The Hitchhikers Guide to Production-ready Trustworthy Foundation Model powered Software (FMware)

Kirill Vasilevski[1], Benjamin Rombaut[1], Gopi Krishnan Rajbahadur[1,2], Gustavo A. Oliva[1,2],
Keheliya Gallaba[1], Filipe R. Cogo[1], Jiahuei (Justina) Lin[1], Dayi Lin[1], Haoxiang Zhang[1], Bouyan
Chen[1], Kishanthan Thangarajah[1], Ahmed E. Hassan[2], Zhen Ming (Jack) Jiang[3]
[1]Centre for Software Excellence, Huawei Canada; [2]Queen's University, Canada; [3]York University, Canada

Abstract

Foundation Models (FMs) such as Large Language Models (LLMs) are reshaping the software industry by enabling FMware, systems that integrate these FMs as core components. In this KDD 2025 tutorial, we present a comprehensive exploration of FMware that combines a curated catalogue of challenges with real-world production concerns. We first discuss the state of research and practice in building FMware. We further examine the difficulties in selecting suitable models, aligning high-quality domain-specific data, engineering robust prompts, and orchestrating autonomous agents. We then address the complex journey from impressive demos to production-ready systems by outlining issues in system testing, optimization, deployment, and integration with legacy software. Drawing on our industrial experience and recent research in the area, we provide actionable insights and a technology roadmap for overcoming these challenges. Attendees will gain practical strategies to enable the creation of trustworthy FMware in the evolving technology landscape.

CCS Concepts

• **Computing methodologies → Natural language generation**; **Artificial intelligence**; • **Software and its engineering → Software development techniques**.

Keywords

Foundation models, Large Language Models, AI4SE

1 Introduction

Foundation Models (FMs), particularly Large Language Models (LLMs), are transforming software engineering by enabling **FMware**,

i.e., software systems that integrate FMs as core components. FMware expands software capabilities, allowing domain experts to develop AI-driven applications without extensive programming expertise. As adoption accelerates, the market for FMware is projected to grow at a compound annual rate of 35.9% through 2030 [38]. Despite this rapid growth, developing FMware remains a significant challenge, requiring engineering practices that differ fundamentally from traditional software development.

Building FMware is difficult because it introduces stochastic behavior, evolving dependencies, and high computational costs while lacking established engineering methodologies. Developers must handle unpredictable outputs, data sensitivity, prompt brittleness, and limited interpretability, making even reliable prototyping difficult. FMware often integrates multiple AI components, creating interdependencies that amplify failure points and complicate engineering. These complexities set FMware apart from conventional software, demanding new software engineering approaches.

Deploying FMware at scale presents an even greater challenge. Production-ready FMware must ensure performance stability, robustness, and compliance while managing scalability constraints, security risks, and deployment costs. Industry experiences illustrate these difficulties: LinkedIn required four additional months to refine the final 20% of an FMware deployment, facing diminishing returns on each incremental improvement [2]. Microsoft and GitHub reported that as FMware complexity increases, validation and testing become prohibitively expensive [72] due to the absence of standardized evaluation frameworks. The cost of deployment further complicates production readiness. For instance, OpenAI's ChatGPT infrastructure alone was estimated to cost $700,000 per day in 2023 [7].

Existing research has explored FMs' role in software engineering, including their applications in code generation, testing, documentation, and reasoning [33, 46, 49, 50, 101]. These studies primarily rely on academic benchmarks and structured datasets, often overlooking real-world deployment challenges. Meanwhile, research on software production readiness has largely focused on release criteria, system reliability, and ML-specific deployment challenges [14, 18, 27, 72, 100]. Studies on enterprise FM integration [24, 67] highlight developer concerns but remain constrained to specific ecosystems. A recent survey accompanying a KDD 2024 tutorial [29] examined how LLMs enhance software development workflows. While that work focused on improving development processes using LLMs, we examine a different challenge: **the engineering complexities of FMware itself**. Unlike prior surveys, which emphasize how LLMs assist software engineering, we analyze the core software engineering difficulties of building, deploying, and maintaining FMware at scale. Furthermore, this survey improves

on previous work [43, 76] by providing a cohesive overview of all challenges related to FMware development in a way that is relevant to researchers and practitioners alike. Our work systematically identifies these challenges and provides a roadmap for addressing them.

To bridge these gaps, we propose a KDD 2025 lecture-style tutorial that explores insights, concerns, and practical strategies for developing and deploying trustworthy, production-ready FMware. We first present an overview of the FMware lifecycle in Section 2. We then discuss key the challenges and present a critical overview of the state of practice in FMware engineering in Section 3. Finally, we outline actionable insights and emerging solutions, offering a roadmap for advancing FMware engineering in Section 4. Section 5 concludes our paper.

2 FMware Lifecycle

The FMware lifecycle defines how foundation models (FMs) are integrated into software systems, from selection and alignment to long-term operation. Figure 1 shows an overview of the FMware lifecycle. Unlike traditional software, which follows deterministic execution paths, FMware relies on probabilistic inference, dynamic adaptation, and real-time decision-making. This requires new engineering approaches that blend AI-driven capabilities with structured software development principles. Below we discuss about the different stages of FMware lifecycle.

2.1 FM Selection and Alignment

Selecting an appropriate FM is the first step in the FMware lifecycle. Unlike traditional software components with predefined logic, FMs generate responses based on statistical patterns from large-scale training data. Developers must consider task requirements, scalability, licensing constraints, and adaptability when choosing an FM.

Once selected, alignment is necessary to fine-tune an FM for application-specific needs. *Data alignment* involves curating domain-relevant datasets, ensuring the FM performs reliably on specialized tasks [9, 59]. *Model alignment* further refines behavior using techniques such as instruction tuning and reinforcement learning with human feedback (RLHF) to guide model outputs toward desired behaviors [84, 98].

2.2 Prompt Engineering and Grounding

Since FMware interacts with users through natural language prompts rather than explicit function calls, prompt engineering defines system behavior. Developers iteratively refine prompts to improve accuracy, consistency, and adaptability across different contexts [74]. Production-ready systems require structured prompt design, incorporating formatting, context augmentation, and constraint mechanisms.

To enhance reliability, developers employ well-established *prompt patterns*, such as chain-of-thought prompting [94], self-consistency prompting, and structured templating. These patterns improve coherence, reduce variability, and enable more predictable FM responses. Prompt portability is another critical consideration, ensuring that prompts remain effective when transitioning between

different FMs [43]. However, manually crafting and optimizing prompts remains a labor-intensive and iterative process, leading researchers and practitioners to explore automated and systematic solutions.

Grounding improves response reliability by linking FM outputs to external information sources. Techniques such as retrieval-augmented generation (RAG) ensure factual accuracy and reduce hallucination risks [56]. Grounding also aids compliance by ensuring alignment with authoritative sources, mitigating risks of generating misleading content [67].

2.3 Agents, Orchestration, and Cognitive Architecture Planning

While FMware can operate through direct prompt-response interactions, modern applications increasingly incorporate autonomous agents for decision-making, task execution, and interoperability with external systems. Agents extend FM capabilities by enabling multi-step reasoning, maintaining state, and integrating with traditional software components. An FMware agent functions as an adaptive entity that executes complex workflows, interacts with external tools and databases, and refines decision-making through feedback loops.

To manage these capabilities, cognitive architecture planning defines how FMware components coordinate within a system and enable agent orchestration. Depending on the application, orchestration can take different forms: **static workflows**, where execution paths are predefined; **tool-augmented execution**, where FMs interact with databases or predefined functions dynamically; and **agent-based orchestration**, where autonomous agents adjust execution flows and collaborate with other agents in real time.

FMware agents integrate with traditional software architectures, enhancing existing deterministic systems by bridging structured programming with adaptive reasoning.

2.4 Observability and Guarding

Observability in software engineering refers to the ability to monitor system behavior, detect failures, and analyze performance trends. Traditional observability relies on logs, traces, and metrics to ensure expected functionality. However, FMware introduces new challenges due to its probabilistic nature, requiring deeper insights into dynamic behaviors beyond conventional monitoring.

Semantic observability extends traditional practices by capturing user corrections, intent recognition, and model drift detection to refine FMware performance over time. Unlike classical observability, which tracks system health at a functional level, FMware observability must also analyze emergent behaviors arising from model interactions and decision-making processes.

Ensuring trust in FMware requires robust guarding mechanisms to enforce compliance, safety, and reliability [30, 78]. Unlike static rule-based validation, FMware must assess correctness dynamically at inference time, incorporating safeguards for data validation, response filtering, and risk mitigation.
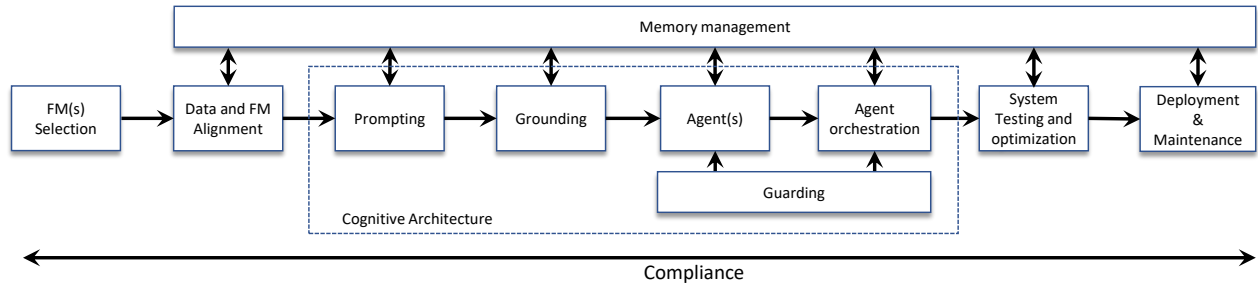
Figure 1: FMware Lifecycle.

2.5 Performance Engineering, Testing, and Optimization

FMware validation differs from traditional software testing due to its non-deterministic outputs. Unlike conventional applications, where the same input produces the same output, FMware responses vary based on model updates, prompt changes, and shifting input distributions [63]. Ensuring reliability requires continuous evaluation pipelines that benchmark consistency across evolving model versions while assessing distributional stability, response variability, and edge case handling [79]. Optimization efforts extend beyond accuracy, balancing latency, compute efficiency, and cost constraints. Unlike deterministic APIs, FM inference introduces variable processing times, requiring adaptive strategies such as query batching, caching, and model routing.

2.6 Deployment, Maintenance, and Tooling

Deploying FMware extends beyond traditional CI/CD practices, as model behavior evolves dynamically rather than following static updates [107]. Maintaining FMware requires continuous inference monitoring, prompt tuning based on observed performance, and version rollback mechanisms to mitigate performance regressions. Unlike conventional software, FMware lacks standardized tooling for versioning, fine-tuning management, and scalable inference deployment, necessitating adaptive update strategies to ensure stability.

2.7 Cross-cutting concerns

2.7.1 Memory Management. FMware requires efficient memory management to preserve context across interactions, optimize multi-agent workflows, and manage inference workloads [62, 79]. Unlike traditional applications with explicitly stored state, FMware must dynamically handle context persistence, agent coordination, and compute optimization to ensure efficiency.

2.7.2 Compliance & Regulations. Compliance in FMware spans the entire lifecycle, requiring continuous evaluation due to the probabilistic nature of model outputs and evolving regulations. Organizations must ensure that training data and model alignment comply with licensing, privacy laws, and ethical AI principles [43, 67]. Regulatory risks arise from data misuse, while model fine-tuning techniques such as RLHF must be audited to prevent biases.

Inference-time compliance demands real-time safeguards, including content filtering, bias detection, and automated moderation to mitigate harmful outputs [30, 78]. Post-deployment, FMware requires robust logging, auditing, and transparency measures such as model cards and AI Bill of Materials (AI BOM) [43]. Governance frameworks must define accountability, risk management, and alignment with emerging AI regulations [67], ensuring responsible FMware deployment while maintaining adaptability to regulatory shifts.

The FMware lifecycle defines how foundation models integrate into modern software systems, introducing new considerations for prompt engineering, agent orchestration, observability, and continuous adaptation. Unlike traditional software, FMware evolves dynamically through interaction feedback, semantic observability, and model-driven optimization, requiring engineering approaches that blend AI, software reliability, and real-time adaptability. As FMware adoption accelerates, structured software engineering principles will be critical to ensuring scalability, trustworthiness, and enterprise-level integration.

3 Development & Productionization Challenges in FMware

In this section, we critically analyze the state of practice in FMware development and productionization. We examine existing approaches, their limitations, and the persistent challenges that hinder the transition from experimental FMware to robust, trustworthy production-ready systems.

3.1 FM Selection and Alignment

Current approaches to FM selection and alignment remain largely ad hoc. Selecting and aligning FMs for FMware requires balancing functionality, cost, latency, and infrastructure constraints. As of February 2025, over 1.4 million open-source models are available on Hugging Face [32], alongside proprietary models from providers

like OpenAI [69] and Anthropic [11]. Proof-of-concept FMware often relies on large general-purpose FMs (e.g., GPT-4o [68], o1 [70], Claude 3.7 Sonnet [12]) due to their broad capabilities, but transitioning to production necessitates optimizing for task-specific accuracy while mitigating cost and deployment overheads [51, 102]. Developers rely on manual experimentation to compare models, with few automated benchmarking tools for real-world deployment scenarios. Additionally, complex FMware systems may require multiple specialized models rather than a single FM, compounding the selection challenge.

Beyond model selection, data and model alignment are critical to ensuring that an FM meets specific enterprise and use-case requirements [98]. However, ensuring data quality, bias mitigation, and efficient data estimation, required for efficient data alignment remains a persistent challenge. Low-quality datasets degrade reliability [9], while poor domain coverage can introduce unintended biases. More recently, prioritizing data quality over volume has been shown to improve alignment efficiency [40, 103]. Furthermore, estimating the required data volume is difficult—underestimation leads to degraded performance, whereas overestimation results in unnecessary computational costs [98].

Finally, debugging alignment remains challenging, as FM opacity limits fine-grained dataset impact analysis, even with techniques like influence functions and feature importance analysis [39, 82]. Automating model evaluation and selection for production FMware remains an open problem.

3.2 Prompt Engineering and Grounding

3.2.1 Prompt Engineering.
Designing prompts that reliably elicit the desired behavior from foundation models (FMs) remains a significant challenge in FMware development. While basic applications rely on ad-hoc prompt crafting, production-grade FMware demands a structured approach to ensure robustness, maintainability, and cross-model portability [74]. Despite efforts to automate prompt generation through optimization techniques [53, 105], most real-world systems still rely on trial-and-error refinement, making the process time-consuming and error-prone [86].

Another key challenge is *prompt sensitivity*, where minor changes in wording, model updates, or inference parameters can lead to drastic output variations [35]. This instability complicates software maintenance and necessitates continuous prompt adjustments. Additionally, prompts optimized for one FM often fail to generalize across different models due to architectural differences and variations in tokenization [23, 80]. Such lack of portability hinders FMware evolution and increases development costs.

Another major issue is the *lack of built-in validation mechanisms*, making it difficult to assess prompt reliability. Unlike traditional software, where unit tests verify deterministic behavior, FMware lacks standardized QA methodologies to systematically test prompt performance [8, 52]. Debugging also remains challenging, as failures can stem from poorly designed prompts, FM limitations, or inference-time parameters [10, 13]. Research into *automated prompt optimization* [34] and *reinforcement learning-based prompt tuning* [28] has shown promise, but such techniques remain limited by their inability to adapt across different FMs and deployment environments.

3.2.2 Grounding.
Integrating grounding into FMware presents multiple challenges, including data selection, retrieval efficiency, and content integration. A primary challenge is ensuring *high information density* in grounding data. Poorly structured or overly broad information can degrade FM performance, leading to irrelevant or incoherent responses [55]. While structured retrieval can improve precision, it requires carefully curated datasets and filtering mechanisms to maintain relevance [26, 58]. Additionally, balancing retrieval breadth and response conciseness is difficult, as excessive grounding information can lead to response dilution, while insufficient data may fail to correct FM errors [89].

Another challenge is optimizing *retrieval efficiency*. Many FMware systems rely on computationally expensive neural retrieval methods, such as dense vector search, to improve grounding accuracy. However, these techniques often introduce latency overhead, which can hinder real-time applications [81]. In some cases, lightweight keyword-based retrieval achieves comparable performance with significantly lower computational costs, yet determining when to use neural or lexical retrieval remains an open problem [100].

Finally, the seamless *integration of retrieved information* into FM responses is non-trivial. While naive concatenation of retrieved documents can overwhelm the model with extraneous data, overly restrictive filtering risks omitting crucial context. Effective fusion strategies, such as selective in-context learning or hybrid retrieval-conditioning approaches, are needed to improve response accuracy without introducing excessive complexity [89].

3.3 Agents, Orchestration, and Cognitive Architecture Planning

Agents are integral to FMware, enabling decision-making, automation, and interaction with external tools and software. However, their effective deployment in production FMware remains an open challenge, requiring careful consideration of modularity, orchestration, and tool integration.

Agent modularity is crucial for reliability and maintainability. While large monolithic agents provide broad functionality, they introduce compounding errors, increase debugging complexity, and reduce adaptability [20]. Conversely, breaking down tasks into smaller, specialized agents enhances modularity and scalability [77], but also raises coordination challenges. As agent ecosystems grow, managing interdependencies, communication, and context sharing becomes increasingly difficult, requiring robust state management mechanisms.

Agent orchestration is another key challenge. While FM-based planners can automatically decompose tasks and assign workloads, they often suffer from hallucinations and inconsistent task delegation [41, 47]. The inherent unpredictability of FM-driven orchestration makes it difficult to ensure reliable task execution, particularly in multi-agent FMware. Hybrid approaches that incorporate structured task delegation mechanisms, rather than relying solely on FM-based planning, are needed to improve consistency. Existing frameworks attempt to address this by introducing structured oversight. For instance, AutoGen [96] and Copilot Workspace [37] enable human intervention in agent workflows, providing corrective oversight to mitigate FM-based errors. Similarly, MetaGPT [45]

encodes standard operating procedures (SOPs) into prompts, guiding agents towards structured decision-making, while symbolic planning approaches [104] enhance controllability.

Tool integration presents additional challenges. Agents rely on external tools and APIs, but their ability to correctly select and use tools depends heavily on documentation structure and abstraction levels. High-level abstractions mitigate complexity, as exposing low-level tool functions increases error rates and impairs scalability. Poorly documented or capability-centric tool descriptions further hinder FMware reliability by making it harder for agents to generalize tool usage [90]. Some frameworks attempt to abstract tool interactions for FMware systems. For instance, Semantic Kernel [64] encapsulates legacy software functionalities using *native functions*, making them accessible to FM-driven agents while maintaining modularity.

Despite these efforts, current orchestration frameworks remain limited in handling real-time reactivity, dynamic coordination, and ensuring robustness across complex agent-driven workflows. Production-ready FMware must address these gaps by refining agent orchestration strategies, improving state-sharing mechanisms, and incorporating structured oversight to mitigate FM unpredictability.

3.4 Performance Engineering, Testing, & Optimization

3.4.1 Performance Optimization. Production FMware systems demand rigorous optimization due to the compute-intensive nature of FMs and the need to ensure Service Level Agreements (SLAs) [16]. A primary challenge is high inference latency, which is exacerbated by long prompts, multi-round interactions, and large model sizes [48]. Using external FM providers introduces additional constraints such as token limits, request throttling, and unpredictable performance variations, all of which impact enterprise FMware deployment.

One approach to improving performance is selecting smaller, more efficient models. For example, Meta prioritized the 1.3B InCoder model over a 6.7B variant to reduce inference latency while maintaining acceptable accuracy [66]. Additionally, techniques such as quantization, distillation, and pruning help optimize FM inference speed, though balancing efficiency without performance degradation remains an open problem [106]. Semantic caching [36] is another promising optimization strategy, reducing redundant computation by storing and retrieving previous responses for similar queries.

Enterprise FMware often integrates multiple FMs and agents, requiring multiple rounds of inference. However, due to the non-deterministic nature of FM-based orchestration and planning, the exact number of inference rounds is often unknown before execution. This unpredictability can cause FMware to miss SLAs, even if individual FMs meet model-level constraints. This issue is magnified in agent-based FMware, where agents continuously generate new tasks and queries, making runtime costs difficult to estimate.

To address infrastructure constraints, organizations may opt to self-host FMs rather than relying on external providers. While this mitigates vendor limitations, it introduces challenges in managing hardware resources efficiently. In FMware requiring multiple

models, GPU memory contention forces frequent model swaps, adding tens of seconds to inference time per operation. These swap-induced latencies necessitate optimized scheduling strategies to prevent bottlenecks in production deployments.

3.4.2 Testing. Testing FMware is fundamentally different from traditional software testing due to its non-deterministic outputs and evolving model behaviors [72, 73]. Even with unchanged code, FMware responses can vary due to model updates, inference temperature changes, and external FM provider modifications. These challenges are particularly severe for systems relying on closed-source FM APIs, where developers have limited control over model internals.

Current FMware testing strategies remain immature and heavily rely on human evaluation or automated reference-based validation using large FMs such as GPT-4 [60]. However, FM-based evaluators exhibit only moderate correlation with human assessments (0.51) [60], often favoring superficial linguistic features over factual correctness [21, 87]. This inconsistency raises concerns about the reliability of AI-as-judge frameworks in evaluating FMware outputs.

Additionally, traditional software testing approaches do not fully address FMware's dynamic nature and resource constraints. Existing caching mechanisms, such as those used in LangChain, fail to account for real-time model updates and changing prompts, leading to ineffective cache invalidation. Furthermore, FMware lacks dependency-aware test execution strategies [83], resulting in redundant and computationally expensive testing pipelines. Retry mechanisms also remain underdeveloped; while conventional systems rely on re-execution for fault recovery, FMware often requires adaptive re-prompting or alternative query structures to ensure robustness.

Due to the complexity of testing multi-FM and multi-agent FMware, system-wide test suites must account for dynamic interactions, robustness under workload variations, and handling of unpredictable responses. However, current automated testing frameworks remain insufficient for large-scale FMware, making fault detection slow, increasing regression risks, and limiting CI/CD capabilities [1, 43]. Addressing these challenges requires new FMware-native testing methodologies, including structured prompt evaluation, dependency-aware regression testing, and improved reliability metrics for evaluating FMware correctness across inference variations.

3.5 Observability, Guarding, Deployment, Maintenance & Tools

Limited FMware-Native Observability: Observability in FMware remains a major challenge, as existing tools lack support for tracing reasoning paths, emergent behaviors, and inter-agent coordination [43]. While traditional observability platforms such as LangSmith [4], TraceLoop [5], and Weights & Biases [6] provide operational metrics like latency, token usage, and grounding accuracy, they fail to capture cognitive processes and implicit decision-making in FM-driven applications.

This issue is particularly problematic in multi-agent FMware, where decision-making is distributed across multiple autonomous

components. Current tools do not provide insight into how agents collaborate, share state, or dynamically adjust workflows. This lack of transparency complicates debugging, increases the risk of reasoning failures, and hampers the ability to ensure system-wide reliability. Furthermore, regulatory compliance mandates detailed logging of FM interactions, yet FMware lacks standardized mechanisms for maintaining audit trails across evolving deployments.

Lack of Efficient Feedback Mechanisms: Continuous feedback integration is essential for improving FMware performance, yet current implementations remain ad hoc and manual. Unlike structured feedback loops in traditional ML systems, FMware relies primarily on explicit user ratings [91] rather than passive, scalable feedback mechanisms. While GitHub Copilot employs semantic signal telemetry (e.g., tracking post-editing behavior) [3], no generalizable solution exists for capturing implicit corrections and real-world performance issues across diverse FMware applications.

Another challenge is distinguishing global insights (outer knowledge) from user-specific optimizations (inner knowledge). Without this differentiation, valuable model refinements may be overlooked, or user-specific preferences may be misapplied across all users, leading to degraded system-wide reliability. Production FMware must incorporate *automated feedback pipelines* that facilitate real-time bias detection, error correction, and adaptive learning.

Ineffective FM Update Mechanisms: Unlike traditional software, where updates introduce incremental and predictable improvements, FMware updates are often non-deterministic and unpredictable [43]. Adding more training data does not guarantee resolution of specific issues, and enhancements listed in release notes often lack clear mappings to observable behavior changes. As a result, developers struggle to test, validate, and adapt to new model versions, leading to performance regressions and degraded user trust.

Additionally, FM updates introduce hidden risks causing previously stable features to break, and model alignment may shift in unintended ways. This unpredictability complicates maintenance in production environments, requiring automated monitoring and rollback mechanisms to detect and mitigate regressions effectively.

3.5.1 **Guarding**. Despite widespread adoption of guardrails, current implementations lack adaptability and struggle with complex linguistic manipulation. Many rely on static keyword filtering, which can be easily bypassed through paraphrasing or subtle rewording [30]. While *semantic filtering* provides more contextual awareness, it remains prone to false positives and false negatives, failing to distinguish between benign variations and genuine violations.

FMware also lacks confidence estimation mechanisms, making it difficult to flag uncertain outputs for human review [15]. Without *real-time reliability assessments*, FMware deployments must default to human oversight, which severely limits scalability in large-scale applications. Addressing these limitations requires adaptive guardrails that dynamically refine themselves based on evolving threats and real-world usage patterns.

3.6 Cross-cutting Concerns

Cross-cutting concerns permeate across all stages of the FMware lifecycle, with solutions to them being essential for development of reliable production-ready FMware.

3.6.1 **Memory Management**. Effective memory management in FMware is essential for maintaining context consistency, optimizing workloads, and ensuring scalable inference [43, 54]. Unlike traditional applications with explicit state management, FMware dynamically stores and retrieves knowledge to preserve context across interactions, manage agent state, and handle retries efficiently [62, 79].

Knowledge representation inefficiencies remain a core challenge. Poorly structured memory leads to redundant or irrelevant data storage, inflating computational costs and degrading response quality [42, 97]. Optimizing knowledge retrieval is essential to *p*revent inconsistencies in multi-step reasoning and minimize unnecessary computation overhead.

Knowledge synchronization across multiple FMs and user contexts is another persistent issue. Feedback loops can introduce conflicting knowledge, leading to misalignment between FM states, context loss, and incorrect outputs [22, 99]. Managing retry storms, state sharing across agents, and ensuring consistent knowledge updates is particularly challenging in FMware where decisions are probabilistic and evolve dynamically [75, 107].

Additionally, robust memory architectures are vital for regulatory compliance and optimizing long-term FM behavior [43, 93]. Without structured memory management, FMware remains prone to context fragmentation, inefficient adaptation, and unpredictable system performance in production deployments [71].

3.6.2 **Compliance & Regulations**. Regulatory compliance in FMware is complex due to data privacy concerns, opaque licensing, and lack of transparency mechanisms. Unlike traditional software, FMware's probabilistic nature requires ongoing compliance assessment, as model behaviors evolve dynamically rather than remaining static.

Data privacy regulations, such as GDPR, require enterprises to prevent sensitive data leakage, posing challenges when FMware relies on third-party models hosted externally [72]. Self-hosting FMs mitigates this risk but is often infeasible due to infrastructure and resource constraints.

License compliance remains a critical challenge, particularly in closed-source FMs where data provenance is undisclosed, raising legal and ethical concerns regarding the use of copyrighted or restricted datasets [19, 61, 65, 85]. Existing AI licensing frameworks, such as Responsible AI Licenses (RAIL) [25] and SPDX 3.0 Dataset Profiles [88], attempt to address these gaps but remain limited to individual models or datasets, failing to account for FMware-wide compliance across multiple integrated components.

Lack of transparency and oversight is another regulatory barrier, particularly under frameworks like the EU AI Act, which mandates clear explainability and human oversight for high-risk AI systems [31]. However, FMware's black-box nature makes achieving these requirements intractable with current methodologies. While efforts like enhanced model and data cards [17] and AI Bill

of Materials (AI BOMs) [95] aim to improve traceability, they lack standardization and do not fully capture the evolving nature of FMware components.

Despite progress in compliance tooling, current approaches remain manual, fragmented, and poorly scalable for enterprise FMware. Without automated compliance monitoring and adaptive governance frameworks, regulatory risks will continue to challenge FMware's adoption in high-stakes domains.

4 Action Plan

Driven by the surveyed challenges listed in Section 3 as well as our practical experience in shipping real-world FMware, we henceforth present an *action plan.* This action plan is organized into a broad set of directions, each supplemented by a concise list of directives. While some directives can be somewhat readily applied to mitigate challenges, others are research topics in themselves. Therefore, we see this action plan as a valuable instrument to bridge research and practice (a key goal of KDD).

4.1 FM Selection and Alignment

Systematic choice of models. Investigate new or improved benchmarking suites that go beyond single-metric leaderboards (e.g., include cost-efficiency parameters). Create analytical and simulation-based frameworks to explore how different models impact cost, latency, and accuracy.

Data IDEs Covering the Entire Alignment Lifecycle. Integrate model- and human-in-the-loop approaches for labeling, debugging, testing, and versioning data. Incorporate programmatic labeling (data programming) to reduce manual effort. Enable collaborative review and aggregation of labeled data from multiple labelers.

Bias Detection, Compliance, and Testing Frameworks. Develop methods to automatically identify and mitigate inconsistencies or biases in alignment data. Provide mechanisms for ensuring regulatory and ethical compliance across different stages of data handling. Facilitate testing and validation of aligned data to improve quality and trustworthiness.

Asset Management for Open and Inner Source Data. Offer robust management and tracking capabilities to address licensing restrictions and maintain compliance. Ensure data accuracy, consistency, completeness, and representativeness for diverse tasks. Mitigate risks associated with shared or publicly available data through secure handling and version control.

User-Centric Data Programming and Labeling. Simplify labeling workflows and define user-friendly labeling functions to increase adoption and reduce errors. Support collaboration between technical and non-technical stakeholders (e.g., domain experts) for more effective data curation.

4.2 Prompt Engineering and Grounding

Multi-Prompt Architectures and Human-in-the-Loop Workflows. Develop prompt chains with branching (e.g., chain-of-thought, tree-of-thought) to handle complex tasks. Incorporate human feedback at intermediate steps to refine the final output. Establish design, debugging, visualization, and evolution techniques for increasingly complex cognitive processes.

Integrated Prompt IDEs for Collaboration and Quality Assurance Support real-time collaboration, including version control and role-based access for prompt design. Provide just-in-time quality assurance with prompt debuggers, linters, and visualization of prompt performance. Enable dynamic selection and adjustment of lower-level prompts based on model responses, context, and user interactions.

Asset Management and Knowledge Integration. Provide collaborative environments for sharing, reviewing, and refining prompts at global or project-specific levels. Incorporate debugging features that reveal token-level or component-level contributions to generated outputs. Integrate grounding data management (e.g., uploading, tokenizing, vectorizing, and retrieving data) to ensure high-quality and contextually accurate responses.

4.3 Agents, Orchestration, and Cognitive Architecture Planning

Fine-Grained Controllability and Component-Based Design Adopt established software engineering principles (e.g., *single responsibility* and *modularity*) to decompose FMware into small, self-contained components with fine-grained control boundaries. Introduce permission models and constraints to regulate component interactions, preserving user privacy and security. Provide holistic architectural views that help identify design deviations and maintain cohesion, coupling, and separation of concerns in FMware.

Built-In Quality Through Structured Knowledge and Curriculum Move beyond simple vector databases toward knowledge graphs for richer, more structured contextual information. Implement *curriculum engineering* [44, 92] and versioning to ensure agents build compositional skills and reuse existing knowledge effectively. Automate curriculum co-creation and continuous QA to remove redundant or outdated information, thereby improving accuracy, reliability, and maintainability of FMware. Employ semantic and structural checks for prompt-like inputs to reduce errors, hallucinations, and unpredictable outputs.

Controlled Execution and Systematic Validation Design frameworks that enforce *repeatability* by restoring execution snapshots and monosemantic units to reproduce exact conditions for debugging and regression testing. Allow *guided exploration* of alternative execution paths, varying inputs and agent decisions in a controlled manner to uncover hidden bugs and assess system robustness. Balance consistent execution flows with exploration to optimize both reliability and resilience of FMware applications.

4.4 Observability and Guarding

Multi-Level Observability Framework for FMware. Capture both functional performance metrics and internal cognitive processes across various abstraction layers. Provide traceability at the system level and throughout autonomous agent decision-making stages.

Analytics Engine and Visualization Tools. Implement a "flight recorder" mechanism to selectively log agent reasoning and communications. Integrate analytics engines for real-time visualization of multi-agent workflows, quickly pinpointing root causes in complex scenarios. Mitigate the observer effect (e.g., altering behavior

by prompting "think step-by-step") through surrogate agents that enable debugging without interfering directly.

Advanced Telemetry and Metrics. Develop enhanced, context-aware metrics to help developers assess capabilities, analyze behaviors, and validate performance in dynamic FMware environments. Employ semantic signal telemetry solutions that can generalize beyond specific implementations or domains. Offer monitoring, logging, and alerting systems tailored to the unique operational requirements of FMware.

4.5 Compliance

Automated FMwareBOM Generation and Formal Verification. Extend SPDX 3.0 (AI and dataset profiles) to create a comprehensive FMwareBOM for components, licenses, synthetic data, RLHF data, and user feedback. Employ formal verification (e.g., SMT solvers) to ensure provable compliance with legal and regulatory requirements. Integrate FMwareBOM tools into the development lifecycle to facilitate real-time, machine-readable compliance checks.

FMware-Specific Licensing and Compliance Tooling Develop tools similar to Fossology, tailored to detect and manage licensing issues unique to FMware (e.g., code, dataset, FM, and agent licenses). Establish documentation standards and disclosure procedures for compliance verification, including known limitations. Enable automated reasoning on generated FMwareBOMs for preliminary license and regulatory compliance assessments.

4.6 Performance Engineering, Testing, and Optimization

Declarative Representation for Intent-Aware Optimization. Represent FMware at a higher level of abstraction, preserving developer intent (e.g., data locality and task placement considerations). Enable application graph-level optimization by capturing the execution graph in a declarative format. Facilitate joint optimizations between front-end (intent-aware representation), back-end (graph performance), and FM inference accelerations.

Resource-Aware QA Framework and Smart Caching. Integrate a robust FMware-native caching mechanism to store and reuse FM query results, reducing repeated FM calls and maintaining cache integrity. Dynamically prioritize and optimize test execution, grouping similar tests to lower resource consumption and accelerate feedback. Leverage prompt compression techniques to further reduce latency and resource demands during testing.

Dynamic Prompt Re-Engineering and Retry Mechanisms. Implement advanced retry strategies that adapt prompts and validate responses, reducing the risk of repeated failures. Adjust queries intelligently when outputs are unsatisfactory to improve the likelihood of success in subsequent attempts. Balance thorough testing with resource efficiency to keep FMware systems robust and performant.

Automated Test Generation Through Metamorphic Relations (MRs). Leverage real-world user feedback (e.g., thumbs-up/down) to generate domain-specific metamorphic relations. Conduct metamorphic testing (MT) on FMware to evaluate diverse system properties and behaviors. Integrate an iterative feedback loop where experts refine the MRs, enhancing test coverage and reliability.

Next-Generation AI-Judge Framework. Create specialized AI-judges capable of evaluating deeper attributes such as factual accuracy, consistency, and compliance with domain logic [57]. Enable developers to craft prompts for these judges that capture specific business constraints, moving beyond superficial checks. Employ curriculum engineering techniques to train judges in a structured, guided, and flexible manner for cost-effectiveness and adaptability.

Reduced Human Effort with Enhanced Reliability. Streamline test result analysis and summarization through automation, reducing dependence on heavy manual evaluation. Combine automated testing with AI-judges to address the non-deterministic nature of FM outputs. Evolve testing mechanisms with each iteration, continuously enhancing the robustness of FMware quality assurance.

4.7 Deployment, Maintenance, and Tooling

Efficient Deployment. Consider FM characteristics, hardware heterogeneity, and data movement costs for global optimizations in deployment, rather than treating FM inference as a standalone concern. Simplify computing resource management through *uni-clusters*, unifying training, fine-tuning, serving, and agent customization in a single infrastructure.

Automated Feedback Integration. Passively collect implicit user signals (e.g., hesitations, corrections) to capture behavioral cues, not just explicit ratings. Employ real-time analysis of user interactions to continuously refine and improve FMware responses. Reduce reliance on manual, explicit feedback by seamlessly integrating AI-driven feedback solicitation into regular usage patterns.

Knowledge Partitioning and Federated Learning. Distinguish between "outer knowledge" (general improvements) and "inner knowledge" (user-specific optimizations) to prevent overgeneralization. Automate the categorization of feedback into these knowledge domains for more targeted adaptation. Incorporate federated learning techniques to share "inner knowledge" across multiple FMware instances, enhancing overall performance without compromising privacy.

Versioning and Collaboration Support for FMware Assets Define standards and protocols to version-control fine-grained FMware entities (e.g., individual prompt components like persona, instructions, examples). Facilitate inter-organizational reuse through community-driven platforms and prompt hubs, promoting asset sharing and interoperability. Leverage role-based access control to maintain security and granular management of asset visibility and contributions.

5 Conclusion

As FMware adoption grows, bridging the gap between research and production remains a challenge due to scalability, reliability, and a plethora of other issues that we detail in this paper. Our tutorial along with this survey paper provides a structured perspective on FMware's lifecycle, integrating insights from AI, data science, and software engineering to address practical deployment challenges. By highlighting key gaps in alignment, orchestration, observability, and optimization, it equips the KDD community with strategies for building scalable, trustworthy, and production-ready FMware.

References

[1] 2023. Testing LLM-Based Applications: Strategy and Challenges — blog.scottlogic.com. https://blog.scottlogic.com/2023/11/14/testing-LLM-based-applications-strategy-and-challenges.html. [Accessed 07-10-2024].

[2] 2024. Musings on Building a Generative AI Product — linkedin.com. https://www.linkedin.com/blog/engineering/generative-ai/musings-on-building-a-generative-ai-product. [Accessed 08-10-2024].

[3] 2025. Copilot Internals. https://thakkarparth007.github.io/copilot-explorer/posts/copilot-internals last accessed: 2025-03-11.

[4] 2025. LangSmith by LangChain. https://www.langchain.com/langsmith/ last accessed: 2025-03-11.

[5] 2025. Traceloop. https://www.traceloop.com/ last accessed: 2025-03-11.

[6] 2025. Weights and Biases - Weave. https://wandb.ai/site/weave/ last accessed: 2025-03-11.

[7] Dr. Assad Abbas. 2024. The Financial Challenges of Leading in AI: A Look at OpenAI's Operating Costs — unite.ai. https://www.unite.ai/the-financial-challenges-of-leading-in-ai-a-look-at-openais-operating-costs/. [Accessed 11-10-2024].

[8] Toufique Ahmed and Premkumar Devanbu. 2023. Better patching using LLM prompting, via Self-Consistency. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1742–1746.

[9] Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, et al. 2024. A survey on data selection for language models. *arXiv preprint arXiv:2402.16827* (2024).

[10] Oluyemi Enoch Amujo and Shanchieh Jay Yang. 2024. How Good Is It? Evaluating the Efficacy of Common versus Domain-Specific Prompts on Foundational Large Language Models. *arXiv preprint arXiv:2407.11006* (2024).

[11] Anthropic. n.d.. All models overview - Anthropic. https://docs.anthropic.com/en/docs/about-claude/models/all-models. Accessed: 2025-02-01.

[12] Anthropic. n.d.. Claude 3.7 Sonnet and Claude Code. https://www.anthropic.com/news/claude-3-7-sonnet. Accessed: 2025-02-01.

[13] Narges Ashtari, Ryan Mullins, Crystal Qian, James Wexler, Ian Tenney, and Mahima Pushkarna. 2023. From Discovery to Adoption: Understanding the ML Practitioners' Interpretability Journey. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*. 2304–2325.

[14] Abhaya Asthana and Jack Olivieri. 2009. Quantifying software reliability and readiness. In *2009 IEEE international workshop technical committee on communications quality and reliability*. IEEE, 1–6.

[15] Suriya Ganesh Ayyamperumal and Limin Ge. 2024. Current state of LLM Risks and AI Guardrails. *arXiv preprint arXiv:2406.12934* (2024).

[16] R. Bommasani, D.A Hudson, E. Adeli, Altman, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[17] D. Brajovic, N. Renner, V.P. Goebels, P. Wagner, B. Fresz, et al. 2023. Model Reporting for Certifiable AI: A Proposal from Merging EU Regulation into AI Development. *arXiv preprint arXiv:2307.11525* (2023).

[18] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley. 2017. The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE international conference on big data (big data)*. IEEE, 1123–1132.

[19] K.K Chang, M. Cramer, S. Soni, and D. Bamman. 2023. Speak, memory: An archaeology of books known to chatgpt/gpt-4. *arXiv preprint arXiv:2305.00118* (2023).

[20] Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, et al. 2024. CodeR: Issue Resolving with Multi-Agent and Task Graphs. *arXiv preprint arXiv:2406.01304* (2024).

[21] Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. 2024. Humans or llms as the judge? a study on judgement biases. *arXiv preprint arXiv:2402.10669* (2024).

[22] Hung-Ting Chen, Michael JQ Zhang, and Eunsol Choi. 2022. Rich knowledge sources bring complex knowledge conflicts: Recalibrating models to reflect conflicting evidence. *arXiv preprint arXiv:2210.13701* (2022).

[23] L. Chen, M. Zaharia, and J. Zou. 2023. How is ChatGPT's behavior changing over time? arXiv:2307.09009 [cs.CL]

[24] Xiang Chen, Chaoyang Gao, Chunyang Chen, Guangbei Zhang, and Yong Liu. 2025. An Empirical Study on Challenges for LLM Application Developers. *ACM Transactions on Software Engineering and Methodology* (2025).

[25] D. Contractor, D. McDuff, J.K. Haines, J. Lee, C. Hines, et al. 2022. Behavioral use licensing for responsible AI. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 778–788.

[26] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 719–729.

[27] James Cusick. 2013. Architecture and Production Readiness Reviews in Practice. *arXiv preprint arXiv:1305.2402* (2013).

[28] M. Deng, J. Wang, C.-P. Hsieh, Y. Wang, H. Guo, et al. 2022. RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning. In *Proc. of Conf. on EMNLP*. ACL, 3369–3391.

[29] Hao Ding, Ziwei Fan, Ingo Guehring, Gaurav Gupta, Wooseok Ha, Jun Huan, Linbo Liu, Behrooz Omidvar-Tehrani, Shiqi Wang, and Hao Zhou. 2024. Reasoning and Planning with Large Language Models in Code Development. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 6480–6490. doi:10.1145/3637528.3671452

[30] Ximing Dong, Dayi Lin, Shaowei Wang, and Ahmed E Hassan. 2024. A Framework for Real-time Safeguarding the Text Generation of Large Language. *arXiv preprint arXiv:2404.19048* (2024).

[31] L. Edwards. 2021. The EU AI Act: a summary of its significance and scope. *AI (the EU AI Act)* 1 (2021).

[32] Hugging Face. n.d.. Models - Hugging Face. https://huggingface.co/models. Accessed: 2025-02-01.

[33] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.

[34] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution. arXiv:2309.16797 [cs.CL] https://arxiv.org/abs/2309.16797

[35] T. Gao, A. Fisch, and D. Chen. 2021. Making Pre-trained Language Models Better Few-shot Learners. In *Proc. of Annual Meeting of the ACL-IJCNLP*. (ACL), 3816–3830.

[36] I. Gim, G. Chen, S.S. Lee, N. Sarda, A. Khandelwal, et al. 2023. Prompt Cache: Modular Attention Reuse for Low-Latency Inference. *arXiv preprint arXiv:2311.04934* (2023).

[37] Github. [n. d.]. GitHub Next | Copilot Workspace. https://githubnext.com/projects/copilot-workspace/. Accessed 02-06-2024.

[38] Grand View Research. 2023. Large Language Model Market Size, Share & Trends Analysis Report By Application (Customer Service, Content Generation), By Deployment, By Industry Vertical, By Region, And Segment Forecasts, 2024 - 2030. https://www.grandviewresearch.com/industry-analysis/large-language-model-llm-market-report.

[39] R. Grosse, J. Bae, C. Anil, N. Elhage, A. Tamkin, et al. 2023. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296* (2023).

[40] S. Gunasekar, Y. Zhang, J. Aneja, C.C.T. Mendes, A. Del Giorno, et al. 2023. Textbooks Are All You Need. *arXiv preprint arXiv:2306.11644* (2023).

[41] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680* (2024).

[42] Tiezheng Guo, Qingwen Yang, Chen Wang, Yanyi Liu, Pan Li, Jiawei Tang, Dapeng Li, and Yingyou Wen. 2024. Knowledgenavigator: Leveraging large language models for enhanced reasoning over knowledge graph. *Complex & Intelligent Systems* 10, 5 (2024), 7063–7076.

[43] Ahmed E. Hassan, Dayi Lin, Gopi Krishnan Rajbahadur, Keheliya Gallaba, Filipe Roseiro Cogo, Boyuan Chen, Haoxiang Zhang, Kishanthan Thangarajah, Gustavo Oliva, Jiahuei (Justina) Lin, Wali Mohammad Abdullah, and Zhen Ming (Jack) Jiang. 2024. Rethinking Software Engineering in the Era of Foundation Models: A Curated Catalogue of Challenges in the Development of Trustworthy FMware. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (Porto de Galinhas, Brazil). 294–305.

[44] Ahmed E. Hassan, Gustavo A. Oliva, Dayi Lin, Boyuan Chen, Zhen Ming, and Jiang. 2024. Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap. arXiv:2410.06107

[45] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, et al. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI]

[46] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.

[47] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716* (2024).

[48] A.Q Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D.S. Chaplot, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[49] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A Survey on Large Language Models for Code Generation. *arXiv preprint arXiv:2406.00515* (2024).

[50] Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2024. From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479* (2024).

[51] Uday Kamath, Kevin Keenan, Garrett Somers, and Sarah Sorenson. 2024. LLMs in Production. In *Large Language Models: A Deep Dive: Bridging Theory and Practice*. Springer, 315–373.

[52] Ryo Kamoi, Sarkar Snigdha Sarathi Das, Renze Lou, Jihyun Janice Ahn, Yilun Zhao, Xiaoxin Lu, Nan Zhang, Yusen Zhang, Ranran Haoran Zhang, Sujeeth Reddy Vummanthala, et al. 2024. Evaluating LLMs at Detecting Errors in LLM Responses. *arXiv preprint arXiv:2404.03602* (2024).

[53] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *The Twelfth International Conference on Learning Representations*.

[54] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 611–626.

[55] Philippe Laban, Alexander R Fabbri, Caiming Xiong, and Chien-Sheng Wu. 2024. Summary of a haystack: A challenge to long-context llms and rag systems. *arXiv preprint arXiv:2407.01370* (2024).

[56] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9459–9474.

[57] Jiahuei Lin, Dayi Lin, Sky Zhang, and Ahmed E. Hassan. 2024. Engineering AI Judge Systems. arXiv:2411.17793 [cs.SE] https://arxiv.org/abs/2411.17793

[58] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.

[59] Ruibo Liu, Jerry Wei, Fangyu Liu, Chenglei Si, Yanzhe Zhang, Jinmeng Rao, Steven Zheng, Daiyi Peng, Diyi Yang, Denny Zhou, et al. 2024. Best practices and lessons learned on synthetic data for language models. *arXiv preprint arXiv:2404.07503* (2024).

[60] Y. Liu, D. Iter, Yi. Xu, S. Wang, R. Xu, and C. Zhu. 2023. Gpteval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634* (2023).

[61] S. Longpre, R. Mahari, A. Chen, N. Obeng-Marnu, D. Sileo, et al. 2023. The Data Provenance Initiative: A Large Scale Audit of Dataset Licensing & Attribution in AI. *arXiv preprint arXiv:2310.16787* (2023).

[62] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Qingwei Lin, Jianguang Lou, Shifeng Chen, Yansong Tang, and Weizhu Chen. 2024. Arena Learning: Build Data Flywheel for LLMs Post-training via Simulated Chatbot Arena. *arXiv preprint arXiv:2407.10627* (2024).

[63] Wanqin Ma, Chenyang Yang, and Christian Kästner. 2024. (Why) Is My Prompt Getting Worse? Rethinking Regression Testing for Evolving LLM APIs. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*. 166–171.

[64] Microsoft. [n. d.]. microsoft/semantic-kernel: Integrate cutting-edge LLM technology quickly and easily into your apps. https://github.com/microsoft/semantic-kernel. Accessed 01-30-2024.

[65] M. Mozes, X. He, B. Kleinberg, and L.D Griffin. 2023. Use of llms for illicit purposes: Threats, prevention measures, and vulnerabilities. *arXiv preprint arXiv:2308.12833* (2023).

[66] V. Murali, C. Maddila, I. Ahmad, M. Bolin, D. Cheng, et al. 2023. CodeCompose: A Large-Scale Industrial Deployment of AI-assisted Code Authoring. *arXiv preprint arXiv:2305.12050* (2023).

[67] Nadia Nahar, Christian Kästner, Jenna Butler, Chris Parnin, Thomas Zimmermann, and Christian Bird. 2024. Beyond the Comfort Zone: Emerging Solutions to Overcome Challenges in Integrating LLMs into Software Products. *arXiv preprint arXiv:2410.12071* (2024).

[68] OpenAI. n.d.. Hello GPT-4o | OpenAI. https://openai.com/index/hello-gpt-4o/. Accessed: 2025-02-01.

[69] OpenAI. n.d.. Models - OpenAI API. https://platform.openai.com/docs/models. Accessed: 2025-02-01.

[70] OpenAI. n.d.. OpenAI o1 Hub | OpenAI. https://openai.com/o1/. Accessed: 2025-02-01.

[71] Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560* (2023).

[72] C. Parnin, G. Soares, R. Pandita, S. Gulwani, J. Rich, et al. 2023. Building Your Own Product Copilot: Challenges, Opportunities, and Needs. arXiv:2312.14231 [cs.SE]

[73] O. Parry, G.M. Kapfhammer, M. Hilton, and P. McMinn. 2022. Surveying the developer experience of flaky tests. In *Proc. of Int. Conf. on Softw. Eng.: Software Engineering in Practice*. 253–262.

[74] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. 2024. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices,

Applied Research Challenges and Opportunities. *arXiv preprint arXiv:2408.13296* (2024).

[75] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. 2023. Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback. arXiv:2302.12813 [cs.CL] https://arxiv.org/abs/2302.12813

[76] Gopi Krishnan Rajbahadur, Gustavo A. Oliva, Dayi Lin, and Ahmed E. Hassan. 2025. From Cool Demos to Production-Ready FMware: Core Challenges and a Technology Roadmap. arXiv:2410.20791 [cs.SE] https://arxiv.org/abs/2410.20791

[77] Sumedh Rasal and EJ Hauer. 2024. Navigating Complexity: Orchestrated Problem Solving with Multi-Agent LLMs. *arXiv preprint arXiv:2402.16713* (2024).

[78] Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. *arXiv preprint arXiv:2310.10501* (2023).

[79] RobBagby. [n. d.]. Retry Storm antipattern - Performance antipatterns for cloud apps — learn.microsoft.com. https://learn.microsoft.com/en-us/azure/architecture/antipatterns/retry-storm/. [Accessed 07-10-2024].

[80] Antonio Sabbatella, Andrea Ponti, Ilaria Giordani, Antonio Candelieri, and Francesco Archetti. 2024. Prompt Optimization in Large Language Models. *Mathematics* 12, 6 (2024), 929.

[81] Kunal Sawarkar, Abhilasha Mangal, and Shivam Raj Solanki. 2024. Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers. *arXiv preprint arXiv:2404.07220* (2024).

[82] S. Schoch, R. Mishra, and Y. Ji. 2023. Data Selection for Fine-tuning Large Language Models Using Transferred Shapley Values. *arXiv preprint arXiv:2306.10165* (2023).

[83] Aizaz Sharif, Dusica Marijan, and Marius Liaaen. 2021. Deeporder: Deep learning for test case prioritization in continuous integration testing. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 525–534.

[84] Tianhao Shen, Renren Jin, Yufei Huang, Chuang Liu, Weilong Dong, Zishan Guo, Xinwei Wu, Yan Liu, and Deyi Xiong. 2023. Large language model alignment: A survey. *arXiv preprint arXiv:2309.15025* (2023).

[85] W. Shi, A. Ajith, M. Xia, Y. Huang, D. Liu, et al. 2023. Detecting pretraining data from large language models. *arXiv preprint arXiv:2310.16789* (2023).

[86] H. Strobelt, A. Webson, V. Sanh, B. Hoover, J. Beyer, et al. 2023. Interactive and Visual Prompt Engineering for Ad-hoc Task Adaptation with Large Language Models. *IEEE Trans. on Visualization and Computer Graphics* 29, 1 (2023), 1146–1156.

[87] Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. 2024. Judging the Judges: Evaluating Alignment and Vulnerabilities in LLMs-as-Judges. *arXiv preprint arXiv:2406.12624* (2024).

[88] Unknown. 2023. SPDX 3.0 Dataset Profile. https://spdx.github.io/spdx-spec/v3.0/model/Dataset/Dataset/ Accessed: 2024-10-11.

[89] Cangqing Wang, Yutian Yang, Ruisi Li, Dan Sun, Ruicong Cai, Yuzhu Zhang, and Chengqian Fu. 2024. Adapting llms for efficient context processing through soft prompt compression. In *Proceedings of the International Conference on Modeling, Natural Language Processing and Machine Learning*. 91–97.

[90] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).

[91] Qiaosi Wang, Koustuv Saha, Eric Gregori, David Joyner, and Ashok Goel. 2021. Towards mutual theory of mind in human-ai interaction: How language reflects what students perceive about a virtual teaching assistant. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–14.

[92] Xin Wang, Yuwei Zhou, Hong Chen, and Wenwu Zhu. 2024. Curriculum Learning: Theories, Approaches, Applications, Tools, and Future Directions in the Era of Large Language Models. In *Companion Proceedings of the ACM on Web Conference 2024*. 1306–1310.

[93] Yuntao Wang, Yanghe Pan, Miao Yan, Zhou Su, and Tom H Luan. 2023. A survey on ChatGPT: AI-generated contents, challenges, and solutions. *IEEE Open Journal of the Computer Society* (2023).

[94] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Info Processing Systems* (2022).

[95] OpenDataology workgroup. 2022. OpenDataology. https://github.com/OpenDataology/. Accessed 02-07-2024.

[96] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, et al. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 [cs.AI]

[97] Siye Wu, Jian Xie, Jiangjie Chen, Tinghui Zhu, Kai Zhang, and Yanghua Xiao. 2024. How Easily do Irrelevant Inputs Skew the Responses of Large Language Models? *arXiv preprint arXiv:2404.03302* (2024).

[98] Yuchen Xia, Jiho Kim, Yuhan Chen, Haojie Ye, Souvik Kundu, Nishil Talati, et al. 2024. Understanding the Performance and Estimating the Cost of LLM

Fine-Tuning. *arXiv preprint arXiv:2408.04693* (2024).

[99] Jian Xie, Kai Zhang, Jiangjie Chen, Renze Lou, and Yu Su. 2023. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. *arXiv preprint arXiv:2305.13300* (2023).

[100] Eugene Yan, Bryan Bischof, Charles Frye, Hamel Husain, Jason Liu, and Shreya Shankar. [n. d.]. What We Learned from a Year of Building with LLMs (Part I) — oreilly.com. https://www.oreilly.com/radar/what-we-learned-from-a-year-of-building-with-llms-part-i/. [Accessed 04-10-2024].

[101] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. 2023. A survey on large language models for software engineering. *arXiv preprint arXiv:2312.15223* (2023).

[102] Zhimin Zhao, Abdul Ali Bangash, Filipe Roseiro Côgo, Bram Adams, and Ahmed E. Hassan. 2025. On the Workflows and Smells of Leaderboard Operations (LBOps): An Exploratory Study of Foundation Model Leaderboards. *IEEE Transactions on Software Engineering* (2025), 1–18. doi:10.1109/TSE.2025.3533972

[103] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, et al. 2023. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206* (2023).

[104] W. Zhou, Y.E. Jiang, L. Li, J. Wu, T. Wang, et al. 2023. Agents: An Open-source Framework for Autonomous Language Agents. arXiv:2309.07870 [cs.CL]

[105] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models Are Human-Level Prompt Engineers. arXiv:2211.01910 [cs.LG] https://arxiv.org/abs/2211.01910

[106] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. 2023. A survey on model compression for large language models. *arXiv preprint arXiv:2308.07633* (2023).

[107] Małgorzata Łazuka, Andreea Anghel, and Thomas Parnell. 2024. LLM-Pilot: Characterize and Optimize Performance of your LLM Inference Services. arXiv:2410.02425 https://arxiv.org/abs/2410.02425