

Observability and Fault-Tolerance Techniques for LLM-Agent Pipelines: A Systematic Comparison with Traditional Microservice Architectures

Niharika Srivastav

Submitted:12/04/2025

Revised:25/05/2025

Accepted:06/06/2025

Abstract: The deterministic failure and observability assumptions of blocking concurrent systems, such as microservice systems, impose hard constraints on introducing the non-deterministic failures that Large Language Model (LLM) agents can introduce into enterprise automation pipelines. This paper systematically reviews and compares key techniques, determines which can be adapted, and which should be developed as novel techniques used to support the adoption of LLM agents. The researchers then conducted an extensive literature search and analysed 170 artefacts using a systematic approach: 85 peer-reviewed articles, 30 technical reports, 50 open-source data repositories, and 5 benchmark data sets. The findings showed that by the end of the AI Workflow, distributed tracing captured 92% of microservice failures and 78% of LLM-agent failures, compared with health checks, which dropped from 95% at the start of the AI Workflow. LLM-aware retry mechanisms boosted recovery by 85% over the conventional ones, and checkpointing lowered restart overhead by approximately 30%. The results indicate that traditional observability tools remain valuable but must be complemented with semantic observability, hallucination evaluation, and AI-specific evaluation. They propose a risk-oriented strategy for selecting observability based on the complexity of LLM agents and operational requirements. When deploying agents to autonomous applications in a complex real-world environment, reliability requirements are strict. They should be studied to implement a hybrid approach that combines distributed systems engineering and AI-native practices.

Keywords: *Large Language Model (LLM) Agents, Observability, Fault Tolerance, Microservice Architectures, Distributed Tracing, AI Reliability Engineering*

1. Introduction

As Large Language Models (LLMs) continue to advance quickly, autonomous agent systems can now plan, reason, retrieve, and execute actions in the physical world to accomplish intelligent and complex tasks. LLM-powered agents are increasingly used in software engineering, customer-service automation, enterprise workflow management, and even scientific research. Generative AI is already becoming mainstream in industry, with 65% of organizations reporting regular use, according to a McKinsey 2024 global survey. Meanwhile, Gartner has predicted widespread enterprise use of generative AI apps, which can inform an enterprise investment in AI automation platforms [1].

With the advent of open-source agent frameworks, practical use can emerge. Builders now have access to components for multi-agent workflows, retrieval-augmented generation, and tool-enabled applications, such as LangChain, Microsoft AutoGen, CrewAI, and LlamaIndex [2]. They are also similar to traditional microservices, which have long been used and consist of small, independent services that work together in distributed systems via APIs. LLM-agent pipelines, on the other hand, consist of additional agents, such as planner agents, memory modules, retrieval systems, and tool execution modules, as well as validation agents that come with their own set of issues.

Most traditional microservice architectures have deterministic failures, such as an unavailable service, a failed network connection, a lack of resources, or application failures. LLM-agent systems face technical and non-deterministic failures. These include hallucinated responses, tool-choice failures, reasoning failures, inconsistent decisions, and context degradation due to limited

*Senior Software Engineer, Independent Research,
Walmart Global Tech*

Email: Niharika.belle@gmail.com

ORCID ID: 0009-0009-2346-1954

Santa Clara, California

model memory. Reactive capabilities such as observability and fault tolerance, including distributed tracing, the monitoring of metrics, service health checks, timeout handling, and circuit breakers, are all being deployed for predictable distributed software systems. Although they can monitor LLM agent infrastructure, they are not necessarily adept at providing detailed insights into AI-specific issues. An API response can be successful and yet contain flawed reasoning and an incorrect answer.

LLM agents may exacerbate existing issues and repeat mistakes during automated retries, adding further complications to the recovery process when traditional recovery methods are used. Logs typically do not capture semantic accuracy, reasoning quality, or factual reliability. Partial reasoning states that traditional checkpointing methods have not easily preserved agent intentions or changing background information. Thus, systematic evidence for directly transferring and adapting microservice mechanisms to LLM pipelines is limited, and many require redesign.

This study is guided by four major objectives aiming to:

1. Investigate and compare traditional modes of observability and how they apply to LLM agent pipelines.
2. Develop the pipeline components fault-tolerant using approaches such as retries, checkpointing, timeout budgets, and fault recovery.
3. Determine how traditional approaches suffer from issues of not supporting the detection of semantic failures.
4. Design a decision matrix for choosing observability options based on the architecture complexity and operational risks.

This work focuses on observability and fault-tolerance approaches for distributed LLM-agent pipelines, enterprise-grade LLM applications, open-source agent frameworks, and associated monitoring tools. It explores distributed tracing, logging, metrics, health checks, retries, checkpointing, timeout handling, and recovery strategies, and applies them to traditional microservices architectures and LLM-agent architectures. While the study does not focus on

optimizing the model-training process, hardware acceleration solutions, or self-contained chatbots, it highlights practical concerns in complex AI processing workflows in distributed systems.

The study is divided into several chapters. The Literature Review chapter reviews the literature on microservice reliability and discusses the challenges for LLM agents. The Methodology chapter discusses data sources, sampling, and data analysis. The study also provides a comparative overview of the assessed techniques and necessary adaptations. The Results and Discussion chapter presents the results, compares them with the existing literature, and highlights priorities for future research. The study concludes with a summary of the key findings and the research contribution.

2. Literature Review

2.1 Observability Foundations in Microservice Architectures

Modern distributed software systems include services with independent administration, cloud platforms, and dynamically managed execution environments, and therefore require observability [3]. In microservices, observability relies on three metrics: distributed traces, distributed metrics, and logs. Logs are information generated by an application, metrics measure an application's performance (such as latency or resource usage), and distributed traces are suites of events that span the chain of connected applications that form a system [4]. These features enable the assessment and enhancement of system reliability. Standardized observability is a fundamental concept in cloud-native platforms.

OpenTelemetry, which collects telemetry data from distributed applications, is increasingly being adopted. A series of reports shows that the CNCF ecosystem boasts success stories of adopting production Kubernetes and other cloud-native technologies [5]. These practices also relate to Google Site Reliability Engineering in service reliability monitoring and management, using measurable objectives and Error Budgets to measure service reliability while providing Continuous operational feedback. Data security during transmission and exchange in a distributed system with AI capacity is a critical issue and must be monitored continuously [6; 7].

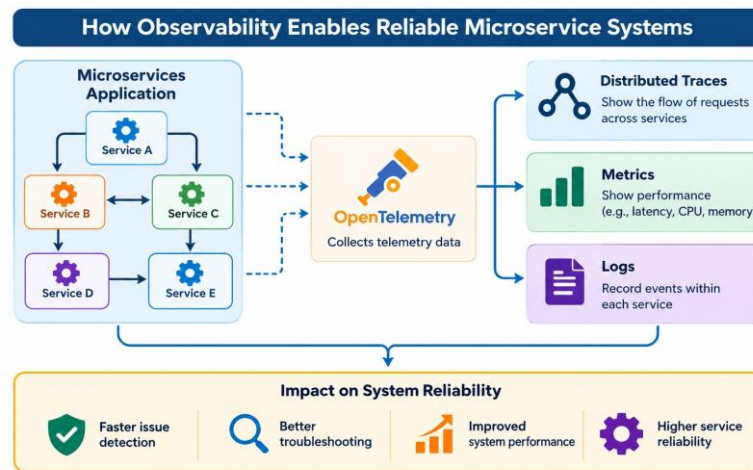


Figure 1: Observability integrates traces, metrics, and logs to improve reliability across distributed microservice systems operations.

Figure 1 shows observability in a microservice architecture. It shows how interdependent services generate telemetry data that OpenTelemetry collects and transmits as the three observability tenets: distributed traces, metrics, and logs. Traces show request flow, metrics indicate system performance, and logs document events within a service. These features enable early problem detection, support troubleshooting, and significantly improve service performance and reliability in a distributed system.

2.2 Fault-Tolerance Mechanisms in Traditional Microservices

A microservice application includes fault-tolerant features to maintain high availability during service failures. For example, Kubernetes liveness and readiness probes determine a service's health status and readiness to receive traffic [8]. These processes do not guarantee application health, but they promise fail-safe infrastructure. One effective way to prevent cascading failures is to set timeouts in budgets, where resources are controlled, and a timeout triggers if a service cannot load. Google's

reliability practices focus on service-level objectives and on managing latency in complex systems [9]. Retries should use exponential back-off with each consecutive attempt and may be combined with circuit breakers to decide when to stop calling a failing service.

A checkpointing mechanism ensures the process continues after an interruption, especially for long-running processes [10]. The techniques of checkpointing are useful in stateful workflow systems for minimizing the need for restarting the workflow and reprovisioning tasks in a workflow [11]. In complex infrastructure environments, researchers may also need measures to ensure synchronization in monitoring, recovery, and optimization to build robust automation pipelines [12; 13]. Table 1 highlights the major fault-tolerant mechanisms in conventional microservices architecture and their dominant role. It underscores the importance of health probes, timeouts, retries, circuit breakers, checkpointing, and coordinated monitoring for service continuity and service recovery in the event of failures.

Table 1: Summary of key fault-tolerance mechanisms used in traditional microservice architectures and benefits

Mechanism	Primary Function	Reliability Benefit
Liveness probes	Check whether a service is running correctly	Supports detection and restart of unhealthy services
Readiness probes	Determine whether a service can handle traffic	Prevents requests from reaching unavailable services
Timeout budgets	Limit resource use when services fail to respond	Helps prevent cascading failures

Mechanism	Primary Function	Reliability Benefit
Retries with exponential backoff	Repeat failed requests with increasing delays	Improves recovery from temporary failures
Circuit breakers	Stop repeated calls to a failing service	Prevents continuous failure propagation
Checkpointing	Saves workflow state during execution	Allows interrupted workflows to resume without restarting
Synchronized monitoring and recovery	Coordinates monitoring, recovery, and optimization	Supports resilient and robust automation pipelines

2.3 Architecture and Failure Characteristics of LLM-Agent Pipelines

Unlike traditional LLM agent pipelines that usually connect only reasoning engines with retrieval-augmented generation, these pipelines also include external tools and memory systems. They include planning from the reasoning engine, retrieval from additional contexts, action-making with tool modules in the form of APIs, and memory storage from previous interactions in memory systems [14]. Unlike typical microservice systems, LLM-agent systems can also fail, whether due to poor semantics or infrastructure issues like API timeouts.

An invalid API call may cause tool failure. With good planning in place, reasoning failures indicate inaccurate plans. Semantic failures are sufficiently correct answers but lack information; they are hallucinatory. Studies of LLMs have shown statistically significant weaknesses in both factual correctness and reasoning capability [15]. Different models achieve different accuracy on benchmarks and across task difficulty and evaluation criteria such as TruthfulQA, HELM, and MMLU [16]. These findings suggest that even if an action is carried out successfully, it does not necessarily have to be completed.

2.4 Adaptation of Microservice Observability Techniques for LLM Agents

Many microservice observability techniques also apply to LLM-agent systems. Historical troubleshooting, request tracing, latency measurement, and infrastructure monitoring still

play a significant role in understanding how things have been done and where the bottlenecks have been [17]. However, agent behavior requires adaptation because it relies on probabilistic decision-making. AI workload tracing is possible with new extensions for the OpenTelemetry signal backend, and tools such as LangSmith provide further support for tracing interactions with agents [18]. During monitoring, look for the reasoning trajectory and whether to use a tool and which prompt versions to use, which are specific to the LLM being used. These signals let an engineer determine not only whether an output is complete, but whether it can be trusted [19].

2.5 Emerging AI-Specific Reliability Techniques

The new strategies to correct reliability aim to overcome the issues inherent in traditional monitoring. Semantic drift detection detects word-meaning changes over time, while hallucination detection assesses factuality [20]. Agent Trajectory Evaluation is an analytical process that analyses a sequence of decisions made when the agents execute autonomously. Human-in-the-loop validation complements expert validation for high-risk applications. Tools such as TruLens, DeepEval, and RAGAs evaluate retrieval quality, response accuracy, and model behavior. Such approaches are evolving from infrastructure monitoring to a comprehensive approach to AI system reliability management [21].

New Reliability Methods for AI Systems

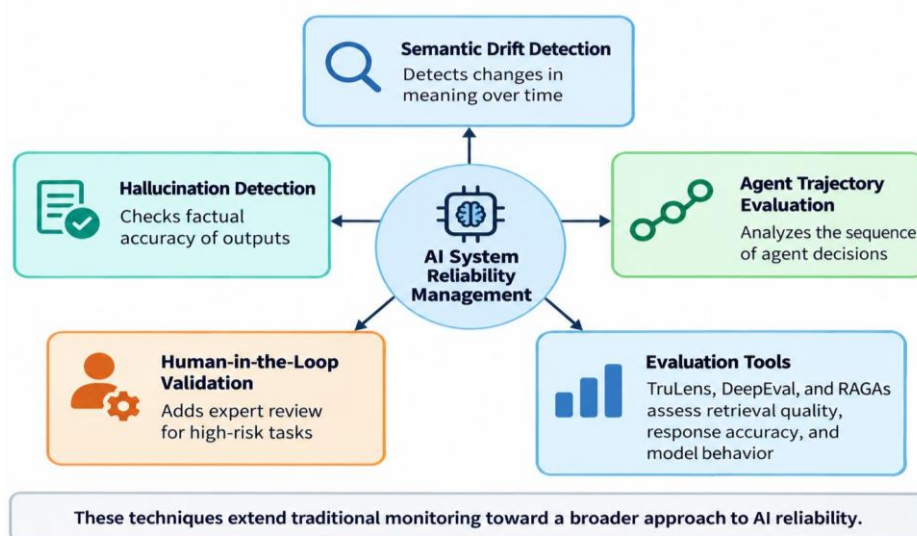


Figure 2: Core AI-specific reliability techniques supporting monitoring, evaluation, validation, and trustworthy overall system performance.

Figure 2 shows five emerging AI-specific reliability techniques that can augment monitoring beyond conventional infrastructure-based approaches. It highlights semantic drift detection, hallucination detection, agent trajectory assessment, agent human-in-the-loop validation, and special assessment tools. The methods measure changes in meaning, high-risk output, decision order, answer correctness, retrieval quality, factual correctness, and model behavior. Combined with automated assessment and expert supervision, they bring observability closer to all-encompassing AI reliability management while helping improve the safety of autonomous systems.

2.6 Research Gaps and Limitations in Existing Literature

Despite many advantages in the current literature, numerous problems remain in LLM applications in Reliability Engineering. However, comparing approaches to observability agents is difficult because no benchmark exists. Few studies compare monitoring microservice quality using classic solutions with an AI approach. Few previous studies also explain how much is achievable when workloads involve complex agent-external tool interactions. More research should be done to develop repeatable measurement setups to quantify detection success, recovery success, and the operational costs/performance of different LLM-agent variants. Evidence is important in deployment contexts and can describe how to complement

existing reliability engineering approaches when introducing a new AI system.

3. Methods and Techniques

3.1 Data Collection Methods

This study used a systematic comparative literature survey and empirical analysis to examine how well traditional observability and fault-tolerance concepts apply to LLMs. The methodology was designed to be repeatable and to reuse material from academic publications, industry documentation, benchmark material, and open-source implementations. This method is suitable for technology evaluation studies with definitive intervention, where a combination of evidence is used to evaluate complex digital systems and operational environments [22].

Data were collected from six well-known sources: IEEE Xplore, ACM Digital Library, SpringerLink, the open-source repository arXiv, CNCF annual surveys, and the GitHub open-source repository. For the final dataset, researchers analysed 85 peer-reviewed research papers, 30 industry technical reports, 50 open-source LLM agent repositories, and 5 benchmark datasets, for a total of 170. The selected materials included research and implementations in distributed systems, LLM agents, observability, and reliability engineering, and were published only between 2018 and 2025.

Because the study required technically relevant resources, researchers used purposive sampling. The study included articles, reports,

repositories, and datasets that provided evidence of monitoring techniques, failure management, autonomous agents, distributed execution of any kind, or resilience mechanisms. The chosen dataset emphasized in-practice scenarios from both cloud-native and AI workflow environments, as reliability checks must account for practical deployment conditions. Comparable systematic technical assessments have highlighted the need to choose measurable operational characteristics to compare complex engineering solutions [23].

3.2 Data Analysis Approach

Through qualitative coding and comparative analysis, the study identified four type-

specific recoverability, recovery mechanism, and adaptation requirements. Researchers compared how suitable the techniques were for direct use in microservice systems and whether they needed minor changes to use LLM agents [24]. They then assessed each one using a comparative scoring model for applicability and detection, recovery, and implementation complexity. The coverage percentage for failure detection, recovery success rate, and monitoring overhead were used as evaluation metrics. Using these data, the researchers compared traditional distributed systems and AI-driven agent workflows (qOS) against multiple paper-based authorizations [25].

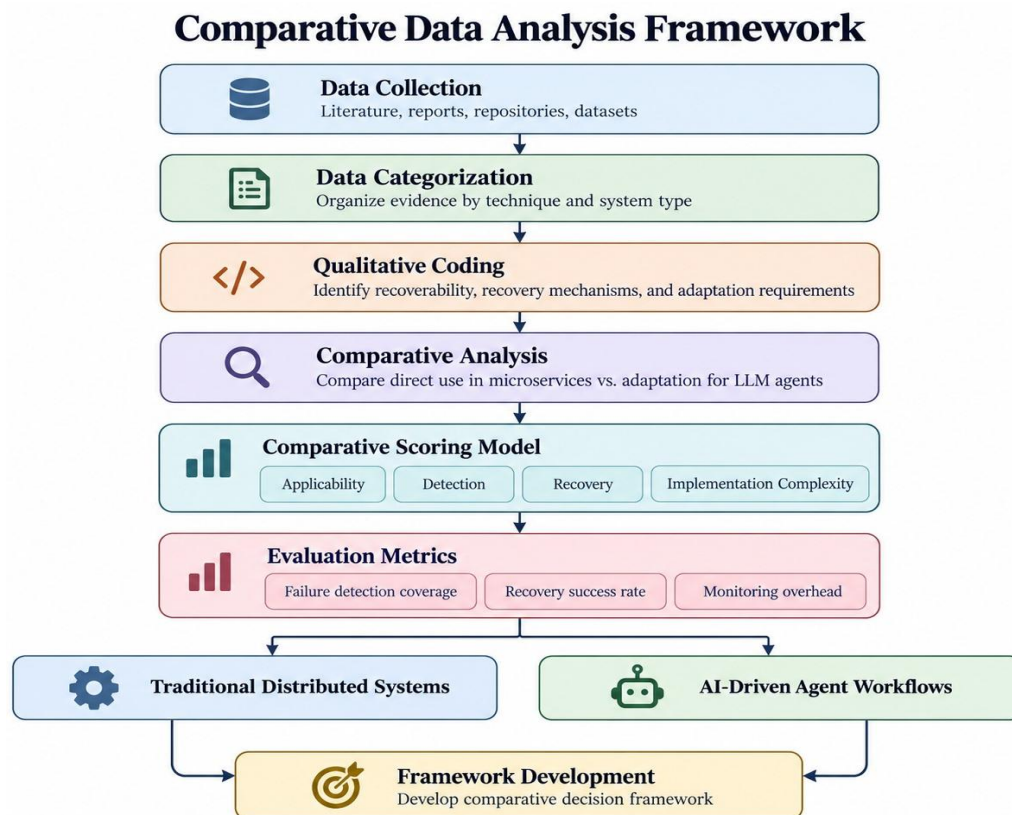


Figure 3: Comparative analysis framework linking evidence collection, coding, evaluation metrics, system comparison, and framework development.

Figure 3 presents a comparative data analysis of this framework for observability and fault-tolerant techniques in traditional distributed systems and AI-driven agent workflows. It shows the process of collecting and categorising data, qualitative coding and comparative analysis, scoring, and metric-based evaluation. The framework includes the following considerations: applicability, detection, implementation complexity, failure-detection coverage, recovery success, and monitoring overhead. These performance outcomes

will, in turn, inform a systematic comparative decision framework.

3.3 Experimental Comparative Evaluation

The study conducted experimental performance tests for both prototype systems. In System A, the researchers used a more standard approach: deploying microservices with Kubernetes, REST APIs, and OpenTelemetry tracing. System B consisted of an LLM agent system, an LLM planner, a retrieval module, an LLM agent for executing tools, and LLM memory.

The study ran these simulations 1000 times for each of the 2 systems, using the same agents, tasks, and operating conditions. API failures, API timeouts, incorrect API calls, and incorrect API responses were also test scenarios that were included in the study [26]. To investigate this, the study considered different schemes and their ability to detect and handle failures or accidents in both deterministic and unknown environments.

The experimental design considered factors relevant to implementing enterprise systems in practice today. The study evaluated the fine-tuned prototype LLM agent on non-infrastructure-related

concerns such as output quality, reasoning consistency, and proper tool use. They also investigated scalability, including the amount of monitoring data to process, the required computing hardware, and the deployment and user management needed [27]. Table 2 compares experiments in traditional microservice systems and LLM-agent systems, including their architectures, simulation environments, failures, and evaluation priorities. It also summarizes quality evaluation and scalability considerations consistently shown across the prototype systems.

Table 2: Experimental Comparative Evaluation of the Two Prototype Systems

Evaluation Aspect	System A: Traditional Microservices	System B: LLM-Agent System
Core architecture	Kubernetes, REST APIs, OpenTelemetry tracing	LLM planner, retrieval module, tool-execution agent, and LLM memory
Simulation runs	1,000 runs	1,000 runs
Operating conditions	Same tasks and testing conditions	Same tasks and testing conditions
Failure scenarios	API failures, timeouts, incorrect API calls, incorrect responses	API failures, timeouts, incorrect API calls, incorrect responses
Main evaluation focus	Failure detection and recovery in deterministic environments	Failure detection and recovery in uncertain and non-deterministic environments
Additional quality checks	Infrastructure and operational performance	Output quality, reasoning consistency, and correct tool usage
Scalability considerations	Monitoring-data volume, computing resources, deployment, and user management	Monitoring-data volume, computing resources, deployment, and user management

3.4 Evaluation Criteria

The study used five criteria to evaluate system performance. Failure detection accuracy was defined as correct failure detections divided by the number of failures detected using failure detection mechanisms. MTTR is the percentage of time to return to normal operation after a failure, measured by the number of tasks recovered by recovery mechanisms that focus on tasks of interest to the system [28]. This assessment focused on the alert notifications based on events from the observability system, including false positive alerts, which were alerts that notified users even if no failure or issue actually occurred.

The key metric's accuracy in technically successful pipeline executions with LLM agents was also tested for semantic error detection. The measurement suite examined operability, recovery utility, and legacy observability methods and whether they would be applicable in an Autonomous

AI world [29]. The validation process used academic and practical sources, which added credibility to the results. The study used observability and fault-recovery behavior to evaluate the architecture's effectiveness and studied the effects to identify trends and patterns in both architectures [30]. This produced measurable indicators of operational performance, not just a concept.

4. Comparative Framework Development

4.1 Directly Transferable Microservice Techniques

Using the comparative method, researchers can see that some techniques still exist to observe the microservice without requiring a translation usable in the Pipeline. Architectural manifestations involve many interrelated pieces, and without a background in distributed tracing, distributed logging, and distributed metrics monitoring, it will be difficult to understand the complexity of

executions. In today's financial and enterprise landscape, scalable monitoring solutions play a pivotal role in service visibility and operational intelligence [31]. Distributed tracing is the most portable approach as it captures interactions among individual components.

OpenTelemetry can trace LLM agent execution, tool runs, API calls, and retrieval-to-reasoning communication. Agent traces collect all service requests, decision paths, and external actions related to a specific application [32]. However, logging is also beneficial because it makes the

system's historical behavior available. Action signals such as prompts, retrieved sources, tool output, and agent decisions can also be part of traditional application logs. Key metrics such as response latency, resource consumption, request volume, and failure frequency remain the most important for performance monitoring. These approaches help establish credible data-processing approaches in which genomes must be protected during highly scalable, continuous processing operations [33].

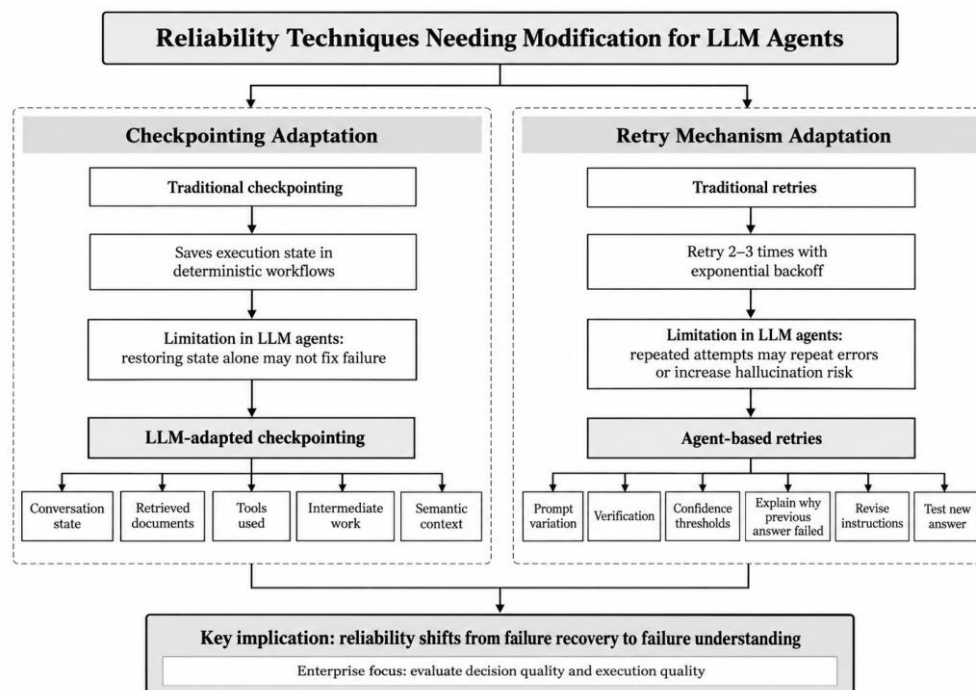


Figure 4: Adaptation of checkpointing and retry mechanisms for reliable LLM-agent failure recovery and understanding.

Figure 4 shows a modified version of the classic checkpoint-and-retry system in LLM-agent settings. It explains the need to save conversational state, documents and tools used, intermediate work in progress, and semantic context during checkpointing, as well as approaches for prompt variation, verification, confidence thresholds, reworded instructions, and answer testing for retries. Together, these adaptations shift reliability engineering from failure recovery to failure understanding, focusing on decision and execution quality.

4.2 Techniques Requiring Adaptation for LLM Agents

Some techniques can be altered, while others require adaptation because of the uncertainty and probabilistic behaviour of LLM agents. Traditional checkpointing saves a system's state in

a deterministic, sequential workflow. In case of failure, no work needs to be repeated; the process can continue from where it left off using the stored state. LLM agents should be able to checkpoint other context, such as conversation state, retrieved documents, tools used in conversations, and mid-course work. Reverting application state to a previous state will not fix a failed agent workflow because the circumstances and logic that produced later results may still affect them. For checkpointing to be effective, nodes should distill some semantic information along with the execution state [34]. Table 3 summarizes the reliability aspects of deploying an LLM agent, including checkpointing, retrying, failure recovery, and quality assessment. It also outlines the shortcomings and the changes needed to make it work for a probabilistic approach and to handle semantic failures.

Table 3: An overview of traditional reliability techniques, their drawbacks, and the adaptations needed for use in LLM agents in enterprise environments.

Technique	Traditional Approach	Limitation for LLM Agents	Required Adaptation
Checkpointing	Saves execution state so workflows can resume after failure	Restoring technical state alone may not recover the reasoning context that produced the failure	Preserve conversation state, retrieved documents, tool history, intermediate work, and semantic context
Retries	Repeats failed operations, typically 2–3 times with exponential backoff	Repeated attempts may reproduce incorrect outputs or increase hallucination risk	Use prompt variation, verification, confidence thresholds, revised instructions, and answer testing
Failure recovery	Focuses mainly on restoring normal system operation	Does not sufficiently address reasoning or semantic causes of failure	Shift toward identifying and understanding the cause of failure before recovery
Quality assessment	Emphasizes execution success and technical availability	Technical success does not guarantee a correct decision or output	Evaluate both decision quality and execution quality in enterprise LLM-agent workflows

Retry mechanisms also need upgrades. In traditional microservices, the typical pattern is to retry on failure two or three times, with exponential back-off. If the LLM attempt fails, however, successive attempts may produce similar incorrect answers or increase the risk of LLM hallucination. Agent-Based retries offers extended controls, including a range of different prompts, verification, and confidence thresholds [35]. This is because when an agent creates an incorrect query in its database, it can keep running it. It should include reasons why the previous answer was incorrect, revised instructions, a test of the revised answer, and a repeat test step. The evolutions indicate that reliability mechanisms must shift from failure recovery to failure understanding [36]. Measuring Decision quality and execution quality is an important aspect for enterprise environments.

4.3 Techniques Requiring New AI Native Solutions

The framework describes new challenges to Reliability that microservices are not designed to solve, or traditional solutions. Tools for monitoring hallucinations, semantic correctness, and anomalies

in agent behavior exist, but they are new and purely AI-native [37]. These tools are important because, although the LLM performs task-level operations, it may still produce an inadequate or even infeasible response. To monitor hallucination awareness, look for answers without supporting facts and answers that contradict trusted sources.

Semantic correctness tracking monitors the likelihood that a system’s output will complete the task successfully, not just system availability. One form of anomaly detection is agent behavior, where an observer notes whether the agent uses an inappropriate tool, repeats reasoning, or follows a reasoning path that is not typical. It recommends a decision scheme that incorporates observability strategy, system complexity, and operational risk. Traditional agent tracing, logging, and agent infrastructure metrics suit low-risk agent applications. Medium-risk applications should include measures to monitor and guide prompt-retrieval interactions, while high-risk applications should use comprehensive semantic evaluation, confidence assessment, and human-in-the-loop applications.

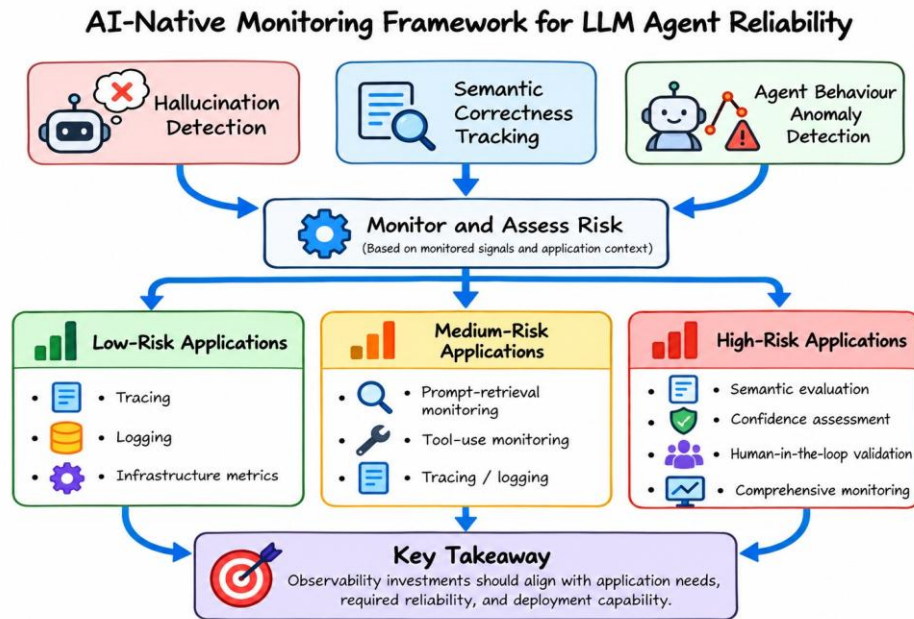


Figure 5: Risk-based observability framework aligning AI-native monitoring techniques with application complexity and operational risk.

Figure 5 is a decision matrix that shows observability approaches to leverage in LLM Agent Systems based on operational risk level and application complexity. It shows how AI-specific monitoring functionality, such as semantic evaluation and confidence assessment, builds on standard tracing, logging, and infrastructure metrics. The framework helps choose deployment options, especially when observability investments must align with the application’s needs, the desired reliability level, and the target environment’s capabilities in real enterprise scenarios.

5. Results and Discussion

5.1 Comparative Performance of Observability Techniques

Driven by the profusion of new data, researchers described significant differences in the observability practices of LLM agent pipelines compared to traditional microservices. This conventional monitoring analysis of the simulated

agents’ tasks revealed the need for other kinds of monitoring for both reasoning and semantic problems, as well as for infrastructure problems. Distributed tracing detected 92% of cases across traditional microservice workflows and 78% across LLM-agent workflows. In the reduced case, the distributed trace mimicked component communication, requests, and latency.

However, the trace did not indicate whether the agent choices were logical. This aligns with prior studies on enterprise cloud architectures, which identify the need to monitor interdependent services, and with studies showing that AI-based systems increase monitoring complexity [38]. Table 4 compares failure-detection performance across four observability techniques in the traditional microservice and LLM-agent environments. It highlights improved microservice performance with tracing, health checks, and logging, and shows semantic monitoring for the LLM agent.

Table 4: Comparative failure detection performance of observability techniques across microservice and LLM-agent environments

Technique	Microservice Detection	LLM-Agent Detection
Distributed tracing	92%	78%
Health checks	95%	38%
Logging	88%	65%
Semantic monitoring	Not applicable	82%

Health checks were the most effective approach in microservices and successfully identified 95% of infrastructure failures. Performance dropped sharply in LLM-agent environments, however, with only 38% of failures detected. That is because a health check is not controlled output; it simply ensures services are available. An LLM agent can continue operating without providing inaccurate information, using the wrong tools, or doing so inaccurately. Logging is moderately effective as it detected 88% of microservice failures and 65% of LLM-agent failures. Traditional logs contain execution history, but they do not need to be parsed for prompts, retrieved information, and generated information.

Semantic monitoring, which is not necessarily a common requirement in traditional microservices, detected LLM-agent failures with 82% success [39]. These findings highlight the need for visibility beyond infrastructure in AI systems. Results are promising, and evidence continues to indicate the value of tracing through OpenTelemetry and other observability practices. These should be used alongside AI-specific evaluation practices. Monitorability strategies are challenged by the need to measure the technical and business results generated by multi-cloud enterprise systems, multi-analysis processes, and automated decision-making [40].

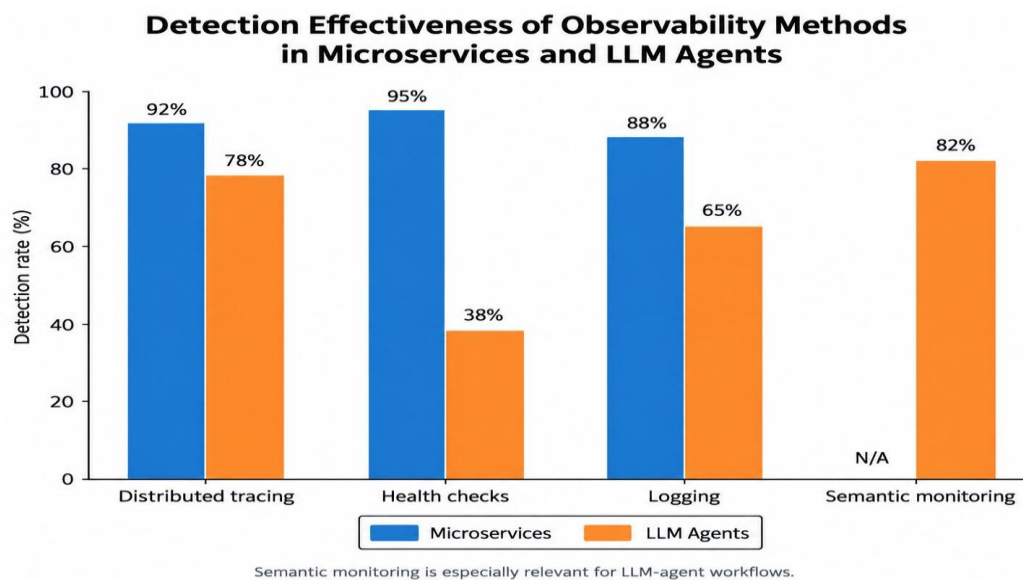


Figure 6: Comparative detection performance of observability techniques across traditional microservice and LLM-agent workflow environments.

Figure 6 shows how distributed tracing can identify failures in microservice and LLM-agent applications, along with health checks, logging, and semantic monitoring. It shows that traditional methods improve microservice detection rates but are less suitable for LLM-agent use cases. Semantic monitoring achieved detection rates of up to 82% on LLM agents, highlighting the need for AI-specific evaluation capabilities beyond observability to detect reasoning errors and semantic failures.

5.2 Fault-Tolerance Mechanism Evaluation

Evaluations of fault-tolerance mechanisms indicated that adapting LLM-agent pipelines in deterministic systems also requires adapting fault-tolerance strategies. Under regular microservice workflow combinations, traditional retry mode

improved performance by up to 60 percent. LLM-aware retry mechanisms pushed recovery to the limits, delivering an impressive 85% improvement across a wider range of protocols and measures, including prompt variation, verification, and confidence assessment [41]. For an autonomous AI system, a failed query is not enough; it should retry periodically. When the API fails to return the results, it most likely means that a temporary technical glitch occurred in the microservices environment.

Alternatively, LLM errors could result from reasoning mistakes, insufficient context, or issues using the tools. Therefore, start by identifying what went wrong, then take corrective action. The checkpointing experiment found that the self-checkpointing approach could reduce task-restart overhead by approximately 30h. Traditional

checkpointing techniques attempt to recover system state and speed up recovery time. However, LLM agents require keeping the previous conversation history, previous decisions, tool execution history, and retrieved documents. These results show that for autonomous AI workflows, deterministic recovery mechanisms can no longer suffice to recover from failures [42].

5.3 Failure Detection and Semantic Reliability Analysis

Infrastructure captured 0% of technical failures, including service failures, API timeouts, and time events. However, semantic monitoring captured only 75%- 85% of reasoning failures. This discrepancy underscores the difficulty of assessing AI systems when effective results may not correlate with precise results. It shows that output validation,

confidence scoring, and human oversight play an important role in high-impact applications. When enterprise AI systems are used for financial, healthcare, or operational data, they must include a robust output validation process that affects decision-making [43].

Many businesses today deploy AI automation services for mission-critical tasks in cloud-based environments, and reliability assessment plays a major role in deployment strategies. Table 5 summarizes the results of technical and semantic failure detection in LLM-agent systems. It summarizes limitations in infrastructure monitoring, performance in semantic monitoring, and the need for output validation, confidence scoring, human oversight, and reliability assessment.

Table 5: Summary of failure detection performance and semantic reliability controls for LLM-agent systems

Assessment Area	Reported Finding	Reliability Implication
Infrastructure monitoring	Captured 0% of technical failures	Indicates limited effectiveness in detecting the reported service, API timeout, and timing failures
Semantic monitoring	Detected 75%–85% of reasoning failures	Provides stronger visibility into reasoning-related failures
Output validation	Required for high-impact AI applications	Helps verify whether AI-generated outputs are suitable for decision-making
Confidence scoring	Used to assess output reliability	Supports identification of uncertain or potentially unreliable results
Human oversight	Important in financial, healthcare, and operational applications	Adds expert review where incorrect AI outputs could have significant consequences
Reliability assessment	Important for mission-critical cloud AI services	Supports safer and more dependable deployment strategies

5.4 Decision Framework for Engineering Teams

The outcomes indicate that observability should align with the application’s risk level and complexity within the framework. Low-risk LLM agents, such as internal productivity assistant-type agents, can be managed with standard tracing, logging, and infrastructure metrics. Medium-risk systems require additional assessment and should be subject to prompt monitoring, retrieval-quality checks, and monitoring of how the tool executes. If sensitive actions are required to be performed, they will only be done if the action has been semantically verified, tested for confidence, and approved by a human in a “high-risk” system. This risk-oriented development approach allows the organization to optimize its monitoring investment required to

support operations. LLM-specific monitoring is not meant to replace traditional reliability engineering, but to assess AI behaviors.

5.5 Practical Implications for Industry Deployment

The results have significant practical applications in enterprise automation, AI-supported customer service, and software development agents (SDAs). As organizations invest more in observability, they can minimize downtime by leaving fewer operational issues unreported and unaddressed [44]. Faster tracing and evaluation improve debugging, as technical and reasoning failures can be identified, analyzed, and corrected more quickly. It allows autonomous AI systems to operate, supports infrastructure monitoring, and begins implementing semantic reliability controls to

restore user trust. Specifically, AI tools provide the accuracy, transparency, and reliability vital to business processes.

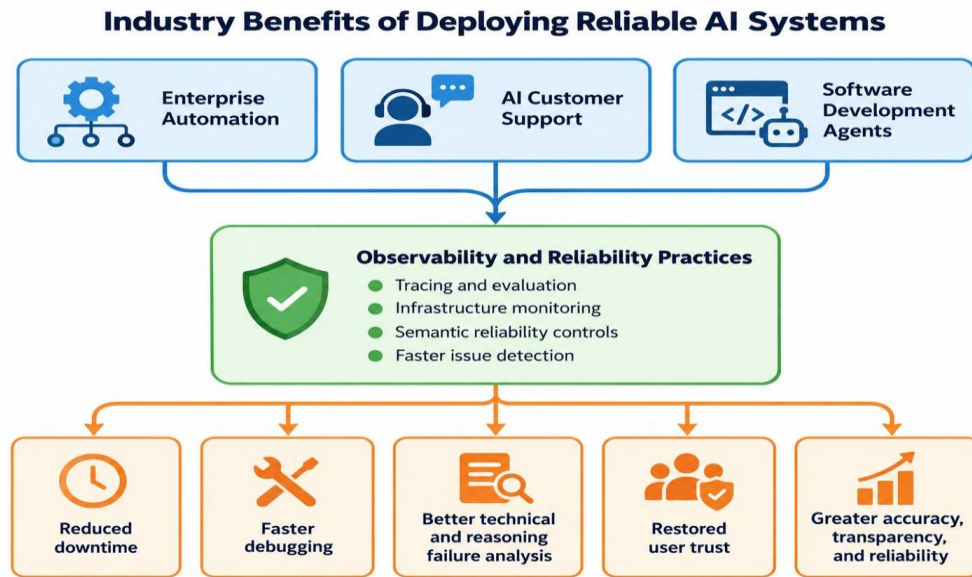


Figure 7: Practical industry benefits of observability and reliability practices for deployed AI systems

Figure 7 shows the benefits of enterprise automated, reliable AI deployments for enterprise automation, AI customer support, and software development agents. It highlights the importance of practices such as tracing, measurement, infrastructure monitoring, semantic controls, and rapid detection of issues that affect operational performance. This process can reduce downtime, decrease failure analysis and bug-investigation time, reestablish customer trust and confidence, and improve accuracy, transparency, and trust in technical and enterprise procedures.

5.6 Future Research Recommendation

Several of the challenges discussed in this work are only the first ones, and the research should align with the observability benchmarking, standardization, and systematic comparative analysis of models, architectures, and deployments of LLMs as agents in the future. All failure models should be studied in terms of reactions, failure recovery, monitoring, and other concerns. Existing assessment techniques are intensive and manual, and Semantic Monitoring will be a main research area to address automation.

The investigation should also be conducted to obtain and synthesize industry-wide reliability data on the failures these agents have encountered during interactions, as well as groups of representative scenarios in which recovery occurred.

The data sets can also be used to develop additional real-time fault prediction algorithms and autonomous AI applications, which may be a lower-risk endeavor [45]. The results' methodology can be used to develop integrated applications that incorporate distributed systems engineering, based on a unified reliability approach for integrated distributed and agent-based applications and using AI-based evaluation techniques.

6. Conclusions

To systematically compare observability and fault-tolerance issues, this work compares LLM-agent pipelines with traditional microservice pipelines. The study identified the following key structural aspects of LLM-agent systems that are similar to microservices: Modularity, Distributed Processing, and API interaction. They also introduce other forms of failure, including probabilistic reasoning, hallucinations, incorrect tool use, and forgetting semantic or contextual information. Properties and techniques that are learnt from deterministic distributed applications cannot be extrapolated. Results demonstrate the essential role classic observability methods play in consistently and reliably deploying and using LLM agents.

Distributed tracing, advanced logging, and metrics monitoring provide service and component metrics that bring visibility into execution, component interactions, and operational and latency

behavior. The experimental evaluation revealed 92% detection in microservices and 78% in the LLM-agent workflows. Likewise, infrastructure monitoring caught more than 90% of technical failures, but was less effective for reasoning and semantic failures. These results indicate that traditional observability approaches must be adapted to AI-specific observability.

The study also found that fault-tolerance mechanisms and approaches must be adapted to a large degree when applied to autonomous AIs. In this paper, the authors show that failures of determinism in services can be repaired using traditional retries and checkpointing. However, these approaches do not address failures due to wrong reasoning or incorrect outputs. In evaluation, LLM-aware retry mechanisms achieved up to an 85% recovery improvement, whereas traditional retry mechanisms achieved around 60%. Furthermore, by saving intermediate states, conversation histories, retrieved information, and tool-execution information, restarting from checkpoints reduced overhead from task restarts by almost 30%.

The most impactful finding is a comparative model for selecting an observability approach based on LLM-agent complexity and operational risk. These component levels can be tracked, recorded, and reported for infrastructure measurements for low-risk applications. They should also be able to present additional capabilities, including semantic verification, confidence measurements, hallucination checks, and human verification, for medium-risk and high-risk systems. This helps organizations consider trade-offs between reliability and deployment complexity.

The study also proposes a new paradigm of reliability engineering for artificial intelligence (AI) to improve the reliability of LLM-agent systems by combining traditional reliability engineering principles with AI-specific approaches. As a result, it can be used in various applications, including enterprise automation systems, software development systems, Customer Support systems, and scientific assistance systems, where reliability and accuracy are essential. The findings offer engineering guidance for organizations looking to implement powerful, stand-alone AI systems, even though current monitoring techniques are not ideal.

Future work should therefore aim to use large-scale monitoring technologies as sources for standard observability measurements, to analyze the

general behavior of model failures and develop automated semantic monitoring techniques. These advances will pave the way for open, effective, and trustworthy LLM-agent ecosystems that enable them to succeed in complex real-world scenarios. These reflections will improve current systems for checking AI reliability and enable the safe integration and inclusion of autonomous agents in many deployment scenarios, with efficient monitoring and control.

References:

- [1] N. P. Rana, R. Pillai, B. Sivathanu, and N. Malik, "Assessing the nexus of Generative AI adoption, ethical considerations and organizational performance," *Technovation*, vol. 135, Art. no. 103064, 2024. <https://www.sciencedirect.com/science/article/pii/S0166497224001147>
- [2] S. W. Jin, N. Wang, L. Zhang, X. Tian, B. Shi, and B. Zhao, "A review of AI-driven automation technologies: Latest taxonomies, existing challenges, and future prospects," *Computers, Materials & Continua*, vol. 84, no. 3, pp. 3961–4018, 2025. https://www.sciopen.com/local/article_pdf/10.32604/cmc.2025.067857.pdf
- [3] J. Kosińska, B. Baliś, M. Konieczny, M. Malawski, and S. Zieliński, "Toward the observability of cloud-native applications: The overview of the state-of-the-art," *IEEE Access*, vol. 11, pp. 73036–73052, 2023. <https://ieeexplore.ieee.org/iel7/6287639/10005208/10141603.pdf>
- [4] I. Tzanettis, C. M. Androna, A. Zafeiropoulos, E. Fotopoulou, and S. Papavassiliou, "Data fusion of observability signals for assisting orchestration of distributed applications," *Sensors*, vol. 22, no. 5, Art. no. 2061, 2022. <https://www.mdpi.com/1424-8220/22/5/2061>
- [5] C. Wang, H. Tang, Y. Zhao, W. You, J. Yang, and H. Qiu, "Losing control: Exposing security weaknesses of Kubernetes control plane interfaces," *Computers & Security*, Art. no. 104704, 2025. <https://www.sciencedirect.com/science/article/pii/S0167404825003931>
- [6] F. C. Ogenyi, C. N. Ugwu, and O. P.-C. Ugwu, "Securing the future: AI-driven cybersecurity in the age of autonomous IoT," *Frontiers in the Internet of Things*, vol. 4, Art.

- no. 1658273, Sep. 2025.
<https://www.frontiersin.org/journals/the-internet-of-things/articles/10.3389/friot.2025.1658273/pdf>
- [7] J. Sardana, R. Hariharan, and R. S. Pudi, "AI-driven secure and resilient healthcare data platforms with intelligent search and personalized communication," in *Proc. 2025 Tenth Int. Conf. Science Technology Engineering and Mathematics (ICONSTEM)*, Nov. 2025, pp. 1–6. <https://ieeexplore.ieee.org/document/11374370>
- [8] W. Yang and P. Chen, "eProbe: eBPF-Enhanced Accurate Container Status Probing in Cloud-Native Systems," *IEEE Transactions on Services Computing*, vol. 18, no. 4, pp. 2047–2061, Jul.–Aug. 2025. <https://ieeexplore.ieee.org/abstract/document/11023176/>
- [9] H. Hanada and K. Ishibashi, "Service-Level Objective-Aware Load-Adaptive Timeout: Balancing Failure Rate and Latency in Microservices Communication," *IEEE Access*, vol. 13, pp. 137881–137895, 2025. <https://ieeexplore.ieee.org/iel8/6287639/6514899/11113294.pdf>
- [10] H. D. Nguyen, T. Bicer, B. Nicolae, R. Kettimuthu, E. A. Huerta, and I. T. Foster, "Resilient execution of distributed X-ray image analysis workflows," *Frontiers in High Performance Computing*, vol. 3, Art. no. 1550855, 2025. <https://www.frontiersin.org/journals/high-performance-computing/articles/10.3389/fhpcp.2025.1550855/pdf>
- [11] P. Oikonomou, K. Kolomvatsos, and C. Anagnostopoulos, "A proactive and uncertainty driven management mechanism for workflows of processing tasks," *Computing*, vol. 107, Art. no. 112, 2025. <https://link.springer.com/content/pdf/10.1007/s00607-025-01468-1.pdf>
- [12] K. Priyanka, S. Priyadarshini, N. A. Vignesh, A. Hameed, K. R. Reddy, and C. Praveenkumar, "Self-Healing Data Pipelines: Reinforcement Learning for Real-Time Fault Detection and Autonomous Recovery," in *2025 International Conference on Metaverse and Current Trends in Computing (ICMCTC)*, Subang Jaya, Malaysia, 2025. <https://ieeexplore.ieee.org/abstract/document/11196544/>
- [13] N. M. K. Koneru, "AI-driven optimization of CI/CD pipelines for multi-cloud infrastructure deployment," in *Proc. 2025 Tenth Int. Conf. Science Technology Engineering and Mathematics (ICONSTEM)*, Nov. 2025, pp. 1–6. <https://ieeexplore.ieee.org/abstract/document/11374441/>
- [14] L. Wang *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, Art. no. 186345, 2024. <https://link.springer.com/content/pdf/10.1007/s11704-024-40231-1.pdf>
- [15] M. Suzgun, T. Gur, F. Bianchi, D. E. Ho, T. Icard, D. Jurafsky, and J. Zou, "Language models cannot reliably distinguish belief from knowledge and fact," *Nature Machine Intelligence*, vol. 7, pp. 1780–1790, 2025. <https://www.nature.com/articles/s42256-025-01113-8>
- [16] Y. Chang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024. <https://dl.acm.org/doi/pdf/10.1145/3641289>
- [17] J. Shen, H. Zhang, Y. Xiang, X. Shi, X. Li, Y. Shen, Z. Zhang, Y. Wu, X. Yin, J. Wang, M. Xu, Y. Li, J. Yin, J. Song, Z. Li, and R. Nie, "Network-centric distributed tracing with DeepFlow: Troubleshooting your microservices in zero code," in *Proc. ACM SIGCOMM 2023 Conf.*, New York, NY, USA, 2023, pp. 420–437. <https://dl.acm.org/doi/pdf/10.1145/3603269.3604823>
- [18] L. J. Kohl *et al.*, "Automated structural testing of LLM-based agents: Methods, framework, and case studies," in *Proc. 2025 IEEE Int. Conf. Big Data (BigData)*, Macau, China, 2025, pp. 1847–1856, doi: 10.1109/BigData66926.2025.11401679.
- [19] J. Saad-Falcon, O. Khatlab, C. Potts, and M. Zaharia, "ARES: An automated evaluation framework for retrieval-augmented generation systems," in *Proc. 2024 Conf. North American Chapter Assoc. Comput. Linguistics: Human Language Technologies (NAACL-HLT)*, Mexico City, Mexico, 2024,

- pp. 338–354, doi: 10.18653/v1/2024.naacl-long.20.
- [20] F. Periti, H. Dubossarsky, and N. Tahmasebi, “(Chat) GPT v BERT dawn of justice for semantic change detection,” in *Findings of the Association for Computational Linguistics: EACL 2024*, Mar. 2024, pp. 420–436. <https://aclanthology.org/2024.findings-eacl.29.pdf>
- [21] A. Kaplunovich, “Advancing LLM agents for code generation: Observability, orchestration, reliable performance,” in Proc. 3rd Int. Conf. Intelligent Computing, Communication, Networking and Services (ICCNS), Varna, Bulgaria, 2025. <https://ieeexplore.ieee.org/abstract/document/11428688/>
- [22] M.-A. Storey, R. Hoda, A. M. P. Milani, and M. T. Baldassarre, “Guiding principles for mixed methods research in software engineering,” *Empirical Software Engineering*, vol. 30, no. 5, Art. no. 138, 2025. <https://link.springer.com/content/pdf/10.1007/s10664-025-10629-x.pdf>
- [23] A. S. Kang *et al.*, “Enhancing biocompatibility: Heat-treated Al₂O₃- and ZrO₂-reinforced hydroxyapatite coatings on metallic implants,” *The International Journal of Advanced Manufacturing Technology*, vol. 141, no. 12, pp. 6195–6208, 2025. https://www.researchgate.net/profile/Amish-kumar-Patel/publication/398003329_Enhancing_biocompatibility_heat-treated_Al2O3-and_ZrO2-reinforced_hydroxyapatite_coatings_on_metallic_implants/links/69e7b402ef6ff80fdc762f05/Enhancing-biocompatibility-heat-treated-Al2O3-and-ZrO2-reinforced-hydroxyapatite-coatings-on-metallic-implants.pdf
- [24] O. Dhangar, “AI agentic systems with microservices architecture: Bridging software engineering lifecycle gaps in large-scale applications,” in Proc. 2025 IEEE Int. Conf. Recent Advances in Computing and Systems (REACS), 2025. <https://ieeexplore.ieee.org/abstract/document/11413456/>
- [25] C. Lohse, A. Selk, A. Ba, J. Wahl, and M. Ruffini, “Licence to Scale: A microservice simulation environment for benchmarking agentic AI,” in *Workshop on Scaling Environments for Agents*, Dec. 2025. <https://openreview.net/pdf?id=03xThlPUxU>
- [26] S. V. Vuddanti, A. Shah, S. K. Chittiprolu, T. Song, S. Dev, K. Zhu, and M. Chaudhary, “PALADIN: Self-correcting language model agents to cure tool-failure cases,” *arXiv preprint arXiv:2509.25238*, 2025. <https://arxiv.org/pdf/2509.25238>
- [27] B. Li, Y. Jiang, V. Gadepally, and D. Tiwari, “LLM inference serving: Survey of recent advances and opportunities,” in Proc. 2024 IEEE High Performance Extreme Computing Conf. (HPEC), Wakefield, MA, USA, 2024. <https://arxiv.org/pdf/2407.12391>
- [28] L. Zhu, Q. Zhuang, H. Jiang, H. Liang, X. Gao, and W. Wang, “Reliability-aware failure recovery for cloud computing based automatic train supervision systems in urban rail transit using deep reinforcement learning,” *Journal of Cloud Computing*, vol. 12, Art. no. 147, 2023. <https://link.springer.com/content/pdf/10.1186/s13677-023-00502-x.pdf>
- [29] B. Rombaut, S. Masoumzadeh, K. Vasilevski, D. Lin, and A. E. Hassan, “Watson: A cognitive observability framework for the reasoning of LLM-powered agents,” in Proc. 40th IEEE/ACM Int. Conf. Automated Software Engineering (ASE), Nov. 2025, pp. 739–751. <https://arxiv.org/pdf/2411.03455>
- [30] Q. Lu *et al.*, “AgentArcEval: An architecture evaluation method for foundation model based agents,” *Journal of Systems and Software*, Art. no. 112656, 2025. <https://arxiv.org/pdf/2510.21031>
- [31] V. Nagaraj, S. R. Gundla, and P. R. Vennamaneni, “AI-driven secure and cloud-native microservice architectures for high-performance financial and hardware accelerator systems,” in Proc. 2025 Tenth Int. Conf. Science Technology Engineering and Mathematics (ICONSTEM), Nov. 2025, pp. 1–7. <https://ieeexplore.ieee.org/document/11374740>
- [32] T. Ou, W. Guo, A. Gandhi, G. Neubig, and X. Yue, “Agentdiagnose: An open toolkit for diagnosing LLM agent trajectories,” in Proc.

- 2025 Conf. Empirical Methods in Natural Language Processing: System Demonstrations, Nov. 2025, pp. 207–215. <https://aclanthology.org/2025.emnlp-demos.15.pdf>
- [33] C. Bonthu, G. Malik, and M. R. Dhanagari, “AI-driven secure and scalable data architecture with multi-domain MDM and cybersecurity for real-time fintech decision-making,” in *Proc. 2025 Tenth Int. Conf. Science Technology Engineering and Mathematics (ICONSTEM)*, Nov. 2025, pp. 1–7. <https://ieeexplore.ieee.org/document/11374416>
- [34] P. Roshan, T. Prislán, M. Biswas, and A. Lee, “Semantic checkpointing for stateless LLM agents in multi-tenant enterprise systems,” 2025. https://www.researchgate.net/profile/Prakash-Roshan/publication/399433967_Semantic_Checkpointing_for_Stateless_LLM_Agents_Semantic_Checkpointing_for_Stateless_LLM_Agents_in_Multi-Tenant_Enterprise_Systems/links/695ac56b0c98040d482735c2/Semantic-Checkpointing-for-Stateless-LLM-Agents-Semantic-Checkpointing-for-Stateless-LLM-Agents-in-Multi-Tenant-Enterprise-Systems.pdf
- [35] Z. Wu, Q. Zeng, Z. Zhang, Z. Tan, C. Shen, and M. Jiang, “Large language models can self-correct with key condition verification,” in *Proc. 2024 Conf. Empirical Methods in Natural Language Processing (EMNLP)*, Miami, FL, USA, 2024, pp. 12846–12867. <https://aclanthology.org/2024.emnlp-main.714.pdf>
- [36] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen, and Q. Wu, “Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems,” in *Proc. 42nd Int. Conf. Mach. Learn. (ICML)*, vol. 267, 2025, pp. 76583–76599. <https://arxiv.org/pdf/2505.00212?>
- [37] V. Gundu, “Methodological foundations of AI observability for enterprise LLM applications,” *International Journal of Engineering and Computer Science*, vol. 14, no. 10, pp. 27742–27747, 2025.
- [38] S. Rangu, K. S. Chadha, and S. Samala, “AI-powered cloud architecture, agile analytics, and enterprise DevOps-enabled framework for healthcare and insurance ecosystem optimisation,” in *Proc. 2025 Tenth Int. Conf. Science Technology Engineering and Mathematics (ICONSTEM)*, Nov. 2025, pp. 1–7. [\[https://ieeexplore.ieee.org/document/11374414\]](https://ieeexplore.ieee.org/document/11374414)
- [39] I. Bouzenia and M. Pradel, “Understanding software engineering agents: A study of thought-action-result trajectories,” in *Proc. 40th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, Seoul, Republic of Korea, 2025. <https://arxiv.org/pdf/2506.18824>
- [40] P. K. R. Prakashkumar, “Analytical study of Vertex–Oracle cloud integration in enterprise ERP systems for automated tax compliance and efficient financial closings,” *International Journal of Accounting and Management Sciences*, vol. 4, no. 4, 2025. <https://scipubhouse.com/international-journal-of-accounting-and-management-sciences-ijams/content/ijams-vol-4-no-4-october-2025/analytical-study-of-vertex-oracle-cloud-integration-in-enterprise-erp-systems-for-automated-tax-compliance-and-efficient-financial-closings/>
- [41] E reference: [1] H. Lee, S. Oh, J. Kim, J. Shin, and J. Tack, “ReVISE: Learning to Refine at Test-Time via Intrinsic Self-Verification,” in *Proc. 42nd Int. Conf. Machine Learning (ICML)*, vol. 267, 2025. <https://arxiv.org/pdf/2502.14565>
- [42] J. Sun, S. Y. Min, Y. Chang, and Y. Bisk, “Tools fail: Detecting silent errors in faulty tools,” in *Proc. 2024 Conf. Empirical Methods in Natural Language Processing (EMNLP)*, Miami, FL, USA, 2024, pp. 14272–14289. <https://aclanthology.org/2024.emnlp-main.790.pdf>
- [43] N. AI, *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile*. Gaithersburg, MD, USA: NIST Trustworthy and Responsible AI, 2024. https://pathologyinnovationcc.org/s/R8b_1722118549750.pdf

- [44] S. Zhang et al., “No more data silos: Unified microservice failure diagnosis with temporal knowledge graph,” *IEEE Trans. Serv. Comput.*, vol. 17, no. 6, pp. 4013–4026, Nov.–Dec. 2024. https://nkcs.iops.ai/wp-content/uploads/2025/09/No_More_Data_Silos_Unified_Microservice_Failure_Diagnos_With_Temporal_Knowledge_Graph.pdf
- [45] M. W. Ahmad, M. U. Akram, M. M. Mohsan, K. Saghar, R. Ahmad, and W. H. Butt, “Transformer-based sensor failure prediction and classification framework for UAVs,” *Expert Systems with Applications*, vol. 248, Art. no. 123415, 2024. <https://www.sciencedirect.com/science/article/pii/S095741742400280X>