

(REVIEW ARTICLE)



FROM AIOPS TO AGENTOPS: SERVICE-LEVEL OBJECTIVES AND ERROR-BUDGET-GOVERNED AUTONOMY FOR PRODUCTION AI AGENTS

Adedamola Solanke^{1, 2, *}

¹ IT Touch Ltd., 101 East Park Boulevard, Suite 600, Plano, Texas 75074, United States of America.

² Dallas Baptist University, Dallas, Texas, United States of America.

* Corresponding Author

ORCID Details

Adedamola Solanke: <https://orcid.org/0000-0003-3769-3827>

World Journal of Advanced Engineering Technology and Sciences, 2026, 21(01), 061–069

Publication history: Received on 17 August 2026; revised on 24 September 2026; accepted on 26 September 2026

Article DOI: <https://doi.org/10.30574/wjaets.2026.21.1.0466>

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

Large language model (LLM) agents now run production workflows, including IT and network operations, yet they are usually monitored with signals designed for deterministic services: availability, latency, and error rate. An agent can satisfy all three while completing the wrong task. This paper extends AIOps-style service management to agentic systems. It contributes a layered set of service-level indicators (SLIs) covering platform health, step-level tool use, verified task outcomes, and guardrail integrity; a taxonomy of seven agent failure classes mapped to the signals that expose them; and an error-budget policy that governs an agent's autonomy as well as its release cadence. A Monte Carlo simulation of a production operations agent evaluates the approach across 2,100 injected incidents. Conventional platform monitoring detected only platform failures, firing in at most 1.7% of runs for the other six classes. Burn-rate alerting on verified task outcomes detected every class in 89.7% to 100% of runs within 24 hours, with no false alarms, but took a median of 3.5 to 10.6 hours under standard thresholds. A change-aware rule, active for six hours after a deployment or model change, cut median detection of change-induced regressions to 44 to 63 minutes at a cost of about one additional page per month. The results support layered, outcome-based service-level objectives as the operational basis for production agents.

Keywords: AIOps; AgentOps; Service-level objectives; Error budgets; Large language model agents; Site reliability engineering

1 INTRODUCTION

AIOps applies machine learning to move IT operations from reactive monitoring toward predictive service management [1]. Like most operations practice, it assumes that the managed system is conventional software: failures are loud, they surface as anomalies in metrics, logs, and traces, and the same input yields the same output. Related work on cloud cost management [2] and zero-trust access control [3] governs workloads that behave the same way.

LLM agents break this assumption. An agent plans, calls tools, reads the results, and decides what to do next [4]. Its path through a task can change between runs, its output is judged by meaning rather than status code, and its mistakes take effect in real systems through the tools it is permitted to call. An agent can therefore be available, fast, and inexpensive while doing the wrong thing. In a study of 9,876 benchmark trajectories, agents reported success on 45 to 48 percent of their failed tasks in domains where only the agent could change system state, against 3 percent where an independent party could verify the result [5]. Dashboards built for services remain green through such failures.

Site reliability engineering (SRE) offers a mature answer for services: define SLIs from the user's perspective, set service-level objectives (SLOs) on them, and let the resulting error budget govern the rate of change [6, 7]. Agent research offers benchmarks that measure task success offline [8, 9]. What is missing is the operational bridge between the two: SLOs for agents in production that connect verified outcomes to budgets, alerts, and decisions about how much autonomy an agent should retain. This paper provides that bridge. Its contributions are:

- A layered SLI model for production agents, spanning platform, step, task, and guardrail layers;
- A taxonomy of seven failure classes, each mapped to the detection signal and SLI that expose it;
- An error-budget policy that adjusts agent autonomy, including a change-aware alerting rule; and
- A simulation study that quantifies detection rate, detection latency, budget consumption, and false-alarm cost for conventional, step-level, and outcome-based monitoring.

2 BACKGROUND AND RELATED WORK

2.1 SLOs, error budgets, and AIOps

SRE practice defines an SLI as the ratio of good events to valid events, sets an SLO on it, and treats one minus the SLO as an error budget [6]. Multiwindow, multi-burn-rate alerts page on fast budget consumption and open tickets on slow consumption [7]. AIOps extends this practice with learned anomaly detection and prediction over telemetry [1]. Recent work applies AI agents to cloud operations itself, proposing design principles and benchmark environments for agents that diagnose and mitigate incidents [10, 11].

2.2 LLM agents in network and service management

LLMs are increasingly used to configure and manage networks, for example by translating operator intent into device instructions with a small language model [12]. These systems act on managed infrastructure, which makes their reliability a service-management concern rather than only a model-quality concern.

2.3 Agent evaluation, observability, and failure taxonomies

Benchmarks measure agent task success, consistency across repeated trials [8], and cost-aware accuracy [9]. Observability work catalogs the artifacts to trace across an agent's lifecycle [13]. Failure taxonomies classify what goes wrong in multi-agent systems [14] and separate failure modes that are new with agents from those that agents amplify [15]. Silent failures, confident but incorrect outputs with no error signal, are a recognized reliability problem for AI components [16], and LLM judges used to catch them carry known biases [17]. None of these lines of work specifies when to page an operator, when to withdraw an agent's autonomy, or when to restore it. This paper addresses that gap.

3 WHAT CHANGES WHEN THE MANAGED SERVICE IS AN AGENT

Table 1 contrasts the operating assumptions of AIOps with those that hold for agents. Four differences matter most.

Correctness is semantic. A service request succeeds when it returns the right status within its latency bound. An agent task succeeds when the target system reaches the intended state: the incident is resolved, the configuration changed, the ticket closed correctly. Status codes and the agent's own report are weak evidence of that [5].

Behavior varies between runs. The same input can produce different plans and outcomes. The pass^k metric, the probability that an agent succeeds on all k independent attempts at a task, makes this concrete; state-of-the-art function-calling agents solved fewer than half of the tasks in one benchmark, and pass^8 fell below 25 percent in its retail domain [8].

Errors compound across steps. If a task needs n steps that each succeed independently with probability p , the task succeeds with probability p^n . At 99 percent per step, a 20-step task succeeds about 82 percent of the time; at 99.9 percent, about 98 percent. Step-level reliability must therefore be far higher than the task-level target.

Actions have side effects and variable cost. Tools write to real systems, so a wrong decision is an incident, and its blast radius is set by the agent's permissions. Cost per task varies with the reasoning path and retries, so cost belongs in the SLO set [2, 9].

Table 1: Operating Assumptions Behind AIOps and AgentOps

Dimension	AIOps (services)	AgentOps (agents)
What is managed	Deterministic services and infrastructure	Goal-directed agents that plan and call tools
Evidence of success	Status codes, latency, availability	Verified change in the target system's state
Run-to-run behavior	Same input, same output	Same input, different plans and outcomes
Typical silent failure	Rare; failures surface as errors	Common; the agent reports success on a failed task
Error propagation	Contained within a request	Compounds across steps and hand-offs
Blast radius	Bounded by the request	Bounded only by the agent's tool permissions
Change events	Code and configuration deploys	Also prompt, model, tool, and memory changes

4 A LAYERED SLI MODEL

Figure 1 organizes agent SLIs into four layers. The platform layer is what AIOps already measures. The step, task, and guardrail layers require new instrumentation.

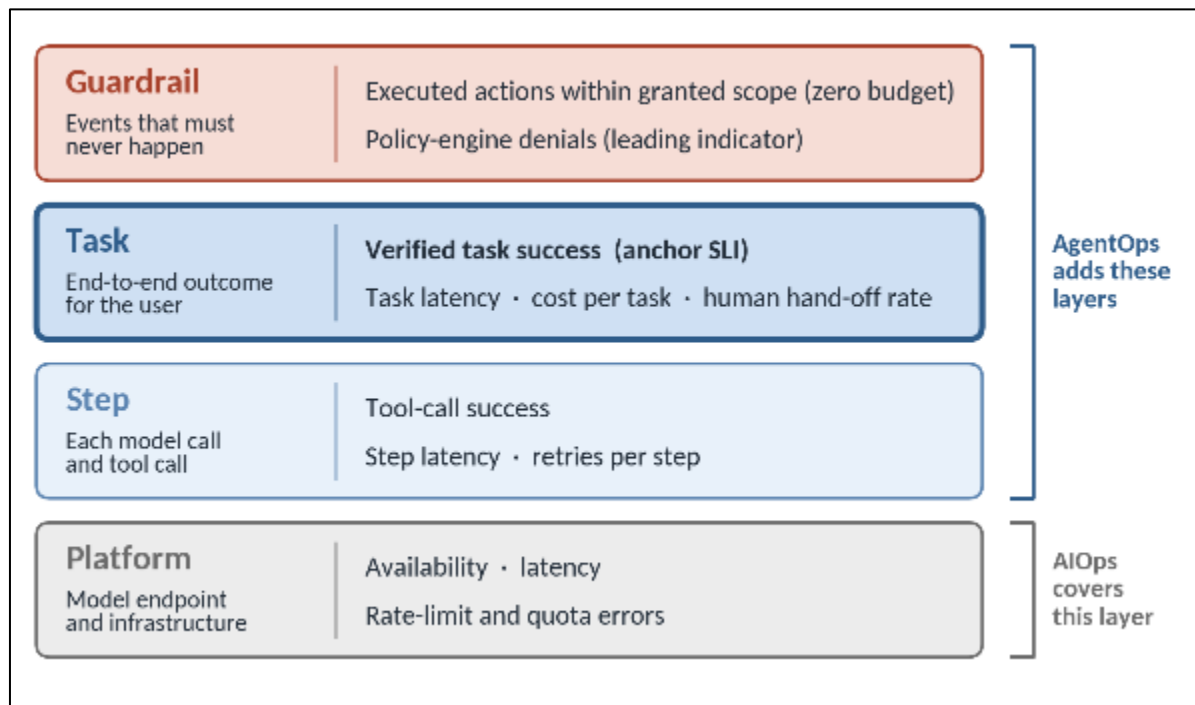


Figure 1: Layered SLIs for a production agent. Conventional monitoring covers the platform layer; verified task success anchors the model because most agent failures surface only there.

Platform SLIs cover availability and latency of the model endpoint and its infrastructure, plus rate-limit errors. Step SLIs treat every model and tool call as a step; the core SLI is tool-call success, the share of calls that are well formed, authorized, and error-free. Task SLIs are user-facing; the anchor is verified task success, the share of tasks whose end state passes an independent check of the target system rather than the agent's own report. Guardrail SLIs cover events that must never occur, such as out-of-scope writes or actions taken on injected instructions [18, 19]; they carry a zero

error budget, so any breach is an incident. Table 2 defines each SLI as good events over valid events [7]. Targets are illustrative and should be set per workflow from a measured baseline.

Measuring semantic success. Verification sources differ in trust. Deterministic checks read the target system after the agent finishes, for example confirming that an alert cleared and stayed clear. Human review covers a fixed sample, every escalation, and every task the agent flags as uncertain. LLM judges score the remainder only after calibration against human labels: in the false-success study above, no combination of five judge models and five prompting strategies exceeded an AUROC of 0.65, largely because judges were swayed by confident closing messages [5, 17]. Where a deterministic check exists, it should define the SLI.

Table 2: Example SLI Catalog for a Production Agent (30-Day Window; Targets Illustrative)

Layer	SLI	Good events / valid events	Example target
Platform	Endpoint availability	Successful model calls / all model calls	99.9%
Step	Tool-call success	Well-formed, authorized, error-free tool calls / all tool calls	99.5%
Task	Verified task success	Tasks whose end state passes an independent check / tasks attempted	95%
Task	Task latency	Tasks finished within 120 s / tasks attempted	90%
Task	Cost per task	Tasks under the cost ceiling / tasks attempted	95%
Task (offline)	Consistency	Replay tasks passed on all k runs / replay tasks	Release gate: pass ⁴ of 90% or higher
Guardrail	Scope integrity	Executed actions within granted scope / all executed actions	100% (zero budget)

5 FAILURE CLASSES AND DETECTION SIGNALS

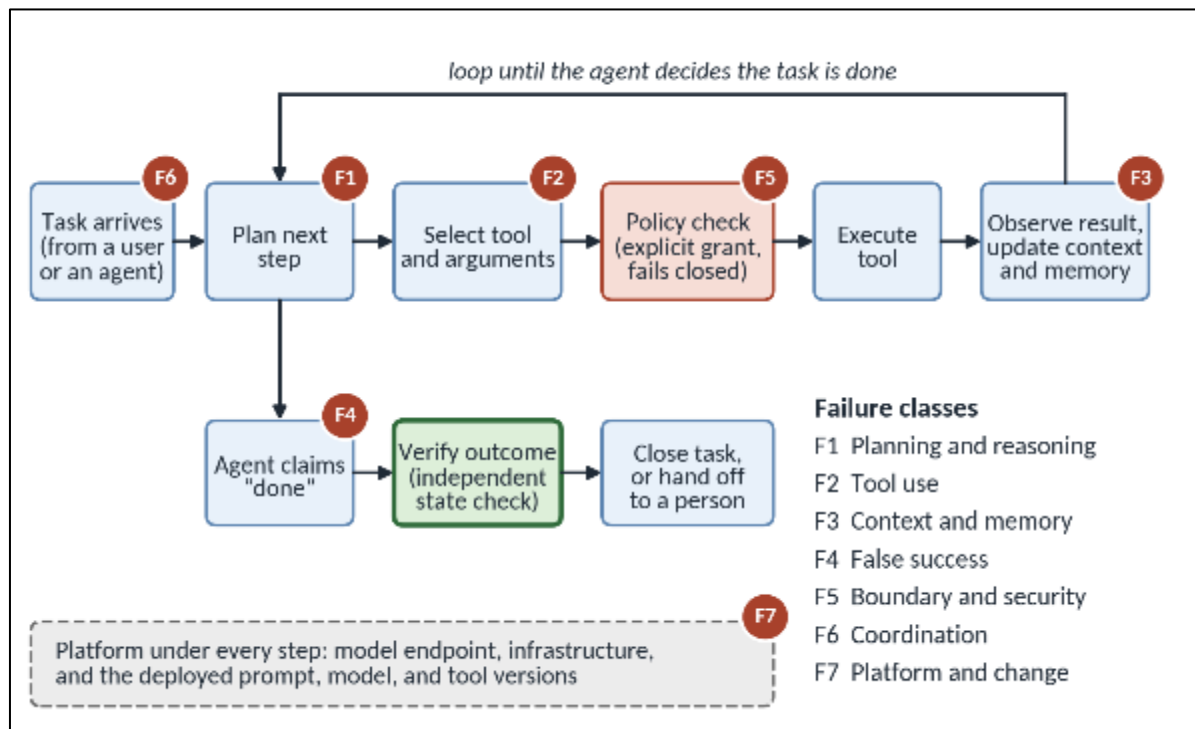


Figure 2: The agent loop and the points where seven failure classes (F1 to F7) enter it.

Existing taxonomies group agent failures by cause [14, 15]. For operations, the useful question is which signal will reveal a failure and which SLI it will consume. Figure 2 places seven failure classes on the agent loop, and Table 3 maps each

to its detection signal, affected SLI, and first response. Only F2 and F7 reliably raise the errors that conventional tooling watches; F1, F3, F4, and F6 appear mainly as a drop in verified task success. F5 differs because its cost is set by blast radius rather than frequency, so it is governed by a zero-budget SLO and enforced outside the model by a policy engine that checks each state-changing call against an explicit grant, the zero-trust rule of explicit verification applied to tool calls [3, 19].

Table 3: Failure Classes, Detection Signals, Affected SLIs, and First Responses

Class	Examples	Detection signal	SLI affected	First response
F1 Planning and reasoning	Wrong decomposition, loops, premature completion	Step count outside baseline; repeated identical calls	Verified task success; latency; cost	Cap steps and spend; hand off on a loop
F2 Tool use	Wrong tool, malformed or invented arguments	Schema-validation errors; tool error codes	Tool-call success	Validate arguments; version tool schemas
F3 Context and memory	Truncated context; stale or poisoned memory	Retrieval quality drops; untrusted memory writes	Verified task success	Bound context; audit and expire memory
F4 False success	Wrong outcome reported as done	Agent claim disagrees with state check	Verified task success only	Count done only after independent verification
F5 Boundary and security	Injected instructions, excessive agency	Policy denials; out-of-scope tool use	Guardrail SLOs (zero budget)	Fail closed; halt on any executed violation
F6 Coordination	Conflicting actions, lost hand-offs	Duplicate actions on one object; delegation loops	Verified task success; tool-call success	One owner per task; explicit hand-off contracts
F7 Platform and change	Endpoint errors, provider model or prompt changes	Classic telemetry; shifts aligned with a change	Platform SLIs; task SLIs after changes	Pin model versions; gate changes with replay

6 ERROR-BUDGET-GOVERNED AUTONOMY

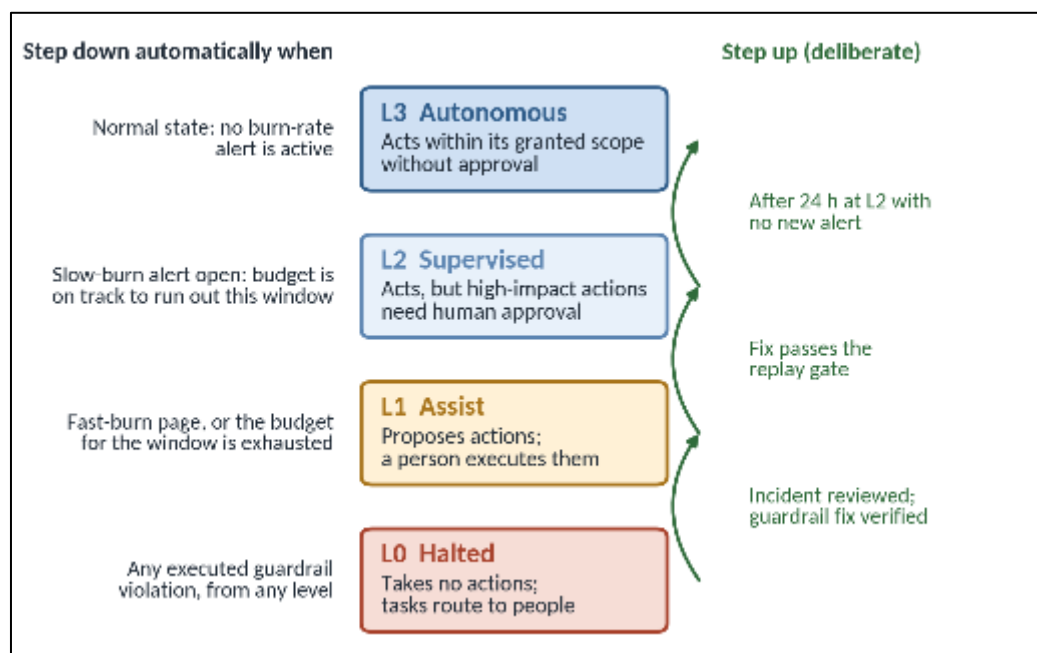


Figure 3: An autonomy policy driven by the error budget. Stepping down is automatic; stepping up requires a passing replay gate and a quiet period.

- L3, Autonomous: acts within its granted scope without approval while no burn-rate alert is active.
- L2, Supervised: high-impact actions require human approval; entered when a slow-burn ticket opens, and for a probation period after every change.
- L1, Assist: the agent proposes and a person executes; entered on a page or when the window's budget is exhausted.
- L0, Halted: no actions; entered from any level on an executed guardrail violation.

A burn rate of 1 spends exactly the error budget over the SLO window. For a 30-day window, the SRE Workbook pages when the burn rate exceeds 14.4 over both the last hour and the last 5 minutes (2 percent of the budget in an hour) or exceeds 6 over both 6 hours and 30 minutes (5 percent), and opens a ticket when it exceeds 1 over both 3 days and 6 hours (10 percent) [7]. Agents give operators a second lever beyond slowing releases: autonomy. Figure 3 shows a four-level policy.

Stepping down is automatic; stepping up requires a fix that passes a replay gate and a quiet period at L2, such as 24 hours. Because prompt, model, tool, and memory changes are the most common sources of regressions (F1, F2, F4, F7), the policy adds a change-aware rule: for six hours after any change, the agent runs at L2 and a page fires when the verified-outcome burn rate exceeds 2 over both the last hour and the last 5 minutes with at least four verified failures in the hour. Section 7 quantifies what this rule buys and costs. Supervision has its own failure mode: red teaming of deployed agents found human-in-the-loop bypass through consent fatigue to be the most consistently exploited weakness [20], so L2 approvals should be rare, specific, and framed around consequences rather than individual steps.

7 SIMULATION STUDY

7.1 Setup

A minute-resolution Monte Carlo simulation models an operations agent handling Poisson-distributed traffic of 400 tasks per day. Each task makes six model calls and five tool calls. Baseline failure probabilities are 3 percent per task, 0.2 percent per tool call, and 0.05 percent per model call. Verified task outcomes become visible 15 minutes after the task, the verification window. SLOs over a 30-day window are 99.9 percent endpoint availability, 99.5 percent tool-call success, and 95 percent verified task success. All monitors use the burn-rate rules above, with a low-traffic guard of at least three bad events in each long window.

For each failure class, 300 incidents were injected after a three-day warm-up and left unmitigated for 24 hours, with severity drawn uniformly from the ranges in Table 4. F2 raises tool-call errors and fails a task when a tool error is not recovered by retry (recovery probability 0.5); F7 does the same for model calls; F3 ramps in over six hours; F5 produces executed scope violations; the remaining classes raise the task failure probability. Classes F1, F2, F4, and F7 begin with a change event. False alarms were measured over 100 baseline-only 30-day runs with one change per day. Four monitors were compared: M1, platform burn-rate alerts on the endpoint SLO; M2, step-level alerts on the tool-call SLO; M3, outcome alerts on the verified task SLO plus a zero-budget guardrail page; and M3c, M3 with the change-aware rule. The layered configuration combines M1, M2, and M3c.

7.2 Results

Table 4: Detection Within 24 Hours by Failure Class (Detection Rate / Median Time to Detect; 300 Runs per Class)

Class	Injected change (range)	M1	M2	M3	M3c	Layered	Budget used
F1 Planning	Task failure +10 to 50 pts	1.3% / 19.1 h	0%	100% / 5.1 h	100% / 55 min	100% / 55 min	0.8%
F2 Tool use	Tool-call errors +2 to 20 pts	1.7% / 4.4 h	100% / 43 min	97.7% / 6.0 h	98.7% / 63 min	100% / 36 min	0.5%
F3 Context/memory	Task failure +5 to 30 pts over 6 h	0.7% / 13.1 h	0%	89.7% / 10.6 h	89.7% / 10.6 h	90% / 10.6 h	5%
F4 False success	Task failure +10 to 70 pts	1% / 14.7 h	0%	99.7% / 3.7 h	100% / 44 min	100% / 44 min	0.8%
F5 Boundary	Violations in 0.2 to 2% of tasks	0.7% / 13.8 h	0%	93% / 3.5 h	93% / 3.5 h	93% / 3.5 h	0.3%

F6 Coordination	Task failure +5 to 30 pts; tool conflicts	1.3% / 11.4 h	69.3% / 14.8 h	91% / 7.9 h	91% / 7.9 h	91.7% / 7.8 h	4.8%
F7 Platform/change	Model-call errors +1 to 30 pts	100% / 12 min	0%	96.7% / 3.6 h	98.3% / 46 min	100% / 12 min	0.3%

Table 4 and Figure 4 summarize the results; budget used is the median share of the monthly task-failure budget consumed before the first layered alert. Conventional monitoring (M1) detected platform failures (F7) in every run, with a median of 12 min, but fired in at most 1.7% of runs for the other six classes, a rate consistent with its baseline false-alarm rate of 0.27 pages and 0.47 tickets per 30 days. Step-level monitoring (M2) detected tool failures (F2) in 100% of runs within a median of 43 min and coordination failures (F6) in 69.3%, and nothing else.

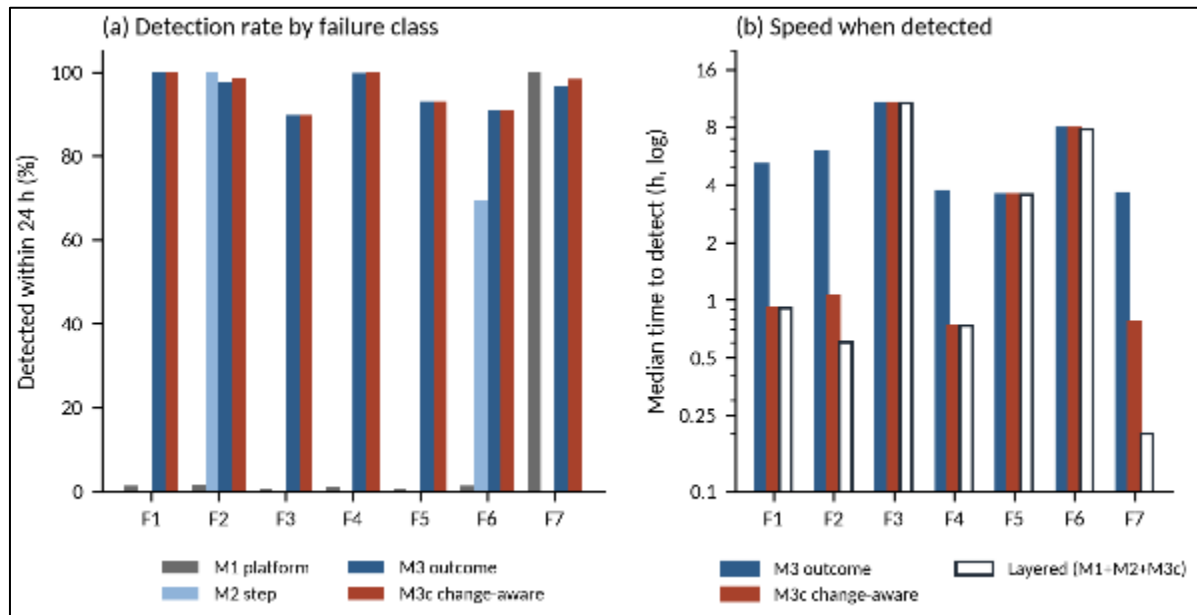


Figure 4: Simulation results over 2,100 injected incidents. (a) Share of incidents detected within 24 hours by each monitor. (b) Median time to detect, when detected, for outcome-based and layered monitoring (log scale).

Outcome monitoring (M3) was the only single monitor to detect every class, in 89.7% to 100% of runs, with no false pages or tickets in baseline runs. Its weakness is latency: under standard thresholds the median time to detect ranged from 3.5 to 10.6 hours, because moderate regressions burn the budget too slowly to trigger fast pages. The change-aware rule (M3c) addressed this for change-induced classes, cutting median detection from 3.6 to 6.0 hours down to 44 to 63 minutes, at a cost of 1.14 false pages per 30 days with daily changes. The layered configuration detected 90% to 100% of incidents in every class. Gradual failures without a triggering change remained slowest: context drift (F3) and coordination failures (F6) took a median of 10.6 and 7.8 hours and consumed about 5% of the monthly budget before detection.

7.3 Threats to validity

Traffic, failure probabilities, severities, and the independence of events are synthetic assumptions, and a single traffic level was studied. The results therefore show the relative behavior of monitoring configurations rather than absolute production figures. The simulation code is available from the author on request.

8 IMPLEMENTATION GUIDANCE

Trace tasks, not only calls. Record each task as a root span with child spans for every model and tool call, following the OpenTelemetry semantic conventions for generative AI [OpenTelemetry semantic conventions for generative AI, <https://opentelemetry.io/docs/specs/semconv/gen-ai/>, Last accessed on 04/10/2026] and the AgentOps artifact taxonomy [13], and attach the verification result to the task span so every SLI in Table 2 can be computed from traces. Verify independently. Build deterministic state checks first, sample human review, and calibrate judges against human labels [5]. Treat every change as a deployment. Gate prompts, models, tools, and memory policies with replay suites scored on pass^k [8], pin model versions, and apply the change-aware rule after each release. Enforce authority outside

the model with a policy engine that fails closed and logs every denial [18, 19]. Budget cost like reliability by attributing spend to tasks and alerting on cost burn [2]. Treat traces as sensitive data, with redaction at collection. Together, SLOs, budgets, and the autonomy policy give governance programs measurable controls for the Measure and Manage functions of the NIST AI Risk Management Framework [21].

Limitations and future work

Some tasks lack a machine-checkable end state, leaving the task SLI to rest on sampling and calibrated judges. SLO targets are workflow-specific. Attributing failures in multi-agent tasks to one agent and one budget remains open [14]. The slow detection of gradual failures (F3, F6) suggests pairing outcome SLOs with leading indicators such as step counts and retrieval quality. Validating these results against production incident data is the most important next step.

9 CONCLUSION

AIOps taught operations teams to predict failures from telemetry. Agents first require a reliable way to know whether the work was done. Verified task success, measured independently of the agent's own report, is the SLI on which the rest of the framework depends. In simulation, it was the only single signal that detected every failure class, and a change-aware rule reduced its detection latency for the most common regressions from hours to under an hour. Layered SLIs locate problems, the failure taxonomy directs responders, and an error-budget policy turns reliability data into decisions about how much an agent is trusted to do.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Disclosure of conflict of interest

Adedamola Solanke declares that he has no conflict of interest.

REFERENCES

- [1] Solanke AA. (2022). AIOps evolution: From reactive monitoring to predictive service management in hybrid cloud environments. *Frontiers in Computer Science and Information Technology*, 3(1), 37-59.
- [2] Solanke AA. (2025). AI-enhanced FinOps: Predictive cost optimization across AWS, Azure, and GCP. *International Journal of Current Science*, 15(1), 353-367.
- [3] Solanke AA. (2021). Zero trust security architectures for multi-cloud environments: Implementation strategies and measurable outcomes. *World Journal of Advanced Engineering Technology and Sciences*, 3(2), 122-134.
- [4] Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K and Cao Y. (2023). ReAct: Synergizing reasoning and acting in language models. In: *Proceedings of the International Conference on Learning Representations (ICLR)*.
- [5] Advani L. (2026). From confident closing to silent failure: Characterizing false success in LLM agents. *ICML 2026 Workshop on Failure Modes in Agentic AI (FA-GEN)*. arXiv:2606.09863.
- [6] Beyer B, Jones C, Petoff J and Murphy NR. (2016). Site reliability engineering: How Google runs production systems. O'Reilly Media, Sebastopol, CA, USA.
- [7] Beyer B, Murphy NR, Rensin DK, Kawahara K and Thorne S. (2018). The site reliability workbook: Practical ways to implement SRE. O'Reilly Media, Sebastopol, CA, USA.
- [8] Yao S, Shinn N, Razavi P and Narasimhan K. (2025). τ -bench: A benchmark for tool-agent-user interaction in real-world domains. In: *Proceedings of the International Conference on Learning Representations (ICLR)*, 74824-74876.
- [9] Kapoor S, Stroebel B, Siegel ZS, Nadgir N and Narayanan A. (2025). AI agents that matter. *Transactions on Machine Learning Research*.
- [10] Shetty M, et al. (2024). Building AI agents for autonomous clouds: Challenges and design principles. In: *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 99-110.
- [11] Chen Y, et al. (2025). AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. In: *Proceedings of Machine Learning and Systems (MLSys)*.
- [12] Angi A, Sacco A and Marchetto G. (2025). LLNet: An intent-driven approach to instructing softwarized network devices using a small language model. *IEEE Transactions on Network and Service Management*, 22(4), 3403-3418.
- [13] Dong L, Lu Q and Zhu L. (2024). AgentOps: Enabling observability of LLM agents. arXiv preprint arXiv:2411.05285.

- [14] Cemri M, et al. (2025). Why do multi-agent LLM systems fail? In: *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.
- [15] Microsoft AI Red Team. (2025). Taxonomy of failure modes in agentic AI systems. White paper, Microsoft Corporation, Redmond, WA, USA.
- [16] Yang GY and Wang F. (2025). Taming silent failures: A framework for verifiable AI reliability. *IEEE Reliability Magazine*. arXiv:2510.22224.
- [17] Zheng L, et al. (2023). Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In: *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.
- [18] Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T and Fritz M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In: *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 79-90.
- [19] OWASP GenAI Security Project. (2025). OWASP Top 10 for agentic applications. Open Worldwide Application Security Project.
- [20] Microsoft AI Red Team. (2026). Updating the taxonomy of failure modes in agentic AI systems: What a year of red teaming taught us. *Microsoft Security Blog*, 4 June 2026.
- [21] National Institute of Standards and Technology. (2023). Artificial intelligence risk management framework (AI RMF 1.0). NIST AI 100-1, Gaithersburg, MD, USA.