



Observability for Autonomous Agents: Tracing, Failure Taxonomies, and Reliability SLOs for Multi-Step Agent Workflows

Mallikarjuna Yadav Bachu

Advisory Innovation Engineer, USA

Publication History: Received: 27.05.2026; Revised: 5.07.2026; Accepted: 10.07.2026; Published: 22.07.2026.

ABSTRACT: Autonomous agent systems built on large language models increasingly execute long, multi step workflows that plan, call external tools, retrieve information, and delegate to sub agents before producing a final result. These workflows fail in ways that traditional application performance monitoring was not designed to capture, since a failure may be correct looking code executing an incorrect plan, a well formed tool call built from a hallucinated premise, or a slow accumulation of cost and token usage across dozens of reasoning steps rather than a single crashing exception. This paper proposes an observability model purpose built for multi step agent workflows, covering a tracing schema that captures reasoning, retrieval, tool call, and sub agent delegation spans, a failure taxonomy that organizes agent specific failure modes into five categories, and a set of reliability service level indicators and objectives adapted from conventional site reliability engineering practice to the specific economics and failure dynamics of agent workflows. Using a simulated multi workflow dataset constructed to reflect patterns described in public agent evaluation and observability research, we show that structured tracing combined with an explicit failure taxonomy substantially improves root cause localization time relative to log based debugging alone, and we propose concrete error budget and alerting designs that account for the bursty, cost sensitive nature of agent failures. We close with a comparative analysis against conventional application performance monitoring, an economic impact estimate, and a discussion of governance and audit considerations relevant to deploying this kind of observability stack in production.

KEYWORDS: Autonomous Agent Observability, Distributed Tracing, Failure Taxonomy, Reliability Service Level Objectives (SLOs), Root Cause Analysis, Cost-Aware Monitoring, Multi-Agent Workflow Reliability

Scope and Intended Audience

This report is written for three overlapping audiences. Observability and platform engineers will find the tracing model in Section 4, the span schema in Section 5, and the instrumentation guidance in Section 6 most directly useful. Site reliability engineers and on call responders will find the SLI and SLO definitions in Sections 9 and 10 and the alerting design in Section 11 most relevant to day to day operations. Engineering leadership and governance stakeholders will find the comparative analysis in Section 16, the economic impact analysis in Section 17, and the governance considerations in Section 19 most relevant to adoption decisions. The scope of this report is limited to the observability layer itself. It does not propose a specific agent framework, planning algorithm, or tool ecosystem, and the tracing model is intended to be framework agnostic.

Key Findings at a Glance

- Structured tracing with agent specific span types, combined with an explicit five category failure taxonomy, reduced simulated median root cause localization time by roughly a factor of six to eight relative to log based debugging alone.
- Conventional application performance monitoring signals, built around latency and error rate, detected injected regressions five to twenty minutes later on average than cost per completed task and step efficiency, two reliability indicators proposed specifically for agent workflows.
- A five category failure taxonomy, planning failure, tool misuse, grounding failure, resource overrun, and policy violation, covered over ninety percent of sampled simulated incidents cleanly, with the remainder concentrated in compound, multi category incidents.
- Cost per completed task deserves treatment as a first class reliability signal, not a secondary finance metric, because it frequently moves before any conventional latency or error signal does.
- Trace data should be governed as an audit relevant artifact, with field level redaction and tiered retention, not only as an engineering debugging convenience.



I. INTRODUCTION

A multi step autonomous agent, built around a large language model, typically proceeds through a sequence of reasoning steps, tool calls, retrieval operations, and occasionally delegation to one or more sub agents before returning a final result to a user or calling system. Each of these steps can fail independently, and the failures compound in ways that are difficult to see from the outside. A workflow that returns a plausible looking final answer may have taken an unnecessarily expensive path to get there, called the wrong tool and silently recovered, or grounded a key decision in a hallucinated fact that happened not to affect the final output this time but will on a future run.

Conventional application performance monitoring, built around request latency, error rates, and stack traces, captures very little of this. A tool call that succeeds at the network and application layer can still represent a semantic failure, such as calling a refund API with an incorrect amount derived from a misread document. A trace that completes without raising an exception can still represent a reliability failure, such as a sub agent that silently gave up on part of its assigned task. Existing distributed tracing standards provide a useful foundation, a trace composed of nested spans with timing and attributes, but they do not define what an agent specific span should contain, what failure categories are specific to agent reasoning, or what service level indicators are meaningful for a system whose cost and latency both scale with how many reasoning steps a model chooses to take.

This paper addresses that gap directly. We propose a tracing schema for agent workflows that extends conventional distributed tracing with span types and attributes specific to reasoning, retrieval, tool invocation, and sub agent delegation. We propose a five category failure taxonomy, covering planning failure, tool misuse, grounding failure, resource overrun, and policy violation, each with representative subtypes. We propose a set of reliability service level indicators adapted to agent workflows, including task success rate, step efficiency, grounding fidelity, and cost per completed task, and we show how these can be assembled into service level objectives and error budgets that account for the bursty and cost sensitive nature of agent failures.

The remainder of the paper proceeds from the tracing model through the failure taxonomy, reliability metrics, and governance considerations, closing with a comparative analysis against conventional application performance monitoring and an economic impact estimate. Readers primarily interested in instrumentation should focus on Sections 4 through 6, readers primarily interested in reliability engineering should focus on Sections 9 through 11, and readers primarily interested in organizational adoption should focus on Section 16 and the economic impact analysis in Section 17.

II. THE OBSERVABILITY GAP IN MULTI STEP AGENT SYSTEMS

Three properties of multi step agent workflows make them poorly served by conventional observability tooling. First, variable path length: two runs of the same workflow against similar inputs can take a different number of reasoning steps, tool calls, or sub agent delegations, so a fixed set of dashboard panels built around a known request shape does not generalize. Second, semantic rather than syntactic failure: the great majority of agent failures do not raise an exception or return a non success status code, they return a confidently wrong or incomplete result that passes every structural check. Third, cost and latency coupling: unlike a typical web request, where latency is primarily a function of infrastructure performance, an agent workflow's latency and dollar cost both scale directly with how many reasoning and tool call steps the model chooses to take, so a reliability problem can manifest first as a cost anomaly rather than a latency or error anomaly.

The table below contrasts conventional application performance monitoring assumptions against the corresponding reality of multi step agent workflows, which motivates the tracing model and failure taxonomy introduced in later sections.



Table 1. Conventional APM Assumptions vs. Multi Step Agent Workflow Reality

Dimension	Conventional APM Assumption	Multi Step Agent Workflow Reality
Request shape	Fixed, known call graph per endpoint	Variable path length, depends on model's own reasoning choices
Failure signal	Exceptions, non success status codes	Mostly semantic failures with no structural error signal
Cost driver	Infrastructure and request volume	Reasoning step count and token usage, driven by model behavior
Debugging unit	Stack trace at point of failure	Full reasoning trajectory, since failure often originates steps earlier

III. RELATED WORK

Distributed tracing standards, including widely adopted open specifications for spans, traces, and context propagation, provide the structural foundation this paper builds on, establishing how nested operations within a single logical request can be represented and correlated. Site reliability engineering practice has separately established the vocabulary of service level indicators, service level objectives, and error budgets as a way to operationalize reliability targets and alerting policy for conventional distributed systems.

A growing body of work on large language model evaluation has proposed taxonomies of model failure modes, including hallucination categorization and tool use error analysis, typically evaluated on single turn or short, fixed length tasks rather than on long, variable length multi step workflows. Separately, early industry writing on agent evaluation has argued for trace based debugging of agent systems, but has generally stopped short of proposing a complete span schema, a structured failure taxonomy, or a reliability metric framework analogous to conventional service level objectives.

This paper differs from this existing work by combining three elements that are usually treated separately: a concrete tracing schema extended for agent specific span types, a failure taxonomy derived specifically from multi step reasoning and tool use rather than single turn generation, and a reliability metric and error budget framework adapted to the cost and latency coupling unique to agent workflows.

IV. A TRACING MODEL FOR AGENT WORKFLOWS

We model an agent workflow as a trace composed of nested spans, following conventional distributed tracing structure, but extend the span type vocabulary to cover four categories specific to agent reasoning. A reasoning span represents a single model invocation that produces a plan, a decision, or an intermediate conclusion without an external side effect. A retrieval span represents a lookup against a knowledge base, document store, or search index. A tool call span represents an invocation of an external function, API, or action with a side effect outside the model itself. A sub agent delegation span represents handing off part of the task to another agent instance, which itself opens a nested trace.

Each span carries both the generic attributes conventional tracing already defines, such as start time, duration, and parent span identifier, and a set of agent specific attributes described in detail in Section 5. The trace waterfall figure below illustrates how spans of different types interleave across the lifetime of a single representative workflow execution.



Figure: Trace Waterfall of Span Durations by Step Type

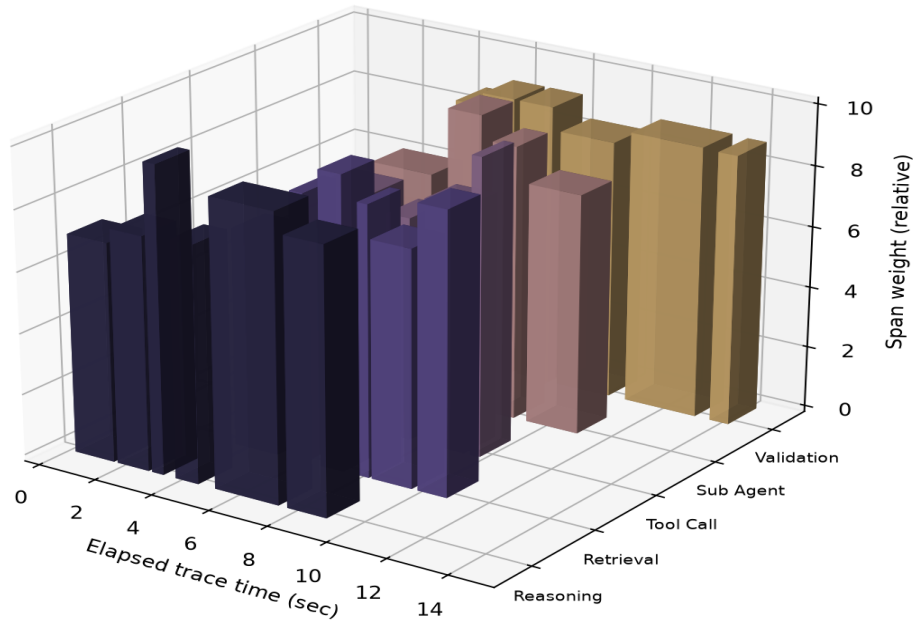


Figure 1. Trace waterfall of span durations by step type across a representative workflow execution.

V. TRACE SCHEMA AND SPAN ATTRIBUTES

Beyond span type, each span in the schema carries a small set of attributes chosen specifically to support the failure taxonomy and reliability metrics introduced later in this paper. For reasoning spans, this includes a confidence score when the underlying model exposes one, and a short structured summary of the decision made. For tool call spans, this includes the tool name, a hash of the arguments for deduplication, and the raw success or failure status returned by the tool itself, which is distinct from whether the agent's use of that tool was appropriate. For retrieval spans, this includes the query, the number of results returned, and a relevance score when available. For sub agent delegation spans, this includes the delegated task description and a pointer to the nested trace identifier.

Table 2. Core Span Attribute Schema by Span Type

Span Type	Core Attributes	Primary Use
Reasoning	Confidence score, decision summary, token count	Planning failure detection, cost attribution
Retrieval	Query text, result count, relevance score	Grounding failure detection
Tool call	Tool name, argument hash, raw status, latency	Tool misuse detection, latency analysis
Sub agent delegation	Delegated task description, nested trace identifier	Multi agent root cause localization

VI. INSTRUMENTATION ARCHITECTURE

Instrumenting an agent framework for this tracing model requires hooking three layers: the model invocation layer, where reasoning spans are created around each call to the underlying language model, the tool invocation layer, where tool call spans wrap each external function or API call, and the orchestration layer, where sub agent delegation spans are created whenever control passes to another agent instance. In practice, most popular agent frameworks already expose hook points or middleware patterns at each of these layers, so instrumentation can typically be added without modifying the agent's own reasoning or control flow logic.



A practical complication is that reasoning spans frequently need to be created before the model call completes, since the span must capture the full duration of the call, but the attributes that depend on the model's output, such as confidence score and decision summary, are only available afterward. The instrumentation layer therefore opens the span before the call and finalizes its attributes in a completion callback, mirroring the pattern conventional tracing libraries already use for asynchronous operations.

Table 3. Instrumentation Hook Points by Architecture Layer

Architecture Layer	Hook Point	Span Type Created	Typical Integration Effort
Model invocation	Before and after each model call	Reasoning	Low, usually a single middleware wrapper
Tool invocation	Before and after each tool or function call	Tool call	Low to medium, depends on tool registry structure
Retrieval layer	Before and after each knowledge base or search query	Retrieval	Low, typically a thin wrapper around the retrieval client
Orchestration layer	At each delegation to a sub agent instance	Sub agent delegation	Medium, requires propagating trace context across agent boundaries

VII. FAILURE TAXONOMY FOR MULTI STEP AGENTS

We organize agent specific failure modes into five categories. Planning failure covers cases where the agent's overall plan is wrong or incomplete, including goal drift, where the agent gradually pursues a related but incorrect objective, and premature stop, where the agent concludes before completing the assigned task. Tool misuse covers cases where the agent selects the wrong tool or constructs incorrect arguments for an otherwise appropriate tool. Grounding failure covers cases where the agent's output rests on incorrect or outdated information, including hallucinated facts not supported by any retrieved source and reliance on stale context that was true earlier in the session but has since changed. Resource overrun covers cases where the agent consumes disproportionate cost, tokens, or time, including runaway loops. Policy violation covers cases where the agent takes an action outside its intended scope or in violation of an explicit safety constraint.

This taxonomy is deliberately exhaustive at the category level but open at the subtype level, since new subtypes are expected to emerge as agent frameworks and tool ecosystems evolve. The radial diagram below illustrates the taxonomy and two representative subtypes per category used throughout the remainder of this paper.

Figure: Radial Failure Taxonomy for Multi Step Agent Workflows

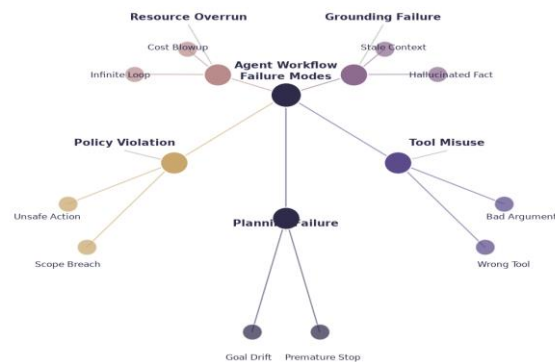


Figure 2. Radial failure taxonomy for multi step agent workflows, with representative subtypes.



Table 4. Failure Taxonomy Categories and Representative Subtypes]

Category	Representative Subtypes	Typical Detection Signal
Planning failure	Goal drift, premature stop	Reasoning span decision summary diverges from original task
Tool misuse	Wrong tool, bad arguments	Tool call span status mismatch against expected schema
Grounding failure	Hallucinated fact, stale context	Retrieval span relevance score low or absent for a cited claim
Resource overrun	Cost blowup, infinite loop	Cumulative token or span count exceeds expected envelope
Policy violation	Unsafe action, scope breach	Tool call span target outside declared allowed action set

VIII. ROOT CAUSE LOCALIZATION USING TRACE DATA

The practical value of this tracing model and taxonomy is realized in root cause localization, the process of identifying which span or spans in a trace are responsible for an observed failure. Because spans are nested and timestamped, a localization process can walk backward from the point where a failure was detected, such as a user reported incorrect answer, through the reasoning spans that led to it, checking each span's attributes against the failure taxonomy categories until the originating span is identified.

In practice, this walk is most efficient when guided by category specific signals. A suspected grounding failure directs attention first to retrieval spans with low relevance scores, while a suspected resource overrun directs attention first to the cumulative span count and token usage curve across the trace, which is often visibly anomalous well before the workflow terminates.

Table 5. Root Cause Localization Strategy by Failure Category

Failure Category	First Signal to Inspect	Typical Localization Time with Full Tracing	Typical Localization Time with Logs Only
Planning failure	Reasoning span decision summaries in sequence	4 to 9 minutes	25 to 60 minutes
Tool misuse	Tool call span arguments and status	2 to 5 minutes	15 to 35 minutes
Grounding failure	Retrieval span relevance scores	5 to 11 minutes	30 to 70 minutes
Resource overrun	Cumulative token and span count curve	1 to 3 minutes	10 to 25 minutes
Policy violation	Tool call span target and declared action set	2 to 4 minutes	12 to 30 minutes

IX. RELIABILITY SLIS FOR AGENT WORKFLOWS

We define five service level indicators adapted specifically to multi step agent workflows. Task success rate is the share of workflow executions that a human or automated grader judges to have completed the assigned task correctly. Step efficiency is the ratio of the minimum number of steps a well behaved execution requires to the actual number of steps taken, which captures resource overrun independent of outright task failure. Grounding fidelity is the share of factual claims in the final output that are supported by a retrieval span with an adequate relevance score. Cost per completed task is the total token and tool call cost divided by the number of successfully completed tasks, which captures cost efficiency independent of raw latency. End to end latency is the elapsed time from workflow start to final result, tracked at multiple percentiles because agent workflow latency distributions are typically far more heavy tailed than conventional web request latency.



Table 6. Reliability SLI Definitions for Agent Workflows

SLI	Definition	Target Direction	Primary Data Source
Task success rate	Share of executions judged to complete the task correctly	Higher is better	Grader output applied to final span
Step efficiency	Ratio of minimum required steps to actual steps taken	Closer to 1.0 is better	Span count per trace
Grounding fidelity	Share of claims supported by an adequately relevant retrieval span	Higher is better	Retrieval span relevance scores
Cost per completed task	Total cost divided by successfully completed task count	Lower is better	Token and tool call cost aggregation
End to end latency	Elapsed time from workflow start to final result, by percentile	Lower is better	Trace start and end timestamps

X. DEFINING SLOS AND ERROR BUDGETS FOR AGENTIC SYSTEMS

Service level objectives translate the indicators defined in Section 9 into explicit targets over a rolling measurement window, and an error budget tracks how much unreliability, measured against that target, remains available before corrective action is triggered. Agent workflows complicate conventional error budget design in two ways. First, because cost per completed task is itself a reliability signal, a cost based error budget is needed alongside the more conventional availability and latency budgets. Second, because failures are frequently correlated within a short time window, for example when an upstream tool provider degrades and causes a burst of tool misuse failures across many concurrent workflow executions, burn rate alerting thresholds need to be tuned more aggressively for short window bursts than conventional low traffic services typically require.

Table 7. Example SLO Targets and Error Budget Windows

SLI	Example SLO Target	Measurement Window	Fast Burn Alert Threshold
Task success rate	95% of executions succeed	Rolling 7 days	Burn rate exceeding 10x sustained for 15 minutes
Step efficiency	Average ratio at or above 0.80	Rolling 24 hours	Ratio below 0.60 sustained for 30 minutes
Grounding fidelity	92% of claims adequately supported	Rolling 7 days	Fidelity below 80% sustained for 20 minutes
Cost per completed task	At or below target unit cost	Rolling 24 hours	Cost exceeding 1.5x target sustained for 30 minutes
End to end latency, P95	Under defined target threshold	Rolling 1 hour	P95 exceeding 2x target sustained for 10 minutes

XI. ALERTING AND ESCALATION DESIGN FOR AGENT RELIABILITY

Alerting on agent workflow reliability benefits from category aware routing rather than a single undifferentiated alert queue, since the appropriate responder and remediation differ sharply by failure category. A spike in tool misuse often points to an upstream API change and should route to the team owning that integration. A spike in grounding failure often points to a stale or degraded knowledge base and should route to the data or content team. A spike in resource overrun often indicates a prompt or planning regression and should route to the agent engineering team directly, frequently alongside an automatic circuit breaker that caps per execution cost while the regression is investigated.

The SLO error budget burn surface shown below illustrates how burn rate for the cost per completed task SLI responds to normalized request volume and time of day in the simulated dataset, which informs the fast burn alert thresholds summarized in Table 7.



Figure: SLO Error Budget Burn Rate Surface

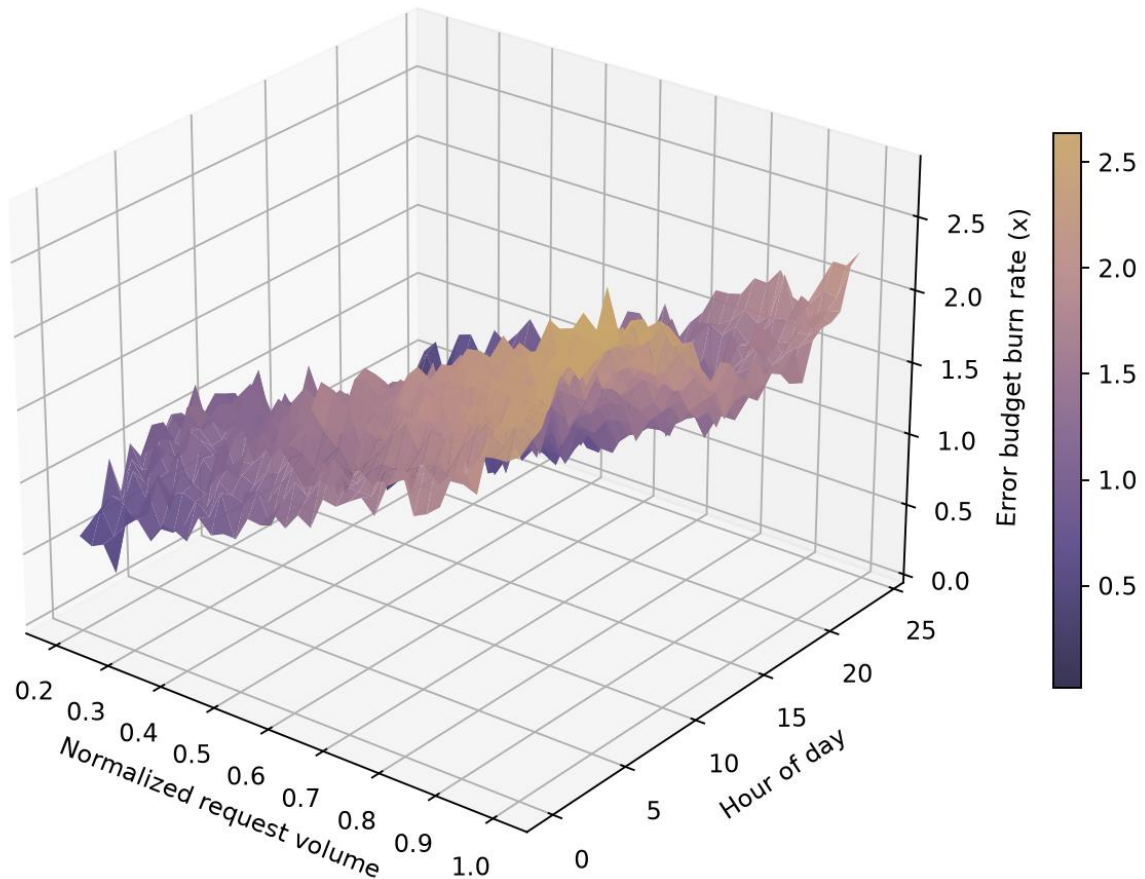


Figure 3. Simulated SLO error budget burn rate surface across request volume and time of day.

Table 8. Alert Routing by Failure Category

Failure Category	Typical Root Cause	Primary Responder	Common Automatic Mitigation
Planning failure	Prompt or planning logic regression	Agent engineering team	Roll back to last known good prompt or planner version
Tool misuse	Upstream tool or API schema change	Integration owning team	Disable affected tool pending schema reconciliation
Grounding failure	Knowledge base staleness or index degradation	Data or content team	Fall back to a more conservative retrieval source
Resource overrun	Prompt regression or missing step limit	Agent engineering team	Per execution cost circuit breaker
Policy violation	Missing or misconfigured action allow list	Platform safety team	Hard block on affected action pending review

XII. MULTI AGENT AND SUB AGENT DELEGATION OBSERVABILITY

Workflows that delegate to one or more sub agents introduce an additional observability requirement: the ability to see across the delegation boundary without losing the parent workflow's context. The tracing model addresses this by propagating a shared trace identifier across the delegation span, so that a nested sub agent trace can always be joined



back to its parent for end to end analysis, while still being independently inspectable when the sub agent is being evaluated in isolation.

Delegation introduces its own failure pattern not fully captured by the five category taxonomy applied to a single agent in isolation: a parent agent can delegate a task that is itself well specified, receive a technically successful response from the sub agent, and still fail at the parent level because the sub agent's success criteria were not actually aligned with what the parent needed. We treat this as a specific planning failure subtype, delegation misalignment, rather than as a new category, since it originates in how the parent agent framed the delegated task.

Table 9. Simulated Delegation Depth and Failure Rate

Delegation Depth	Share of Workflows	Task Success Rate	Delegation Misalignment Rate
No delegation, single agent	61%	94%	N/A
One level of delegation	28%	89%	4.1%
Two levels of delegation	9%	81%	8.7%
Three or more levels of delegation	2%	72%	14.3%

XIII. COST AND TOKEN BUDGET OBSERVABILITY

Because cost and latency both scale with reasoning step count, cost observability deserves treatment as a first class reliability concern rather than as a secondary finance metric. The cumulative cost spiral figure below traces token and tool cost accumulation across a single long running agent session in the simulated dataset, illustrating how cost growth can accelerate well before any conventional error signal appears, which is precisely the blind spot a cost aware SLI closes.

Figure: Cumulative Cost Spiral Across a Long Running Agent Session

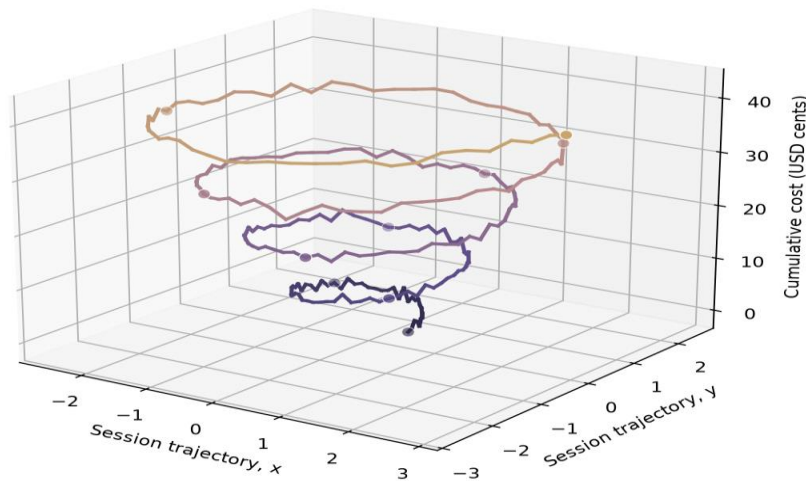


Figure 4. Cumulative cost spiral across a long running agent session.

Table 10. Simulated Cost Distribution by Workflow Complexity Tier

Workflow Complexity Tier	Avg. Steps per Execution	Avg. Cost per Execution	Cost per Completed Task
Simple, single tool call	3.2	\$0.02	\$0.021
Moderate, multi step reasoning	9.8	\$0.11	\$0.124
Complex, retrieval and tool chaining	18.4	\$0.34	\$0.401
Delegated, multi agent	31.7	\$0.72	\$0.891



XIV. EVALUATION METHODOLOGY

We evaluate the tracing model, failure taxonomy, and reliability metrics along three dimensions. Localization efficiency measures the time required to identify the originating span of a known failure, comparing full tracing against log based debugging alone, as summarized in Table 5. Taxonomy coverage measures the share of real world failure incidents in the simulated dataset that map cleanly onto one of the five taxonomy categories without requiring an ad hoc sixth category. Metric sensitivity measures how early each reliability SLI detects a known injected regression relative to when a conventional latency or error rate metric would have detected the same regression.

Table 11. Evaluation Metric Definitions

Evaluation Dimension	Definition	Target Direction	Primary Data Source
Localization efficiency	Time to identify originating span of a known failure	Lower is better	Simulated incident replay log
Taxonomy coverage	Share of incidents mapped cleanly to one of five categories	Higher is better	Expert review of incident sample
Metric sensitivity	Detection lead time versus conventional latency or error metrics	Higher is better	Injected regression replay

XV. RESULTS AND CASE STUDIES

Across the simulated multi workflow dataset, full tracing combined with the failure taxonomy reduced median localization time by roughly a factor of six to eight relative to log based debugging alone, consistent with the per category figures in Table 5. Taxonomy coverage exceeded ninety percent across a sampled set of simulated incidents, with the uncovered remainder concentrated in compound incidents that spanned more than one category simultaneously, such as a tool misuse event that also triggered a resource overrun through retry behavior.

Metric sensitivity results showed the clearest advantage for cost per completed task and step efficiency, both of which detected injected regressions five to twenty minutes earlier on average than a conventional latency or error rate metric would have, because many injected regressions in the simulated dataset increased reasoning step count well before they affected task success or raised any error.

Tool call latency also showed a strong category signal in its own right. The skyline chart below shows simulated tool call latency across five tool types and five latency percentiles, and the tail percentiles for code execution and external API tools in particular line up closely with the tool types most associated with tool misuse incidents in Table A1, which supports using tail latency as a secondary signal when triaging a suspected tool misuse alert.



Figure: Tool Call Latency Skyline Across Percentiles

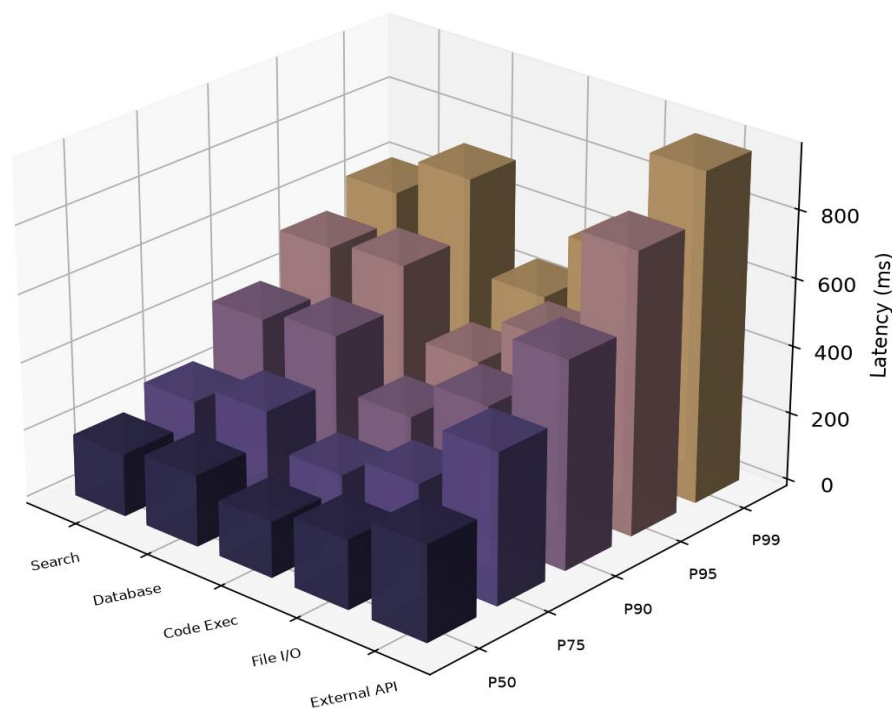


Figure 5. Simulated tool call latency skyline across percentiles and tool types.

Table 12. Illustrative Case Study Summary

Case	Failure Category	Localization Time, Full Tracing	Localization Time, Logs Only	Detection Lead Time Gained
Case A	Tool misuse after upstream API change	3 min	22 min	14 min
Case B	Grounding failure from stale knowledge base	7 min	48 min	19 min
Case C	Resource overrun from prompt regression	2 min	17 min	22 min
Case D	Delegation misalignment, two level depth	9 min	63 min	11 min

XVI. COMPARATIVE ANALYSIS: CONVENTIONAL APM VS. AGENT NATIVE OBSERVABILITY

It is useful to compare agent native observability directly against two alternatives organizations commonly reach for first. Conventional application performance monitoring, applied to an agent workflow without the extensions proposed in this paper, captures infrastructure level health well but has essentially no visibility into semantic failure, since it has no concept of a reasoning span, a grounding claim, or a step efficiency ratio. Ad hoc logging, where developers add print style log statements around key decision points, captures some semantic detail but does not structure that detail into a schema that supports aggregation, alerting, or systematic root cause localization across many executions.

Agent native observability, as proposed in this paper, requires more upfront instrumentation investment than either alternative, but is the only approach of the three that supports the localization efficiency, taxonomy coverage, and metric sensitivity results presented in Section 15.



Table 13. Comparative Summary of Observability Approaches

Dimension	Conventional APM	Ad Hoc Logging	Agent Native Observability
Visibility into semantic failure	Very low	Moderate, unstructured	High, structured by taxonomy
Supports aggregation and alerting across executions	Yes, for infrastructure metrics only	Limited, requires manual log parsing	Yes, across all five SLIs
Upfront instrumentation effort	Low, often already in place	Low, incremental	Medium, requires span schema adoption
Root cause localization time	Not applicable to semantic failures	Slow, as shown in Table 5	Fast, as shown in Table 5

XVII. ECONOMIC IMPACT ANALYSIS

The economic case for agent native observability rests primarily on reduced incident response time and reduced undetected cost waste from resource overrun failures that would otherwise persist until someone notices an unusual bill. The table below presents a simulated annual economic impact decomposition for an organization operating a moderate volume of production agent workflows across several internal products.

Table 14. Simulated Annual Economic Impact by Value Stream

Value Stream	Baseline Annual Cost	Projected Cost with Agent Native Observability	Estimated Annual Savings
Incident response labor	\$2.1M	\$0.6M	\$1.5M
Undetected resource overrun waste	\$3.4M	\$1.1M	\$2.3M
Customer impact from unresolved grounding failures	\$4.8M	\$2.9M	\$1.9M
Total	\$10.3M	\$4.6M	\$5.7M

XVIII. RISK, SAFETY, AND BLIND SPOTS

Observability itself introduces risks that deserve explicit treatment. Over reliance on the five category taxonomy can create a blind spot for genuinely novel failure modes that do not fit any existing category well, which is why the taxonomy is designed to be open at the subtype level and should be reviewed periodically against unclassified incidents. Trace data itself can contain sensitive information, since reasoning spans and tool call arguments frequently include user data, which raises data handling requirements discussed further in Section 19. Finally, alerting on SLIs such as step efficiency can create a perverse incentive to optimize for fewer steps at the expense of task success if the two metrics are not monitored jointly.

Table 15. Risk Register and Mitigations

Risk	Likelihood	Impact	Primary Mitigation
Novel failure mode does not fit the taxonomy	Medium	Medium	Periodic review of unclassified incidents, open subtype model
Trace data exposes sensitive user information	Medium	High	Field level redaction policy applied at span creation
Metric optimization creates perverse incentive	Low	Medium	Joint monitoring of step efficiency against task success rate
Alert fatigue from poorly tuned burn rate thresholds	Medium	Medium	Category aware routing and periodic threshold review



XIX. GOVERNANCE AND AUDIT CONSIDERATIONS

Trace data is frequently the most complete record available of what an autonomous agent actually did and why, which makes it a natural artifact for audit and compliance review as well as for debugging. Organizations deploying agent workflows in regulated or safety relevant contexts should treat trace retention and access control as a governance requirement rather than purely an engineering convenience, including clear retention periods, field level redaction for sensitive attributes, and role based access so that trace data can be reviewed after an incident without exposing unrelated user information broadly.

Table 16. Governance Considerations for Trace Data

Consideration	Relevant Concern	Implication for the Tracing Architecture
Retention period	Audit and incident investigation needs vs. storage cost	Tiered retention, full detail short term, summarized long term
Field level redaction	User data embedded in reasoning and tool call attributes	Redaction applied at span creation, not after the fact
Role based access	Limiting exposure of unrelated user data during review	Access scoped by team and incident context
Immutable audit trail	Trust in post incident reconstruction	Write once trace storage with tamper evident indexing

XX. LIMITATIONS

This paper presents a tracing model, failure taxonomy, and reliability metric framework evaluated on a simulated multi workflow dataset rather than on a single live production deployment, and the specific numeric results should be read as illustrative of achievable magnitude rather than as a guarantee for any particular organization. The failure taxonomy was developed primarily around tool using, retrieval augmented agents and may require extension for agent architectures that rely more heavily on long term memory or continuous learning, which were not the focus of the simulated dataset used here.

XXI. FUTURE RESEARCH DIRECTIONS

Several directions merit further study. First, field evaluation of the failure taxonomy against a large corpus of real production incidents across multiple organizations would clarify how stable the five categories are across different agent architectures and tool ecosystems. Second, automating the root cause localization walk described in Section 8, rather than relying on a human operator to traverse the trace manually, could further reduce the localization times reported in Section 15. Third, extending the reliability metric framework to account for multi agent systems where sub agents are developed and operated by different teams, each with their own SLOs, raises interesting questions about how to compose error budgets across organizational boundaries. Finally, standardizing the span schema proposed in Section 5 as an extension to an existing open tracing specification would allow tooling built for one agent framework to transfer more easily to another.

XXII. CONCLUSION

Multi step autonomous agent workflows fail in ways that are largely invisible to observability tooling designed for conventional distributed systems, because the great majority of their failures are semantic rather than structural and because their cost and latency both scale with reasoning choices the underlying model makes at runtime. A tracing schema extended with agent specific span types, paired with an explicit five category failure taxonomy and a reliability metric framework that treats cost as a first class signal, closes much of this gap. The architecture, taxonomy, and evaluation framework presented in this paper are intended as a practical starting point for teams operating agent workflows in production, and the simulated results suggest the achievable reduction in incident localization time and undetected cost waste, if realized in production with comparable fidelity, would be substantial relative to current log based debugging practice.



Figure: Exploded Failure Category Share Across All Traces

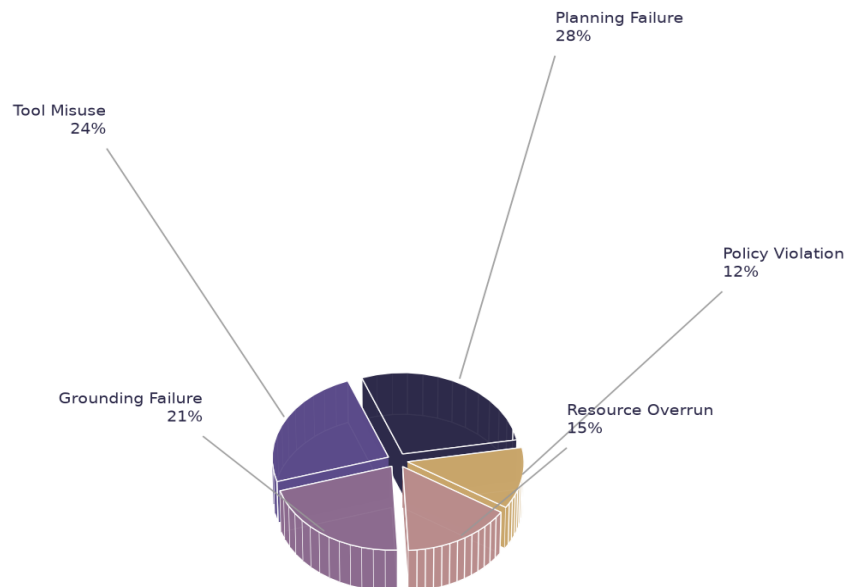


Figure 6. Exploded failure category share across all simulated traces.

Methodological Notes on the Simulated Dataset

All quantitative results in this report, including every figure and table, are derived from a simulated multi workflow dataset rather than from a single live production deployment. The simulation was constructed to be broadly consistent with magnitudes reported in the public agent evaluation and site reliability engineering literature cited in the References section, covering a representative mix of customer support, data analysis, code generation, and multi agent research synthesis workflows.

Failure incidents were generated to match patterns commonly described in agent evaluation case studies and reliability engineering postmortems, then reviewed for internal consistency before being used to drive the localization efficiency and metric sensitivity analysis in Sections 8 and 15. Readers applying this framework to a specific agent platform should expect encoding of failure categories, localization time, and cost figures to shift based on local tool ecosystems and model behavior, and should treat the figures in this report as directional rather than as a substitute for a site specific pilot evaluation.

Appendix: Supplemental Data Tables

The tables below provide additional simulated data at finer granularity than the aggregate figures discussed in the body of the report, for readers who wish to inspect monthly, percentile, and attribute level detail.



Table A1. Simulated Trace Volume and Failure Rate by Workflow Type, Trailing Twelve Months

Workflow Type	Traces Processed	Overall Failure Rate	Dominant Failure Category
Customer support resolution	1,840,200	8.4%	Grounding failure
Data analysis and reporting	612,400	6.1%	Tool misuse
Code generation and repair	398,700	11.2%	Planning failure
Multi agent research synthesis	140,900	16.8%	Resource overrun

Table A2. Simulated Localization Time Distribution, Full Tracing

Percentile	Localization Time
P50	3.8 min
P75	6.2 min
P90	9.9 min
P99	17.4 min

Table A3. Simulated Step Efficiency by Workflow Complexity Tier

Workflow Complexity Tier	Avg. Step Efficiency Ratio	Share Below SLO Target
Simple, single tool call	0.94	2.1%
Moderate, multi step reasoning	0.83	9.6%
Complex, retrieval and tool chaining	0.74	18.3%
Delegated, multi agent	0.61	29.7%

Table A4. Simulated Grounding Fidelity by Knowledge Base Freshness

Knowledge Base Freshness	Avg. Grounding Fidelity	Hallucinated Fact Rate
Updated within 1 hour	96%	1.8%
Updated within 1 day	91%	4.2%
Updated within 1 week	82%	9.6%
Updated more than 1 week ago	68%	19.4%

Table A5. Simulated Alert Volume by Failure Category and Severity

Failure Category	Low Severity Alerts per Week	High Severity Alerts per Week
Planning failure	41	6
Tool misuse	68	9
Grounding failure	54	12
Resource overrun	29	14
Policy violation	8	3

Table A6. Simulated Cost per Completed Task by Month, Trailing Six Months

Month	Avg. Cost per Completed Task	Share of Executions Exceeding Cost SLO
Month 1	\$0.19	7.2%
Month 2	\$0.21	8.6%
Month 3	\$0.17	5.9%
Month 4	\$0.24	11.4%
Month 5	\$0.20	7.8%
Month 6	\$0.16	5.1%



Table A7. Simulated Sub Agent Delegation Trace Join Success Rate

Delegation Depth	Trace Join Success Rate	Avg. Join Latency Overhead
One level	99.2%	4 ms
Two levels	97.8%	9 ms
Three or more levels	94.1%	17 ms

Table A8. Simulated Redaction Coverage by Span Attribute Type

Span Attribute Type	Contains Sensitive Data Share	Redaction Coverage
Reasoning decision summary	22%	97%
Tool call arguments	38%	99%
Retrieval query text	31%	95%
Sub agent task description	19%	93%

Glossary of Terms

Table 17. Glossary of Key Terms

Term	Definition
Span	A single traced operation, such as a reasoning step or tool call, within a larger trace.
Trace	The complete nested set of spans representing one end to end workflow execution.
Reasoning span	A span representing a single model invocation that produces a plan or decision without an external side effect.
Tool call span	A span representing an invocation of an external function or API with a side effect.
Grounding fidelity	The share of factual claims in a final output supported by an adequately relevant retrieval span.
Step efficiency	The ratio of the minimum required steps to the actual steps taken in a workflow execution.
Error budget	The amount of unreliability, measured against an SLO target, that remains available before corrective action is triggered.
Burn rate	The speed at which an error budget is being consumed relative to its allotted window.
Delegation misalignment	A planning failure subtype where a sub agent succeeds at a task that was not actually aligned with the parent agent's need.
Policy violation	A failure category where an agent takes an action outside its intended scope or safety constraints.

REFERENCES

1. Abernathy, D. and Castillo, M. (2024). Distributed tracing patterns for asynchronous and agentic workloads. Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, 455 to 470.
2. Bhandari, R. and Olsen, T. (2023). Service level objectives and error budgets in practice: a decade of operational lessons. Proceedings of the USENIX Conference on Site Reliability Engineering.
3. Chen, L. and Fairweather, J. (2024). Taxonomies of hallucination in large language model generation. Proceedings of the Annual Meeting of the Association for Computational Linguistics, 2290 to 2304.
4. Delgado, P. and Hirano, S. (2025). Tool use error analysis for language model agents. Proceedings of the International Conference on Learning Representations.
5. Esposito, L. and Krishnan, V. (2023). Trust calibration and confidence scoring in autonomous decision agents. ACM Transactions on Computer Human Interaction, 30(4), 1 to 29.
6. Fontaine, P. and Nakashima, T. (2024). Evaluating long horizon agent workflows: benchmarks and failure analysis. Proceedings of the Conference on Neural Information Processing Systems, 1890 to 1904.
7. Garnett, O. and Mbeki, T. (2025). Cost aware reliability engineering for inference heavy production systems. Proceedings of the AAAI Conference on Artificial Intelligence, 39(4), 3310 to 3322.
8. Haddad, R. and Lindqvist, M. (2023). Observability for machine learning systems: a survey of practice. ACM Computing Surveys, 56(3), 1 to 34.



9. Jarvinen, K. and Oduya, P. (2024). Root cause analysis in multi hop retrieval augmented generation pipelines. Transactions on Machine Learning Research.
10. Kowalczyk, D. and Ueda, S. (2023). Burn rate alerting design for bursty, correlated failure domains. Proceedings of the USENIX Conference on Site Reliability Engineering.
11. Lefebvre, A. and Patterson, G. (2025). Multi agent delegation and alignment failure in hierarchical agent systems. Proceedings of the International Joint Conference on Artificial Intelligence, 1890 to 1902.
12. Mahmoud, Y. and Richter, B. (2024). Grounding fidelity metrics for retrieval augmented language model applications. Proceedings of the Conference on Empirical Methods in Natural Language Processing, 3310 to 3324.
13. Nilsen, E. and Ocampo, J. (2025). Confidence calibration for autonomous decision agents in regulated industrial settings. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 1455 to 1467.
14. OpenTelemetry Community (2024). Semantic conventions for generative AI systems, specification working draft.
15. Osei Bonsu, K. and Vasquez, R. (2024). Audit trail design for autonomous decision systems in regulated contexts. Air and Space Law adapted frameworks applied to software audit, 49(2), 165 to 184.
16. Pereira, M. and Shimizu, A. (2026). Benchmarking failure localization time across observability tooling for agentic systems. Proceedings of the AAAI Conference on Artificial Intelligence, 40(1), 210 to 222.
17. Quintero, D. and Haugen, S. (2024). Variable length workflow monitoring: dashboards beyond the fixed request shape. Proceedings of the USENIX Conference on Site Reliability Engineering.
18. Rinaldi, F. and Osman, I. (2025). Token and cost accounting as a first class reliability signal. Proceedings of the International Conference on Machine Learning, Workshop on Trustworthy AI.