Taming Uncertainty via Automation: Observing, Analyzing, and Optimizing Agentic AI Systems

Dany Moshkovich
IBM Research
Haifa, Israel
mdany@il.ibm.com

Sergey Zeltyn
IBM Research
Haifa, Israel
sergeyz@il.ibm.com

***Abstract*—Large Language Models (LLMs) are increasingly deployed within agentic systems—collections of interacting, LLM-powered agents that execute complex, adaptive workflows using memory, tools, and dynamic planning. While enabling powerful new capabilities, these systems also introduce unique forms of uncertainty stemming from probabilistic reasoning, evolving memory states, and fluid execution paths. Traditional software observability and operations practices fall short in addressing these challenges.**

This paper presents our vision of AgentOps: a comprehensive framework for observing, analyzing, optimizing, and automating operation of agentic AI systems. We identify distinct needs across four key roles—developers, testers, site reliability engineers (SREs), and business users—each of whom engages with the system at different points in its lifecycle. We present the AgentOps Automation Pipeline, a six-stage process encompassing behavior observation, metric collection, issue detection, root cause analysis, optimized recommendations, and runtime automation. Throughout, we emphasize the critical role of automation in managing uncertainty and enabling self-improving AI systems—not by eliminating uncertainty, but by taming it to ensure safe, adaptive, and effective operation.

***Index Terms*—Large Language Models, Multi-Agent Systems, Monitoring, Analytics, Observability, Agentic systems, Performance Optimization, Evaluation**

I. Introduction

The rapid rise of Large Language Models (LLMs) has ushered in a new generation of agentic AI systems—where multiple agents collaborate to execute complex, adaptive workflows. These systems execute tasks often defined in natural language, integrate external tools, retain memory, and coordinate dynamic interactions across agents.

Designed for autonomy, agentic systems move beyond fixed instructions to make context-aware decisions using probabilistic reasoning. Their behavior is shaped by LLMs, classical AI planning, and machine learning, often producing divergent outcomes for identical inputs [1].

This unpredictability extends to execution paths. Agents can decompose tasks into subtasks, dynamically plan their execution, or delegate them to other agents or tools, which may follow alternative workflows based on internal inference.

Memory introduces additional variation, as agents persist and retrieve information using similarity-based methods—commonly backed by vector databases—further amplifying nondeterminism. As they operate, agents accumulate knowledge and develop expertise, making each instance unique and increasingly difficult to update or replace.

The operating environment is equally fluid. Tools may change, vanish, or be introduced at runtime, prompting agents to continuously adapt their usage. In multi-agent settings, coordination emerges from real-time interaction. Like human teams, agents delegate, validate, and reassign roles dynamically—creating evolving collaboration patterns and feedback loops.

This dynamic landscape introduces significant challenges for users—developers, evaluators, SREs, and business stakeholders—across the system lifecycle. Surveys show that only 8% of organizations use dedicated observability platforms [2], [3], limiting automation and scalability, and 60% of users report that current analytics tools do not meet their needs [4].

Addressing this uncertainty demands more than traditional operations. It requires a new discipline: **AgentOps** [5]–[7].

AgentOps provides a framework for observing, analyzing, and optimizing intelligent systems that reason and adapt. It treats agents not as static code, but as stateful, evolving entities that must be monitored, guided, and improved—redefining operations beyond classical ITOps. From instrumentation to feedback loops and self-healing, AgentOps offers the practices needed to operate AI systems safely in enterprise contexts.

The core steps of the AgentOps process include: **Observing Behavior, Calculating Metrics, Detecting Issues, Identifying Root Causes, Generating Optimized Recommendations**, and **Automating Operations**—all supported by a foundation of **automation**.

AgentOps frameworks are too complex to manage manually—especially for users lacking machine learning expertise. Even with the right tools, it is difficult to select and apply them correctly. And as systems autonomously adjust prompts, code, or configurations, validating those changes becomes even harder. Automation lowers this barrier by recommending or executing actions to improve system behavior.

This paper makes three contributions. First, it outlines the distinct challenges agentic systems pose for various user roles. Second, it introduces the AgentOps framework taxonomy and maps its components to these roles. Third, it explores automation as a unifying force across roles, analytics pipelines, and self-improving systems.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00327

II. Related Work: Landscape of Agent Analytics

The agent observability and analytics landscape has split into two categories: GenAI tools [8] like Phoenix [9], LangFuse [10], and LangSmith [11] targeting developers, and observability platforms like Datadog [12] and IBM Instana [13], which are evolving to support agentic systems for SREs.

All of these rely on observability and data collection, reinforcing the need for standardized protocols. **OpenTelemetry (OTel)** [14], a key standard for logs, traces, and metrics, is being extended to support agent-based workflows. **OpenLLMetry** [15] by Traceloop enables observability for frameworks like LangGraph [16], CrewAI [17], and AutoGen [18]. Other notable efforts include **OpenInference** [19] and LangFuse [10]. Still, there are no widely adopted semantic conventions for agentic tracing, and instrumentation for behaviors like planning or reflection remains limited [20].

Beyond observability, analytics tools lack standardized failure taxonomies. While early proposals exist [21], [22], they are not widely adopted. Critical issues—like unintended loops in multi-agent workflows—often go undetected. Graph Neural Networks [23] offer a promising direction for encoding agentic structure, but failure analysis using them is still nascent.

Root cause analysis remains similarly underdeveloped. Few existing tools effectively capture causal relationships between agent decisions, tool behaviors, and observed failures. Prior work on causal reasoning in process mining [24] and causal discovery in agentic systems [25] offers promising foundations for adaptation in this context.

There is also a gap in **recommendation systems**. While problems like latency or hallucination may be flagged, actionable suggestions—such as prompt tuning, agent restructuring, or parameter changes—are rare. **Optimization** support is minimal, with users left to manually adjust trade-offs between cost, latency, and quality.

Finally, automation remains limited, with only initial efforts to design approaches tailored to agentic systems [26]. Most systems require manual intervention, even for recurring issues, and workflows are rarely refined or adapted automatically.

III. Roles and Responsibilities: How Agents Change the Game

Agentic systems disrupt boundaries between development, testing, deployment, and operations. Their dynamic, unpredictable nature requires all roles to adapt. We focus on four core roles—developers, testers, operators, and business users—who span the full system lifecycle and, like security, compliance, and product management roles, face distinct challenges as they adapt to agentic reality.

A. Developers

In traditional software development, developers design and implement code, then debug it using breakpoints and by testing a few fixed execution paths. In agentic development, however, this approach is no longer sufficient—running the system once, even with the same initial conditions, may yield different outcomes due to inherent stochasticity and dynamic reasoning.

Developers must tune numerous LLM parameters (e.g., temperature, context window) and configure integrated tools and APIs—often under tight coupling and evolving constraints. Designing and orchestrating dynamic behavior in agentic systems under these conditions is particularly challenging.

Understanding system behavior requires instrumentation to capture structured trace data. In parallel, prompt engineering introduces a vast, unstructured design space that demands iterative experimentation. Dynamic code generation further complicates this, as execution flows can shift at runtime, requiring greater adaptability.

B. Testers

Testers are responsible for comprehensive validation, traditionally measured by **code** and **requirements coverage**. In agentic systems, however, non-determinism complicates this task—executing all code paths and requirements does not guarantee coverage of all relevant behaviors.

Outcomes are often non-binary, with success on a **continuous spectrum** and acceptable thresholds varying by application criticality. Testing must shift from final outputs to **intermediate states** and **decision points**—such as tool choices or routing logic—to fully understand behavior.

Finally, evaluation should not end at deployment. As agentic systems evolve with usage, context, and learning, post-deployment testing and monitoring are essential for sustained reliability, trustworthiness, and performance.

C. Site Reliability Engineers

A Site Reliability Engineer (SRE) bridges software engineering and operations, focusing on building systems that are both reliable and efficient. Traditionally, SREs monitor performance metrics and respond to failures or outages.

In agentic systems, the focus shifts to **proactive trend analysis**. Like physicians tracking vital signs, SREs must monitor numeric and semantic indicators—such as latency, cost, tool usage, and human input—to detect early signs of systemic issues. Waiting for a complete system failure is akin to diagnosing a heart attack after it occurs—by that point, intervention may be too late.

When a drift, anomaly, or inefficiency is detected, SREs must perform **root cause analysis** and identify actionable mitigation strategies. Some issues can be addressed automatically—through dynamic reconfiguration, scaling, prompt changes, or tool replacement—while others require developer intervention. This closes the loop between observability and ongoing system improvement.

D. Business Users

Unlike SREs, business users focus on **business-centric metrics** such as revenue, cost, ROI, and customer satisfaction—especially the trade-off between cost, latency, and quality. These metrics must be monitored, linked to business outcomes, and analyzed for anomalies and root causes.

Beyond issue detection, business users explore new business opportunities through what-if analyses and A/B testing, to support strategic decision-making.

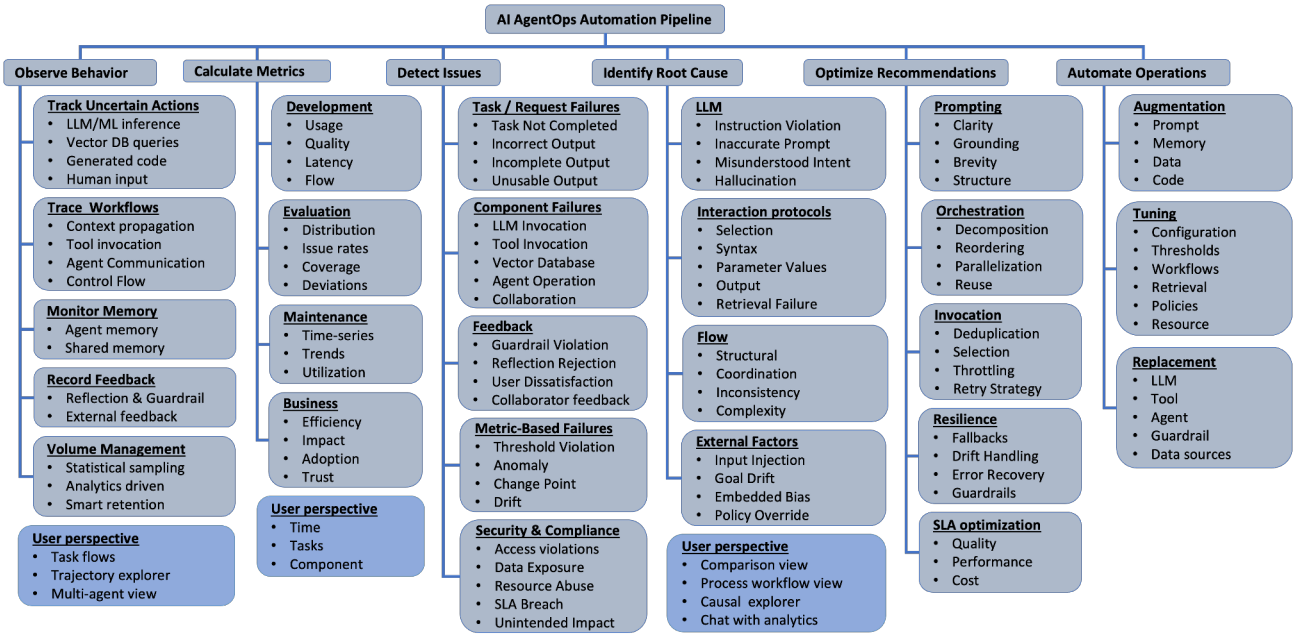

Fig. 1. AI AgentOps Automation Pipeline

IV. AI AgentOps Automation Pipeline

To address emerging challenges faced by developers, testers, operators, and business users, we introduce the AI AgentOps Automation Pipeline—a six-stage process spanning from raw behavior capture to self-healing mechanisms. Each stage targets a core operational need and supports both automated and user-driven methods. The accompanying taxonomy and user perspectives (Fig. 1) illustrate these concepts; below, we explain their rationale and interconnections.

A. Observe Behavior

Observing an agentic system goes beyond classical tracing—it requires capturing how decisions and execution flows emerge dynamically, including those driven by code generated by the agent at runtime. This involves automatic instrumentation to trace probabilistic behaviors such as LLM inference, tool usage, vector database queries, and human input.

Understanding workflows requires tracking context propagation, tool invocations, and inter-agent communication to reconstruct decision paths—especially when control flow is composed on the fly.

Observability must also capture feedback loops, including internal reflection, guardrails, and user input, that influence behavior. To manage trace volume, automatic volume management techniques such as statistical sampling, anomaly-triggered escalation, and smart retention help maintain visibility while limiting overhead.

Finally, observability should aid interpretation through task flow visualizations, execution trajectories, and multi-agent views that reveal how behavior evolves at runtime.

B. Collect Metrics

While observability surfaces raw signals, metrics automatically transform them into structured insights. During development, we track usage metrics like tool call frequency and memory access rate, quality indicators such as task success and output completeness, as well as latency and flow characteristics including task volume, branching complexity, and reasoning depth.

Evaluation emphasizes behavior distribution across runs, failure and degradation rates, functional and edge case coverage, and deviations from golden traces or ground truth.

In maintenance, time-series metrics reveal regressions, drift trends, and system utilization—supporting proactive scaling and adaptation.

For business stakeholders, metrics highlight efficiency through cost and latency, impact via ROI and productivity, adoption through usage penetration, and trust via positive feedback and compliance adherence.

User-scoped metrics enable slicing by time intervals, task categories, and system components, aligning insights with specific user needs.

C. Detect Issues

The automation layer analyzes data and metrics to detect issues—including both outright failures and subtle degradations. It categorizes them by type and scope, assigns severity, correlates related events, and triggers smart alerts.

Task failures cover incomplete tasks, incorrect or partial outputs, and syntactically valid but unusable responses.

Component issues may not block task completion but reveal deeper, often systemic problems—such as LLM timeouts,

low-confidence outputs, tool or vector DB errors, and agent miscoordination.

Failures in handling feedback—such as violations of guardrails, rejection of reflective feedback, user dissatisfaction, or collaboration failures—are another critical class. These break the adaptive loop that makes agents responsive and intelligent.

Metric-based failures manifest when monitored values breach pre-defined thresholds, take on anomalous values, hit change points indicating discrete shifts in behavior, or exhibit pattern shifts that signal evolving system dynamics.

Security and compliance issues cannot be ignored. These include breaches of access control policies, exposure of sensitive data, misuse of resources, SLA violations, and unintended side effects that compromise operational or ethical integrity.

D. Identify Root Cause

Root cause analysis (RCA) automatically bridges the gap between symptoms and solutions. A common root cause category involves LLM-related issues—such as instruction violations, ambiguous or inaccurate prompts, misunderstood intent, or hallucinations. Even subtle prompt flaws can lead to significant behavioral differences.

Problems in interaction protocols—both between agents and in tool invocation—often stem from improper tool or parameter selection, syntax errors, and invalid configurations. These missteps can lead to breakdowns in execution, particularly when the necessary context is missing or retrieval fails.

Flow and coordination failures reflect deeper behavioral problems: misaligned task decomposition, inconsistent coordination between agents, structural gaps, or overly complex multi-step goals.

External factors also play a role. Input injection can derail planning, goal drift shifts system behavior away from intended outcomes, embedded bias may skew decisions, and policy overrides may suppress needed checks.

To support investigation, users benefit from comparison views between healthy and failing traces, end-to-end process workflow reconstructions, causal path explorers that surface potential dependencies, and analytics chat interfaces that allow direct queries like ”Why did this fail?”

E. Optimize Recommendations

Once root causes are known, optimization focuses on making targeted improvements. Prompting issues are often addressed first—by clarifying ambiguous phrasing, grounding responses in relevant context, tightening verbosity, or applying clearer structural patterns.

Workflow-level enhancements involve refining task decomposition, reordering steps for efficiency, enabling parallelization, and reusing results when appropriate. On the invocation side, removing redundant calls, selecting better tools, applying throttling, and using smarter retry logic can stabilize execution.

To improve resilience, systems should incorporate fallback options, detect and manage behavioral drift, recover gracefully from errors, and enforce guardrails. All these adjustments must also consider SLA tradeoffs, balancing quality, performance, and cost in line with user priorities.

F. Automate Operations

Automation closes the loop by enacting improvements automatically when confidence is high. This includes augmentation of prompts, runtime data, and tool instructions to adjust agent behavior on the fly, as well as tuning of configurations, thresholds, retrieval logic, or timeouts to maintain optimal performance.

When deeper issues persist, AgentOps may switch LLMs, replace tools, modify workflows, update guardrails, or reset faulty components—all without requiring code changes or redeployment.

For example, AgentOps **observes behavior** by noticing that an SRE Agent [27] inconsistently uses a new diagnostic tool, based on traces of its *workflow execution* and *tool invocations*. AgentOps **collects metrics**, monitoring *issue rates* and identifying frequent *incomplete outputs* for a specific *task*. Next, AgentOps **detects issues** by spotting a *drift* in this failure pattern, indicating rising *task failures*. AgentOps **identifies the root cause** as *inaccurate prompt* instructions and *tool misuse*. It then **optimizes recommendations** by suggesting a clearer *prompt*. Finally, AgentOps **automates operation** by *augmenting* the Agent's *prompt* and revalidating the fix through continuous monitoring—automatically optimizing behavior without changing code. Like a manager guiding an employee based on performance feedback, AgentOps enables agentic systems to self-correct and improve in real time.

V. Discussion: Taming, not Eliminating, Uncertainty

Uncertainty is intrinsic to intelligence. Just as we accept ambiguity in human reasoning, we must also recognize it in intelligent software systems. But recognition does not imply surrender. While agentic systems will inevitably exhibit behavioral uncertainty, the goal is to tame it—minimizing the frequency and severity of undesirable or strongly suboptimal outcomes.

Promising directions for taming uncertainty through automation include:

- **Standardization.** Our taxonomy provides a foundation for AgentOps instrumentation, evaluation, and automation, aligned with emerging standards such as OpenTelemetry, MCP [28], and the UIM Protocol [29] that promote interoperability across agentic systems.
- **Graph-based, alphanumeric analytics.** Agentic systems produce structured, graph-shaped data with semantic richness. New methods must encode and apply this data for issue detection and root cause analysis.
- **Self-healing and adaptive execution.** Automated mechanisms should enable systems to respond to problems in real time—rerouting tasks, adjusting LLM parameters, or altering execution plans—reducing the impact of suboptimal behavior without always requiring human intervention.

REFERENCES

[1] B. Atil, S. Aykent, A. Chittams, L. Fu, R. J. Passonneau, E. Radcliffe, G. R. Rajagopal, A. Sloan, T. Tudrej, F. Ture *et al.*, "Non-determinism of "deterministic" LLM settings," *arXiv preprint arXiv:2408.04667*, 2024.

[2] Skeen, Julie. (2025) Mastering AI data observability: Top trends and best practices for data leaders. Precisely. [Online]. Available: https://www.precisely.com/blog/data-quality/mastering-ai-data-observability-top-trends-and-best-practices-for-data-leaders

[3] Precisely. (2025) BARC research study: Observability for AI innovation. Precisely. [Online]. Available: https://www.precisely.com/resource-center/analystreports/barc-research-study-observability-for-ai-innovation

[4] D. Moshkovich, H. Mulian, S. Zeltyn, N. Eder, I. Skarbovsky, and R. Abitbol, "Beyond black-box benchmarking: Observability, analytics, and optimization of agentic systems," *arXiv preprint arXiv:2503.06745*, 2025.

[5] AgentOps.ai. (2025) Trace, debug, & deploy reliable AI agents. AgentOps.ai. [Online]. Available: https://www.agentops.ai/

[6] Murphy, Mike. (2025) How to know if your AI agents are working as intended. IBM Research. [Online]. Available: https://research.ibm.com/blog/ibm-agentops-ai-agents-observability

[7] L. Dong, Q. Lu, and L. Zhu, "A taxonomy of AgentOps for enabling observability of foundation model based agents," *arXiv preprint arXiv:2411.05285*, 2024.

[8] E. Jose and P. Prabhakaran, "Harnessing large language models (LLMs) optimizing performance, monitoring, and compliance," *Authorea Preprints*, 2024.

[9] Phoenix. (2025) Phoenix. Arize AI. [Online]. Available: https://phoenix.arize.com/

[10] LangFuse. (2025) Langfuse. LangFuse. [Online]. Available: https://langfuse.com/

[11] LangSmith. (2025) Langsmith. LangChain. [Online]. Available: https://smith.langchain.com/

[12] Datadog. (2025) Datadog. Datadog. [Online]. Available: https://www.datadoghq.com/

[13] IBM Instana. (2025) IBM Instana. IBM. [Online]. Available: https://www.ibm.com/products/instana

[14] D. G. Blanco, *Practical OpenTelemetry: Adopting Open Observability Standards Across Your Organization*. Heidelberg, Germany: Springer, 2023.

[15] OpenLLMetry. (2025) Openllmetry. Traceloop. [Online]. Available: https://www.traceloop.com/openllmetry

[16] LangGraph. (2025) Langgraph. LangChain. [Online]. Available: https://www.langchain.com/langgraph

[17] CrewAI. (2025) CrewAI. CrewAI. [Online]. Available: https://www.crewai.com/

[18] AutoGen. (2025) Autogen. Microsoft. [Online]. Available: https://microsoft.github.io/autogen/stable/

[19] OpenInference. (2025) OpenInference. Arize AI. [Online]. Available: https://arize.com/docs/ax/learn/tracing-concepts/what-is-openinference

[20] D. Moshkovich. (2025) Semantic conventions for generative AI agentic systems. [Online]. Available: https://github.com/open-telemetry/semantic-conventions/issues/2664

[21] D. Deshpande, V. Gangal, H. Mehta, J. Krishnan, A. Kannappan, and R. Qia, "Trail: Trace reasoning and agentic issue localization," *arXiv preprint arXiv:2505.08638*, 2025.

[22] M. Z. Pan, M. Cemri, L. A. Agrawal, S. Yang, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, K. Ramchandran, D. Klein *et al.*, "Why do multiagent systems fail?" in *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025.

[23] A. Niro and M. Werner, "Detecting anomalous events in object-centric business processes via graph neural networks," in *International Conference on Process Mining*. Springer, 2023, pp. 179–190.

[24] F. Fournier, L. Limonad, I. Skarbovsky, and Y. David, "The why in business processes: Discovery of causal execution dependencies," *KI-Künstliche Intelligenz*, pp. 1–23, 2025.

[25] F. Fournier, L. Limonad, and Y. David, "Agentic AI process observability: Discovering behavioral variability," *arXiv preprint arXiv:2505.20127*, 2025.

[26] B. P. Sanwouo, P. Temple, and C. Quinton, "Breaking the loop: AWARE is the new MAPE-K," in *FSE'25-International Conference on the Foundations of Software Engineering*, 2025.

[27] S. Jha, R. Arora, Y. Watanabe, T. Yanagawa, Y. Chen, J. Clark, B. Bhavya, M. Verma, H. Kumar, H. Kitahara *et al.*, "Itbench: Evaluating AI agents across diverse real-world it automation tasks," *arXiv preprint arXiv:2502.05352*, 2025.

[28] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (MCP): Landscape, security threats, and future research directions," *arXiv preprint arXiv:2503.23278*, 2025.

[29] Synapti.ai. (2025) Unified intent mediator protocol. Synapti.ai. [Online]. Available: https://www.uimprotocol.com/