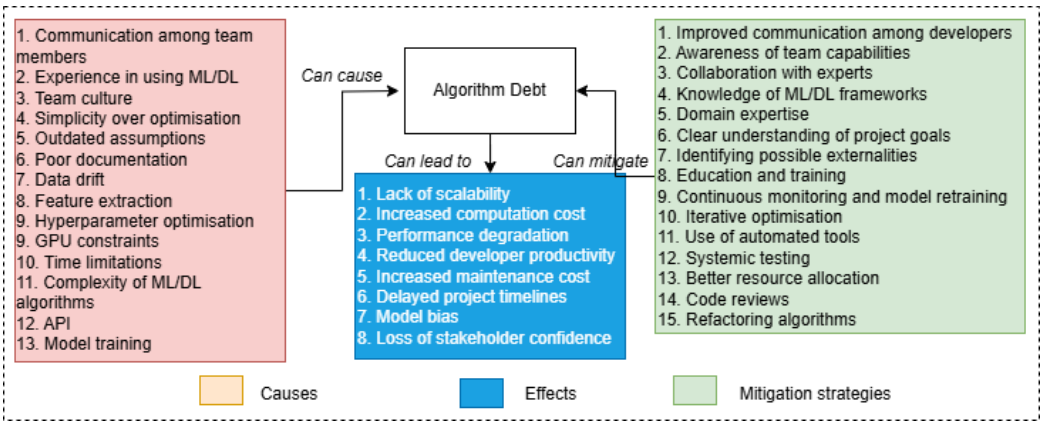

Figure 13: Theoretical framework of AD

questionnaire and interview participants or control for the sequence of data collection to maintain methodological rigor.

Finally, our analysis of AD causes by role is also limited by the small sample sizes for certain roles, such as Model Analysts (6 respondents). These small sample sizes may reduce the reliability of the role-specific findings, as the percentages may be overly sensitive to individual responses (e.g., a single response in the others category shifts the percentage by 20%). With this limitation the role-specific differences observed, such as the low concern for tight deadlines (33.3%, 2/6) by Model Analysts should be interpreted with caution. Future studies should be conducted with larger samples across roles to validate these trends. For example Model analysts could be interviewed to understand the causes, effects, and mitigation strategies specific to their role, providing more empirical evidence.

7.2. Theoretical framework

To synthesise the findings from our RQs, we proposed a theoretical framework for AD in ML/DL systems, illustrated in Figure 13. This framework provides a visual representation of the relationships between AD, its causes, its effects, and the strategies to mitigate it. This framework offers a model that integrates the empirical insights from our study.

At the center of the framework is AD, representing the accumulated debt in ML/DL systems due to suboptimal algorithmic choices. On the left, the "Causes" box includes the primary factors identified in RQ1. The "Effects" box below captured the effects of AD identified in RQ2. An arrow from AD to

Effects illustrated how AD leads to these outcomes, impacting both system performance and team efficiency. Finally, the "Mitigation Strategies" box included the strategies identified in RQ3. An arrow from Mitigation Strategies to AD indicated that these strategies can reduce AD and, consequently, its effects, with the potential benefits through strategic management, as one developer noted: *"AD is not totally a bad thing as it helps to produce a better system".*

This framework extended existing TD models, such as Rios et al. [40]'s Documentation Debt, by isolating AD as a distinct phenomenon in ML/DL systems and providing a structured representation of its dynamics. It offered both theoretical and practical value: theoretically, it provides a foundation for future research to explore domain-specific AD (e.g., in NLP vs. Computer Vision). Practically, this can guide practitioners to identify tailored causes, anticipating its effects, and implementation of targeted mitigation strategies for AD to support proactive management of AD.

8. Future Work

Our findings highlighted several directions for future research. First, the predominance of researchers in the sample (44.6%, 29 respondents) (Section 3.4.4) suggested a need for studies with more balanced samples. Future research could target industry-focused ML/DL practitioners to explore how AD manifests in large-scale deployment contexts, where scalability and operational efficiency are critical. This could involve case studies in industry settings to capture real-world challenges, such as managing AD in cloud-based ML pipelines or edge device deployments.

Specifically, the finding that 72.3% of survey respondents (47 out of 65) identified lack of knowledge and proficiency in ML/DL frameworks as the primary cause of AD represents a critical insight into the practical challenges of ML/DL development. This quantitative result, corroborated by interview themes highlighting practitioners' struggles with complex frameworks like TensorFlow or PyTorch, underscores a pervasive expertise gap in software engineering. For instance, interviewees frequently noted difficulties in selecting appropriate optimisers or tuning hyperparameters, often resorting to default configurations that led to suboptimal models. This aligns with prior literature noting ML/DL's steep learning curve [34], but extends it by framing knowledge deficiencies as a dominant AD driver, filling the practical gap identified. The implication that more expertise is needed suggests tar-

geted mitigation strategies, such as enhanced training programs or standardised documentation, to reduce AD's downstream effects, like performance degradation. Notably, role-specific differences—researchers emphasising algorithm complexity, engineers citing process issues, highlight the need for role-tailored education. This finding not only clarifies TD's manifestation in ML/DL systems but also underscores the urgency of addressing human factors in software engineering, setting the stage for automated detection tools to complement expertise-driven solutions.

Thirdly, the limited awareness of AD among participants (50% not or slightly familiar discussed in Section 3.4.4) and the reliance on self-reported data suggested a need for more objective measurement methods. Future studies could develop AD-specific metrics and tools, such as automated detectors for algorithmic inefficiencies (e.g., suboptimal tensor operations), building on the these findings. Longitudinal studies tracking AD accumulation in ML/DL projects could also provide more reliable data, mitigating biases inherent in self-reported responses.

Fourthly, lack of domain-specific analysis (Section 4) highlighted the need for research into how AD varies across ML/DL applications, such as NLP or Computer Vision. For example, the interview emphasis on domain-specific expertise suggested that AD mitigation strategies may need to be domain-specific. Comparative studies across domains could identify AD patterns across domain to develop tailored frameworks and solutions to mitigate AD.

The mitigation strategies identified (e.g., continuous monitoring (interviews) and systematic testing (questionnaire, 71%) calls for further investigation into their long-term effectiveness. In the future, experimental studies should be conducted to evaluate the effectiveness of these strategies to reduce AD. This could involve comparing projects with and without continuous monitoring to measure improvements in scalability and accuracy.

Finally, to explore the factors contributing to AD and their interrelationships, future work should conduct a mixed-methods study over a period of time, involving ML/DL organisations to validate these findings. A potential methodology is to use structural equation modeling to quantify the relationships between factors (e.g., expertise to tool support to AD mitigation effectiveness), complemented by longitudinal case studies tracking AD management outcomes (e.g., accuracy improvements) after implementing strategic interventions (e.g., training programs, tool adoption). Additionally, developing an open-source AD detection framework (e.g., adapting SonarQube for ML/DL smells) could address tool support gaps, enabling practitioners to

prioritise AD mitigation within constrained budgets and timelines. These efforts integrating insights from practitioner interviews, will refine strategic planning for AD management, enhancing ML/DL system reliability.

9. Threats to Validity

9.1. Internal Validity

The thematic analysis of the interview posed threats to the validity of our research. Bias in the coding by the first author may have introduced subjective interpretations of the responses from participants. To mitigate this bias, we employed a systematic coding process (Section 3.3.3) and the result of the coding was reviewed through weekly meetings among all the authors to validate the thematic analysis to reduce bias.

The predominance of researchers (45%, 29 respondents), could raise sampling bias, as researchers may prioritise theoretical aspects of AD (e.g., model accuracy) over production-focused concerns (e.g., deployment scalability). However, the spread of the different ML/DL roles across industry mitigated the overall bias of responses from researchers.

9.2. External Validity

The recruitment process of the participants may have introduced selection bias, as participants were first recruited through professional networks. This could have excluded less connected practitioners, such as those in smaller organisations or developing regions, potentially skewing the findings (e.g., underrepresenting resource constraints as a cause of AD, questionnaire 39.1% often/always). However, recruiting from conferences and online platforms (X and LinkedIn), mitigated having access to only our professional networks, hence reaching a broader audience.

9.3. Construct Validity

A potential threat was from the limited awareness of AD among participants, with 50% of questionnaire respondents reporting that they were not or slightly familiar with the concept. This lack of awareness might have led to misinterpretation of AD-related questions, particularly for RQ1 (causes), where the participants might not have accurately identified AD-specific practices in their workflows. To address this, we provided definitions of AD in the questionnaire and interview guides. We also shared code snippets with explanations to the participants to mitigate the risk of wrong interpretation.

The questionnaire design could have presented threats to validity, since we adapted the design from the InsighTD survey [41]. While the InsighTD survey was designed to study TD in traditional software engineering contexts, our adaptation for AD in ML/DL systems may have introduced construct validity threats. This may have altered the validated structure of the InsighTD survey, affecting its reliability in this new context. However to mitigate this, we conducted pilot testing, to avoid the questions from been misinterpreted by the participants unfamiliar with AD.

10. Conclusion

In this study, we investigated AD in ML/DL systems through a concurrent mixed-methods design, combining 65 questionnaire responses and 21 semi-structured interviews collected independently. We investigated the causes, effects, and mitigation strategies for AD, offering both quantitative and qualitative depth.

We identified lack of knowledge of ML/DL frameworks as a major cause of AD with themes like team culture, highlighting the interplay of human and technical factors in AD accumulation. Another notable finding was the limited awareness of AD, among ML/DL practitioners with variations across roles. For effects of AD, poor model scalability was the most significant effect, while systemic testing was the top mitigation approach emphasising proactive solutions.

Our study contributes to software engineering by isolating AD as a distinct TD type, with a theoretical framework synthesising its causes, effects, and mitigation strategies. Practically, our study offered actionable guidance, such as the need for education and training of practitioners to address knowledge gaps to mitigate the impact of AD. The findings from our study provides a foundational insights on mitigating AD, while paving the way for future research to enhance the reliability of ML/DL systems.

11. Acknowledgment

This work was supported by the Australian National University (ANU) through the ANU PhD scholarship within the ANU Research School of Computing, ANU College of Systems and Society.

References

[1] A Ilemobayo, J., Durodola, O., Alade, O., J Awotunde, O., 2024. Hyperparameter tuning in machine learning: A comprehensive review. Journal of Engineering Research and Reports 26, 388–395.

[2] Aghajani, E., Nagy, C., Vega-Márquez, O.L., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., 2019. Software documentation issues unveiled, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE. pp. 1199–1210.

[3] Ahmad, M.O., Al-Baik, O., Hussein, A., Abu-Alhaija, M., 2025. Unraveling the organisational debt phenomenon in software companies. Computer Science and Information Systems , 12–12.

[4] Albelwi, S., Mahmood, A., 2017. A framework for designing the architectures of deep convolutional neural networks. Entropy 19, 242.

[5] Aldaeej, A., Alshayeb, M., 2024. Familiarity, common causes and effects of technical debt: A replicated study in the saudi software industry. Arabian Journal for Science and Engineering 49, 4459–4477.

[6] Alves, N.S., Ribeiro, L.F., Caires, V., Mendes, T.S., Spínola, R.O., 2014. Towards an ontology of terms on technical debt, in: 2014 sixth international workshop on managing technical debt, IEEE. pp. 1–7.

[7] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., 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.

[8] Arpteg, A., Brinne, B., Bosch, J., 2018. Software engineering challenges of deep learning, in: 2018 44th euromicro conference on software engineering and advanced applications (SEAA), IEEE. pp. 50–59.

[9] Ayyagari, M., Bhatt, V.D., Pandey, A., 2022. Efficient implementation of pooling operation for ai accelerators, in: 2022 IEEE International Conference for Women in Innovation, Technology & Entrepreneurship (ICWITE), IEEE. pp. 1–5.

[10] Barbosa, L., Freire, S., Rios, N., Ramač, R., Taušan, N., Pérez, B., Castellanos, C., Correal, D., Pacheco, A., López, G., et al., 2022. Organizing the td management landscape for requirements and requirements documentation debt. UMBC Faculty Collection .

[11] Baumann, N., Kusmenko, E., Ritz, J., Rumpe, B., Weber, M.B., 2022. Dynamic data management for continuous retraining, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 359–366.

[12] Besker, T., Martini, A., Bosch, J., 2018. Managing architectural technical debt: A unified model and systematic literature review. Journal of Systems and Software 135, 1–16.

[13] 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), IEEE. pp. 64–73.

[14] Braun, V., Clarke, V., 2006. Using thematic analysis in psychology. Qualitative research in psychology 3, 77–101.

[15] Cunningham, W., 1992. The wycash portfolio management system. ACM SIGPLAN OOPS Messenger 4, 29–30.

[16] Decker, T., Koebler, A., Lebacher, M., Thon, I., Tresp, V., Buettner, F., 2024. Explanatory model monitoring to understand the effects of feature shifts on performance, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 550–561.

[17] Fan, J., Han, F., Liu, H., 2014. Challenges of big data analysis. National science review 1, 293–314.

[18] Fowler, M., 2009. Technical debt quadrant. URL: `https://martinfowler.com/bliki/TechnicalDebtQuadrant.html`.

[19] Freire, S., Rios, N., Pérez, B., Castellanos, C., Correal, D., Ramač, R., Mandić, V., Taušan, N., 2024. Hearing the voice of software practitioners on technical debt monitoring: Understanding monitoring practices and the practices' avoidance reasons. Journal of Software Engineering Research & Development 12.

[20] Guo, X., Jiang, F., Chen, Q., Wang, Y., Sha, K., Chen, J., 2025. Deep learning-enhanced environment perception for autonomous driving: Md-net with csp-darknet53. Pattern Recognition 160, 111174.

[21] Hill, T., Westbrook, R., 1997. Swot analysis: It's time for a product recall, long range planning. S0024-6301 (96) , 00095–7.

[22] Janis, I.L., 1972. Victims of groupthink: A psychological study of foreign-policy decisions and fiascoes. Houghton Mifflin.

[23] Jepkoech, J., Mugo, D.M., Kenduiywo, B.K., Too, E.C., 2021. The effect of adaptive learning rate on the accuracy of neural networks. International Journal of Advanced Computer Science and Applications 12.

[24] Johnson, R.B., Onwuegbuzie, A.J., Turner, L.A., 2007. Toward a definition of mixed methods research. Journal of mixed methods research 1, 112–133.

[25] Khan, S., Yairi, T., 2018. A review on the application of deep learning in system health management. Mechanical Systems and Signal Processing 107, 241–265.

[26] Kruchten, P., Nord, R.L., Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. Ieee software 29, 18–21.

[27] Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. Journal of Systems and Software 101, 193–220.

[28] 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: ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society, pp. 1–10.

[29] Lohoff, J., Neftci, E., 2024. Optimizing automatic differentiation with deep reinforcement learning. arXiv preprint arXiv:2406.05027 .

[30] Makkouk, T., 2023. Investigating and Testing Performance Issues in Deep Learning Frameworks. Ph.D. thesis. Concordia University.

[31] Mallick, A., Hsieh, K., Arzani, B., Joshi, G., 2022. Matchmaker: Data drift mitigation in machine learning for large-scale systems. Proceedings of Machine Learning and Systems 4, 77–94.

[32] Mishra, S., 2019. A distributed training approach to scale deep learning to massive datasets. Distributed Learning and Broad Applications in Scientific Research 5.

[33] Nahar, N., Zhou, S., Lewis, G., Kästner, C., 2022. Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process, in: Proceedings of the 44th international conference on software engineering, pp. 413–425.

[34] OBrien, D., Biswas, S., Imtiaz, S., Abdalkareem, R., Shihab, E., Rajan, H., 2022. 23 shades of self-admitted technical debt: an empirical study on machine learning software, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 734–746.

[35] Onwuegbuzie, A.J., Johnson, R.B., 2021. Mapping the emerging landscape of mixed analysis, in: The Routledge reviewer's guide to mixed methods analysis. Routledge, pp. 1–22.

[36] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, 2019. Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, pp. 8024–8035.

[37] Patton, M.Q., 2003. Qualitative Evaluation and Research Methods. 3rd ed., Sage Publications.

[38] Pepe, F., Zampetti, F., Mastropaolo, A., Bavota, G., Di Penta, M., 2024. A taxonomy of self-admitted technical debt in deep learning systems, in: 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 388–399.

[39] Recupito, G., Pecorelli, F., Catolino, G., Lenarduzzi, V., Taibi, D., Di Nucci, D., Palomba, F., 2024. Technical debt in ai-enabled systems: On the prevalence, severity, impact, and management strategies for code and architecture. Journal of Systems and Software , 112151.

[40] Rios, N., Mendes, L., Cerdeiral, C., Magalhães, A.P.F., Perez, B., Correal, D., 2020. Hearing the voice of software practitioners on causes, effects, and practices to deal with documentation debt, in: Requirements Engineering: Foundation for Software Quality: 26th International Working Conference, REFSQ 2020, Springer. pp. 55–70.

[41] Rios, N., Spínola, R.O., Mendonça, M., Seaman, C., 2018. The most common causes and effects of technical debt: first results from a global family of industrial surveys, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10.

[42] Salamkar, M.A., Immaneni, J., 2021. Automated data pipeline creation: Leveraging ml algorithms to design and optimize data pipelines. Journal of AI-Assisted Scientific Discovery 1, 230–250.

[43] 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. Advances in neural information processing systems 28.

[44] Simon, E.I.O., Vidoni, M., Fard, F.H., 2023. Algorithm debt: Challenges and future paths, in: 2023 IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN), IEEE. pp. 90–91.

[45] Singh, N., Shrivastava, J.N., Agarwal, G., Sanghi, A., Jha, S., 2025. Enhancing human-computer interaction through artificial intelligence and machine learning: A comprehensive review. Navigating Cyber-Physical Systems With Cutting-Edge Technologies , 235–256.

[46] Sklavenitis, D., Kalles, D., 2024. Measuring technical debt in ai-based competition platforms, in: Proceedings of the 13th Hellenic Conference on Artificial Intelligence, pp. 1–10.

[47] Souza, L., Freire, S., Rocha, V., Rios, N., Spínola, R.O., Mendonça, M., 2020. Using surveys to build-up empirical evidence on test-related technical debt, in: Proceedings of the XXXIV Brazilian Symposium on Software Engineering, pp. 750–759.

[48] Tan, M., Le, Q., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks, in: International conference on machine learning, PMLR. pp. 6105–6114.

[49] Tatineni, S., Katari, A., 2021. Advanced ai-driven techniques for integrating devops and mlops: Enhancing continuous integration, deployment, and monitoring in machine learning projects. Journal of Science & Technology 2, 68–98.

[50] Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. Journal of Systems and Software 86, 1498–1516.

[51] Wang, X., Schuster, H., Borrison, R., Klöpper, B., 2023. Technical debt management in industrial ml-state of practice and management model proposal, in: 2023 IEEE 21st International Conference on Industrial Informatics (INDIN), IEEE. pp. 1–9.

[52] Xiao, W., Xue, J., Miao, Y., Li, Z., Chen, C., Wu, M., Li, W., Zhou, L., 2020. Distributed graph computation meets machine learning. IEEE Transactions on Parallel and Distributed Systems 31, 1588–1604.

[53] Ximenes, R.G., 2024. Issues that Lead to Code Technical Debt in Machine Learning Systems. Master's dissertation. Pontifical Catholic University of Rio de Janeiro (PUC-Rio).

[54] Zhang, C., Patras, P., Haddadi, H., 2019. Deep learning in mobile and wireless networking: A survey. IEEE Communications surveys & tutorials 21, 2224–2287.

[55] Zhou, H., Gao, B., Tang, S., 2025. Intelligent detection on construction project contract missing clauses based on deep learning and nlp. Engineering, Construction and Architectural Management 32, 1546–1580.