International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Technical Debt in the Age of Artificial Intelligence and Methods of Its Forecasting

Satyashil Awadhare

Engineering Lead at Google, Burlingame, California, USA

Abstract: ***The article examines the transformation of the technical debt concept amid the rapid proliferation of artificial intelligence and analyzes methods for its forecasting. The aim is to identify the characteristics of the accumulation and evolution of technical debt in AI systems and to develop approaches for its measurement and strategic management. The relevance stems from the fact that technical debt has ceased to be a local engineering problem and has acquired macroeconomic and organizational significance, affecting company productivity, the resilience of business processes, and compliance with regulatory requirements. The novelty of the study is in organizing a multi-layer structure of debt obligations, not by including only classical code and architectural elements, but by adding new layers—data debt, model debt, pipeline debt, as well as regulatory and workforce costs. The article presents an approach to forecasting debt risks, hybridizing static metrics, machine classifiers, graph dependency models, time series of operational metrics, and scenario modeling with large language models. A major takeaway is that AI technical debt can only be effectively managed once a full account of all its layers has been brought into consideration and their linkages to business metrics, plus regular monitoring and forecasting, are institutionalized. In practical terms, turning the debt from being just an abstract metaphor into a manageable asset reduces interest expense while, at the same time, increasing the resilience of technological development. The article will be helpful to researchers in software engineering, machine learning practitioners, IT system architects, and leaders of digital transformations.***

Keywords: technical debt, artificial intelligence, forecasting, data debt, model debt, pipeline debt, ML-Ops, risk management

1. Introduction

Technical debt has long ceased to be merely an engineering metaphor: according to McKinsey, CIOs of large companies believe it eats up 20–40% of the total value of their technology portfolio and adds another 10–20% to the cost of each new project, effectively acting as a tax on development [1]. At the economy-wide level, the problem takes on macro-financial contours: the Consortium for Information & Software Quality has estimated that the total principal of outstanding debt in the United States reached $1.52 trillion, and aggregate losses from poor software quality exceeded $2.4 trillion by the end of 2022 [2]. These figures demonstrate that we are not dealing with a secondary engineering concern but with a factor that directly affects businesses' strategic indicators and national productivity.

The term technical debt was coined by Ward Cunningham when describing WyCash trade-offs at OOPSLA-1992: "by accelerating today, we take a loan for future rework and pay interest in the form of rising maintenance costs" [3]. The original idea concerned mainly application code and architecture, but it quickly evolved: debt came to include outdated libraries, a lack of tests, and even organizational processes. In the classical world, these risks were relatively predictable: system complexity grew linearly, and the release lifecycle was measured in quarters.

The age of artificial intelligence radically changes how interest is calculated. Machine learning has added at least two new layers of debt—data debt and model debt—each with its accumulation mechanisms. Google's Hidden Technical Debt in Machine Learning Systems showed that an ML product has far more hidden dependencies than traditional software: data drift, model staleness, irreproducible experiments, and ML-Ops pipelines that rapidly become brittle graphs of interdependencies [4]. The industry's accelerating pace throws fuel on the fire. According to AI Index 2025, compute used to train frontier models now doubles every five months. The share of organizations deploying AI grew from 55% to 78% in just one year, making technical debt the norm of rapid scaling rather than an exception [5].

Such an environment makes forecasting debt integral to the technology strategy. The goals of this article are twofold: first, a systematization of ways AI ingredients make the classical concept of technical debt more complex; and second, a review of the modern toolkit available for its forecasting, from static analysis and ML classifiers to graph models and LLM-based simulations.

2. Materials and Methodology

This study draws on a broad review of scholarly articles, market studies, and legislative files aimed at the development of the technical debt notion from classic engineering strategies to the era of AI. The conceptual base involves essential works beginning with Ward Cunningham's explanation of creation trade-offs at OOPSLA-1992 [3], as well as modern McKinsey analysis on the Debt Matters for Corporate Portfolios [1] and macroeconomic scale of losses conveyed by the Consortium for Information & Software Quality [2]. As the practical base, we have used analytical information on covert layers of liability associations in machine learning from Google's Hidden Technical Debt in Machine Learning Systems, and statistics from the AI Index 2025 showing facts about the dynamics of AI usage and compute growth.

This article methodologically triangulates three classic paths of analysis. First, it undertakes a scholarly and applied literature comparative review that traces the evolution of code-centric notions to a multilayer structure comprising data debt, model debt, pipeline debt, and regulatory plus workforce costs. The second path involves a systematic analysis of industry surveys and reports, wherein The Morning Consult

and Unqork Study directly reveal how accumulating debt impacts innovation activity in companies. At the same time, Gartner data validates economic loss attributable to the poor quality of data. Thirdly, results from scholarly reviews on the reproducibility crisis in ML experiments were incorporated to help expose gaps in methods for long-term upkeep of models. This was supplemented with case studies about getting DevOps integrated with MLOps, which revealed that the absence of unified versioning practices adds up to pipeline debt [9].

Focus was set on the regulatory dimension: an analysis of the European AI Act [10] synthesized ethical and legal obligations with the financial penalties on organizations. The methodological framework is furnished with content analysis of the labor market from IT Pro and 365 Data Science material, which has evidenced how a shortage of talent catalyzes the growth of people debt [11, 12].

3. Results and Discussion

In classical software, technical debt accumulates primarily in code and architecture, but in AI systems, it spreads across a whole spectrum of new layers. At the level of source code and microservice architecture, generative development and fast releases create blind dependencies: a component diagram loses relevance after just a few sprints, and a single fix deep in the stack triggers a cascade of testing. This is where the bulk of debt lurks: a survey of 500 technology leaders found that 92% of organizations acknowledge significant technical debt, and 80% canceled or froze mission-critical projects over the past year because of it, as shown in Fig. 1 [6].

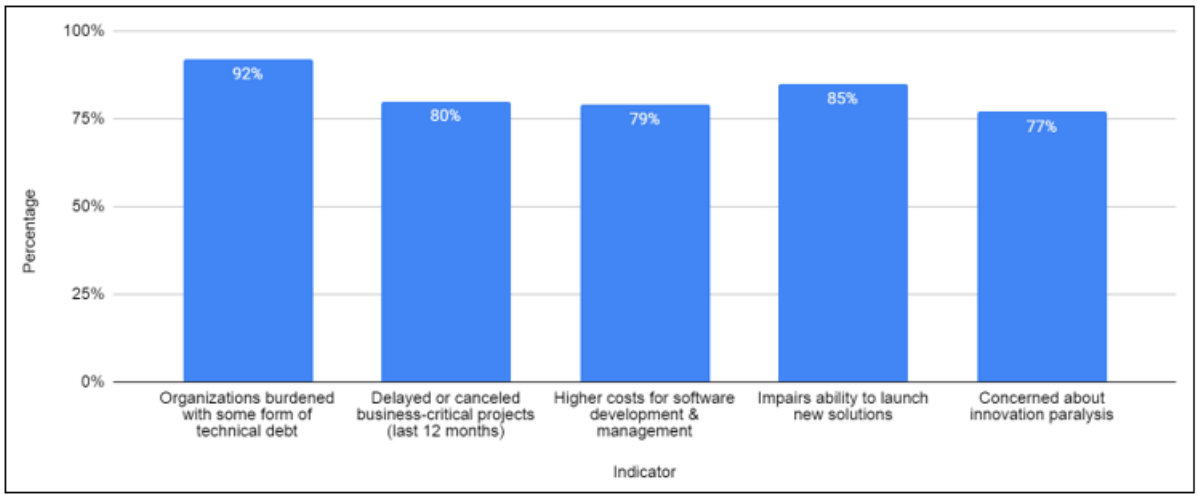

Figure 1: Quantifying the Organizational Impact of Technical Debt [6]

Data debt grows even faster. Data collected for one business task drifts, loses completeness, or falls out of schema currency, turning each model retraining into a costly operation. Gartner estimates direct losses from poor data quality at an average of $12.9 million per company per year, and an additional study showed up to a 20% drop in productivity and a 30% rise in operating costs [7].

Stacked on top is model debt. No amount of perfectly cleaned data will ever avail if the model becomes stale or its experiments cannot be reproduced. In a review encompassing 101 scholarly works on long-term model maintainability, reproducibility reports that only 12.9% of papers might have addressed the problem, thereby confirming systematic underestimation of the situation as depicted in Fig. 2 [8]. And then there's parameter drift: the more actively data and requests change under the system, the faster the interest rate on such debt grows.

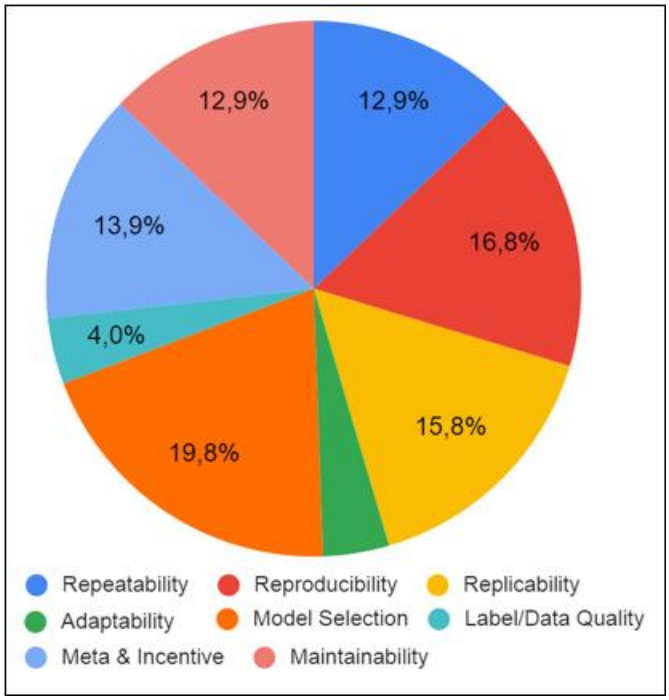

Figure 2: Prioritizing Reproducibility and Model Selection Challenges in ML Research [8]

The next layer is pipeline debt or ML-Ops debt. The absence of unified versioning practices for data, models, and

infrastructure artifacts turns the pipeline into a brittle graph: TechRadar notes that 85% of trained models never reach production due to gaps between DevOps and MLOps [9]. Every manual step in such a process is a potential source of new debt and of release delays.

To this, it is added ethical and regulatory debt. The European AI Act introduces fines of up to €35 million or 7% of a company's global turnover for the use of opaque or discriminatory systems. In comparison, lesser violations are still penalized up to €15 million [10]. Thus, each unimplemented explainability procedure or bias audit automatically increases potential costs.

Finally, there is people debt—the competence deficit required to service all the layers above. In the United Kingdom alone, there are over 11,000 open positions, and 69% of them are tied to AI skills [11]; globally, the ML engineering job market is valued at $113 billion with about 1.6 million specialists and steady annual growth, as shown in Fig. 3 [12]. A shortage of such talent lengthens the time to retire debt: legacy code and pipelines go without proper refactoring, and interest payments—in the form of rising maintenance costs—continue to accrue.

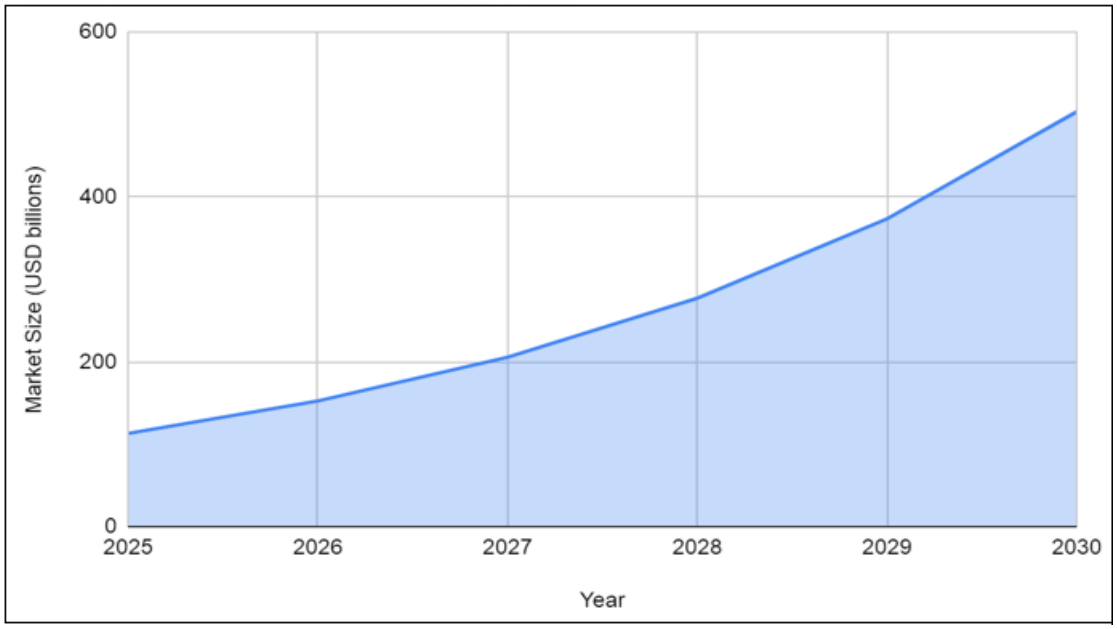

Figure 3: ML Engineering Market Growth [12]

Technical debt in AI systems is, therefore, multi-layer, mixing classic code and architectural obligations with data, model, pipeline, regulatory, and human debts. The better each layer is understood and how they mutually influence the formulation of an interest rate for debt across the product lifecycle (i.e., the better one understands each layer and its mutual influence on the formulation of an interest rate for debt across the product lifecycle), then more precise forecasting and management of the overall interest rate of debt can be.

Once the multilayered nature of technical debt in AI systems is understood, the main work is to forecast its growth so that resources can be shifted before the interest rate goes wild. The easiest way is to bet on static code metrics: complexity, duplication, and smell density. Present-day linters run nonstop, checking pull requests and providing an estimated price for defect fixing right in the build pipeline. There is practically no setup for this approach, and it works well in a CI culture. However, it pays off only one layer of debt and cannot capture the dynamics that are related to models and data.

Signals shall become much richer out of models that shall be trained on repository history. They may take a look at how files evolve, the tone of commit messages, changes in complexity metrics, and provide a probability that this piece of code will need refactoring in the next sprint. Just like traditional classifiers, these machine classifiers can give the team an early warning about hot components so that they can plan proactively, too. The downside to using such approaches is that they require training with labeled datasets and a consistent taxonomy of debt tickets.

Where explainability is more important than accuracy, use graph dependency models. Represent as nodes the modules, the services, even data features, and as edges their relationships: centrality computation, cluster density—hidden stress nodes. Topology changes signal rising brittleness long before a failure ever makes it to production.

DevOps and ML Metrics come in as multi-factor time series. Deployment Frequency, Mean Time to Incident, Feature Drift, and Model Quality metrics compose a multi-factor time series. Algorithms that are usually applied when loads are being forecast discover trend and seasonality, allowing for an estimate of just how quickly technical debt will erode the delivery speed or prediction accuracy.

In the end, scenario crafting with large language models turns evaluation into an active conversation. A helper can run experiments, build information, simulate model generation, and judge the impact of alternative compositional choices. It forecasts both dimensions of the liability and which alterations will reduce it--at least as far as cost. When used

together, they help teams treat debt not as an unavoidable cost of speed but as a type of asset that can be measured and made the subject of strategy.

A great forecast begins with an inventory. The team expresses each liability by layer-code and architecture, data, models, pipelines, and organizational bottlenecks. Mapping on the spot builds vocabulary in common between different engineers' observations, not to mention data analysts and product managers, in a unified typology ready for analysis and planning. Once entities are recorded, debt gets names and priorities with firm links to business metrics, and the layer that is threatening delivery speed or regulatory compliance in the near term becomes apparent.

Proceed to collect Static Code metrics, Data Drift, Model Accuracy Curves, Release Frequency, and Incident Duration in one central repository. Remove differences in scale and periodicity. There is no absolute value; use relative values, rare spikes smoothed by window median, and bring heterogeneous data to comparable scales. At this level, technical debt stops being a metaphor and becomes a set of observable telemetry signals.

Forecasting horizons, over the short horizon—one sprint to a few weeks—linter heuristics and fast classifiers work well, having thrown up warnings of overheating in a particular file or model. The medium-term horizon is several quarters away and relies on DevOps time series plus graph analytics: it picks up trend, picks up seasonality, and picks up slow coarsening of architecture. The long horizon is scenario modeling with language-model copilots at hand supporting teams to play out the effects of their strategic decisions, say by moving onto a new framework version or changes in regulatory requirements, and which reveal where debt risks are getting out of control. No forecast can be complete without ground truth; therefore, continuous verification: compare predicted costs with actual tickets; monitor deviation; retrain models automatically. Such a cycle keeps the prediction mechanism current, makes it sensitive to contextual shifts, and gradually reduces the influence of human assumptions. As a result, technical-debt management transforms from a one-off audit into an ongoing process embedded in the engineering organization’s culture.

To operationalize this multi-horizon approach, we propose a three-tiered forecasting methodology that integrates specific analytical techniques with corresponding organizational decision-making levels. This model imbues the forecasting process with a prescriptive character, transforming it from a set of observations into a framework for action. The first tier, Tactical Forecasting, addresses the short-term horizon of one to four weeks. Its primary objective is to inform sprint-level decisions and prevent the immediate accumulation of new debt. This is achieved by embedding static analysis tools, commit-level classifiers, and automated model quality checks directly into the CI/CD pipeline. The output consists of actionable alerts and automated quality gates within pull requests, generating a prioritized list of refactoring tasks for development teams to address in the current sprint.

The second tier, Operational Forecasting, shifts the focus to the medium-term horizon of one to two quarters, targeting systemic risks and quarterly planning. This level employs time-series analysis of DevOps and MLOps metrics—such as deployment frequency, mean time to recovery (MTTR), and model drift—alongside graph dependency analysis to uncover systemic brittleness. The resulting forecasts identify architectural "hotspots" and subsystems that necessitate larger, planned refactoring initiatives. For stakeholders such as product managers and system architects, these findings serve as an empirical basis for allocating resources and justifying dedicated technical improvement projects within the product roadmap.

The third tier, Strategic Forecasting, operates on a long-term horizon of one year or more, aligning technology evolution with overarching business objectives. It utilizes scenario modeling, often powered by large language models (LLMs), to evaluate the far-reaching consequences of major strategic decisions. These scenarios can simulate the impact of adopting a new AI framework, migrating to a different cloud provider, or adapting to significant regulatory shifts like the full implementation of the EU AI Act. The outcomes are strategic risk assessments, cost-benefit analyses for substantial technology investments, and a long-term technology health roadmap. This tier provides essential foresight for senior leadership, including the CTO and Head of Engineering, enabling them to navigate complex trade-offs and avert architectural dead ends.

This integrated, multi-tiered model ensures that technical debt is managed cohesively across all organizational levels. By systematically linking short-term code quality, medium-term system health, and long-term architectural viability, it transforms technical debt management from a reactive engineering concern into a proactive and continuous element of technology strategy.

Technical debt management delivers the benefit when its magnitude is expressed in terms that business stakeholders understand. Consequently, the team first establishes a direct link between each type of debt and key product indicators: model performance degradation is tied to conversion, data drift to prediction accuracy, and ethical risks to the cost of regulatory compliance. Translating engineering metrics into economic ones allows debt-repayment initiatives to be defended at the portfolio level, not only within technical discourse.

To make this linkage work, debt information is concentrated in a single ticket repository. Regardless of the source—static analysis, pipeline monitoring, or data audits—all records are created according to a unified schema: type, layer, impact on metrics, and estimated effort. The consolidated registry eliminates desynchronization across teams and serves as a training dataset for the forecasting models described in previous sections, which form the core of the analytical strategy.

On top of the unified catalog sits automatic debt-cost calculation. The CI system picks up cues from linters, ML numbers, and production-incident facts, mixes them with business scores, and provides a financial estimate directly in the task-control setup. Workers get a ranked list that shows

not just an unclear code issue but a clear cash danger that gets bigger with each day of delay.

To keep technical debt from becoming an endless list, teams set a regular interest payment. They use part of each sprint's capacity for remediation. This fixed budget is easy to communicate, disciplines planning, and prevents an avalanche of costs that often becomes visible only in late project stages.

However, establishing such a fixed budget is merely the first step in a complex organizational change. The practical implementation of a systematic debt management program faces significant hurdles, primarily concerning ownership and alignment. A central challenge lies in creating and maintaining a unified debt registry. This requires not only technological integration but also a cultural consensus on what constitutes debt and how its severity is measured, compelling engineering, data science, and product teams to agree on a shared taxonomy that links technical metrics to tangible business impact. The question of "who pays" for this debt extends beyond engineering capacity; it necessitates that product owners and business leaders actively participate in prioritizing debt remediation alongside new feature development. Achieving this requires the entire organization to be onboard, a goal typically unreachable without explicit executive sponsorship. Leadership, particularly the CTO and VP of Engineering, must champion this initiative by consistently framing investments in technical health not as an engineering cost center, but as a strategic imperative for mitigating risk and ensuring long-term product velocity.

Furthermore, the implementation of the multi-layered monitoring and forecasting system proposed herein carries its own non-trivial engineering and computational costs. Quantifying these costs precisely is challenging as they vary with organizational scale and system complexity, but they can be conceptually modeled. Engineering costs encompass the initial effort to instrument code, build data pipelines for telemetry collection, and develop the analytical models, as well as the sustained effort required for maintenance and interpretation. Computational costs include the infrastructure for real-time data ingestion, storage, and processing across all six debt layers—from code analysis to model performance tracking. While exact figures differ, industry analyses from cloud and observability providers suggest that comprehensive telemetry systems can consume between 5% and 15% of a total infrastructure budget for traditional software. For the multifaceted nature of AI systems, with their additional data and model-centric telemetry, this figure may represent a conservative baseline. This expenditure, however, should not be viewed as an operational expense but rather as a strategic investment. It provides the necessary intelligence to make informed capital allocation decisions, reducing the risk of catastrophic failures and managing the accrual of high-interest debt that could otherwise jeopardize future innovation.

Finally, as context accumulates, it becomes possible to refinance the debt—to change the remediation strategy to reduce the total cost of ownership. Sometimes it is more advantageous to rewrite a legacy module on a modern framework than to keep patching; sometimes it is worth investing in the collection of more representative data to avoid constant model retraining. A deliberate shift from quick-fix tactics to structural revision completes the management cycle and turns technical debt from a force of nature into a controlled instrument of development.

4. Conclusion

The article shows that technical debt has ceased to be a local engineering problem and has acquired a broad economic and organizational character: it affects not only code and architecture but also data, models, pipelines, regulatory requirements, and human resources. The era of artificial intelligence has given rise to two fundamentally new layers of debt: data debt and model debt. Also in this era, the interdependencies among layers have been so strengthened that classical mechanisms for accounting, managing, and repaying debts are inadequate. Therefore, delivery speed, prediction quality, and compliance with regulations become systemic risks as a result of unfulfilled debt obligations.

This also proved that only a hybrid multi-scale set of methods can come close to accurately predicting technical debt inside AI systems. Static metrics and linters for short horizons and at the code layer work perfectly well. Simple from the repository, what hot components are involved early in the process. At the same time, graph models expose hidden brittleness nodes in dependencies, while DevOps/ML time series express trend and seasonality. The best way to model long horizons is through scenarios made possible by large language models, where one can play out the effects of architectural and regulatory decisions.

The practical scheme of management draws from several key elements, such as the inventory by layers of debts and their mapping to business metrics; centralized collection and normalization of telemetry indicators ; unified ticket registry with single taxonomy ; automatic calculation of the monetary cost of debt; regimen regular interest payments in sprints, as well as the possibility to refinance debt positions.

It underscores the continuous verification whereby predictions have to be constantly matched with actual tickets and incidents, and forecasting models retrained accordingly on observed mismatches. Nothing but a consolidated cyclical approach that will infuse both the quantitative metrics and scenario modeling will transform technical debt from being perceived as just a burden into a manageable resource, minimize its interest payments, and ensure resilient product development under conditions of rapid AI-driven scaling.

References

[1] A. Baig, S. Blumberg, A. Gundurao, and B. Kayyali, "Tame tech debt to modernize your business," *McKinsey*, Apr. 25, 2023. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/breaking-technical-debts-vicious-cycle-to-modernize-your-business (accessed Jul. 19, 2025).
[2] H. Krasner, "Cost of Poor Software Quality in the U.S.: A 2022 Report," *CISQ*, Dec. 15, 2022. https://www.it-

cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/ (accessed Jul. 20, 2025).

[3] S. Fraser, D. Mancl, B. Opdyke, J. Bishop, J. Lacar, and A. Szynkarski, "Technical debt," *SPLASH '13: Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*, pp. 67–70, Oct. 2013, doi: https://doi.org/10.1145/2508075.2516596.

[4] D. Sculley, G. Holt, D. Golovin, E. Davydov, and T. Phillips, "Hidden technical debt in Machine learning systems," *NIPS'15: Proceedings of the 29th International Conference on Neural Information Processing Systems*, vol. 2, pp. 2503–2511, 2015, Accessed: Jul. 23, 2025. [Online]. Available: https://dl.acm.org/doi/10.5555/2969442.2969519?utm_source=chatgpt.com

[5] "The 2025 AI Index Report," *Stanford University*, 2025. https://hai.stanford.edu/ai-index/2025-ai-index-report (accessed Jul. 24, 2025).

[6] "2024 Morning Consult + Unqork Survey: Technical Debt Stifles Innovation at 80% of Enterprises Surveyed," *Unqork*, Sep. 09, 2024. https://unqork.com/resource-center/guides/2024-morning-consult-unqork-survey-tech-debt-stifles-innovation-at-80-of-enterprises-surveyed/ (accessed Jul. 26, 2025).

[7] "Data Quality Across the Digital Landscape," *Esri*, 2024. https://www.esri.com/about/newsroom/arcnews/data-quality-across-the-digital-landscape (accessed Jul. 27, 2025).

[8] E. Raff, M. Benaroch, S. Samtani, and A. L. Farris, "What Do Machine Learning Researchers Mean by 'Reproducible'?," *Arxiv*, Dec. 2024, doi: https://doi.org/10.48550/arxiv.2412.03854.

[9] Y. Fernbach, "Breaking silos: unifying DevOps and MLOps into a unified software supply chain," *TechRadar*, May 26, 2025. https://www.techradar.com/pro/breaking-silos-unifying-devops-and-mlops-into-a-unified-software-supply-chain (accessed Jul. 29, 2025).

[10] "Article 99: Penalties," *EU Artificial Intelligence Act*, 2025. https://artificialintelligenceact.eu/article/99/ (accessed Jul. 30, 2025).

[11] E. Woollacott, "These are the UK industries facing the biggest digital skills gaps in 2025," *IT Pro*, 2025. https://www.itpro.com/business/careers-and-training/these-are-the-uk-industries-facing-the-biggest-digital-skills-gaps-in-2025 (accessed Aug. 15, 2025).

[12] S. Magnet, "Machine Learning Engineer Job Outlook 2025: Top Skills & Trends – 365 Data Science," *365 Data Science*, Apr. 24, 2025. https://365datascience.com/career-advice/career-guides/machine-learning-engineer-job-outlook-2025/ (accessed Aug. 02, 2025).