AgentTelemetry: A Fault Detection Benchmark and Toolkit for LLM Agent Observability

Anonymous Author(s)
Anonymous Institution

Abstract

LLM-based autonomous agents fail in ways that existing observability infrastructure cannot detect. OpenTelemetry's GenAI semantic conventions cover LLM invocation and tool execution but leave five critical agent orchestration phases—planning, reasoning, safety monitoring, inter-agent delegation, and memory management—without span-level representation. We present AgentTelemetry, an open-source benchmark suite and toolkit for evaluating fault detection in agent systems. The benchmark defines (1) a taxonomy of 14 fault types mapped to 9 agent-specific span kinds, (2) a controlled evaluation harness of 2,940 configurations (14 faults × 5 observability conditions × 7 frameworks × 6 models), and (3) a pip-installable library (3,700+ LOC, 78 tests) with adapters for seven frameworks. On the controlled benchmark, the full span taxonomy achieves a Fault Detection Rate (FDR) of 1.000—an upper bound confirming structural completeness—compared to 0.429 for vanilla OpenTelemetry and OTel+GenAI. An ablation study proves all nine span kinds are necessary: removing any one makes at least one fault type undetectable. A case study on 112 SWE-bench Lite instances reveals that reasoning loops account for 75% of agent failures (95% CI: [66%, 82%])—a failure mode invisible to vanilla OTel—and a telemetry-guided intervention improves the patch rate by +8.3 pp over a matched control. All code, data, and benchmark configurations are open-source for reproducibility.

CCS Concepts: • **Software and its engineering → Software verification and validation**.

Keywords: AI agents, observability, fault detection, benchmark, OpenTelemetry, LLM monitoring

Data and code: Anonymized repository available at https://anonymous.4open.science/r/agenttelemetry

1 Introduction

Large language model (LLM) agents pursue goals through iterative reasoning, planning, tool use, and delegation—producing deeply nested, non-deterministic execution traces. Frameworks such as LangChain, CrewAI, AutoGen, and LlamaIndex have accelerated adoption, but production teams lack standardized observability infrastructure to diagnose agent failures [5, 6]. Without structured telemetry, fundamental diagnostic questions remain unanswerable: *Why did the agent choose this tool?* (requires tracing from tool invocation through the LLM call and reasoning chain that selected it); *Where did the hallucination originate?* (requires comparing LLM output against retrieved documents); *How much did this task cost across all agents?* (requires agent-identity propagation with per-call token and pricing metadata).

OpenTelemetry (OTel) [1] is the standard for distributed tracing, and its GenAI semantic conventions define operations for LLM calls (`chat`, `text_completion`), tool execution (`execute_tool`), retrieval (`retrieval`, `embeddings`), and agent lifecycle (`create_agent`, `invoke_agent`). However, five agent orchestration phases—planning, reasoning, safety monitoring, delegation, and memory management—have no span-level representation, rendering them opaque `INTERNAL` spans indistinguishable from application logic. Table 1 summarizes this gap: five of the nine span kinds in our taxonomy have no OTel GenAI equivalent.

This gap matters for fault detection. Our controlled benchmark shows that vanilla OTel and OTel+GenAI both detect only 6 of 14 agent fault types (FDR = 0.429). The remaining 8 faults—including reasoning loops, guardrail bypass, planning failure, and memory corruption—require agent-specific span kinds that neither standard provides.

We present AgentTelemetry, a benchmark suite, fault taxonomy, and reusable toolkit for agent observability. Our contributions are:

1. **A fault detection benchmark** with 14 fault types, 9 span kinds, 5 observability conditions, and 2,940 reproducible configurations (§2).
2. **An ablation matrix** proving that all 9 span kinds are necessary: each is the sole detection signal for at least one fault type (§4.2).
3. **A SWE-bench case study** on 112 instances showing that reasoning loops dominate agent failures (75%) and that telemetry-guided intervention yields +8.3 pp improvement (§4.3).
4. **An open-source toolkit** (3,700+ LOC, 7 adapters, 78 tests) with deterministic mock clients for full reproducibility without API keys (§3).

Table 1. Agent concepts mapped to observability primitives. ● = overlapping with OTel GenAI; ▹ = extends OTel GenAI; ★ = novel, no OTel GenAI equivalent.

Agent Concept	OTel GenAI	AgentTelemetry	
Agent lifecycle	create/invoke_agent	**AGENT**	▹
LLM invocation	chat / text_compl.	**LLM_CALL**	●
Tool execution	execute_tool	**TOOL_CALL**	●
Doc. retrieval	retrieval, embeddings	**RETRIEVAL**	●
Task decomposition	(none)	**PLANNING**	★
Chain-of-thought	(none)	**REASONING**	★
Safety monitoring	(none)	**GUARD_RAIL**	★
Inter-agent handoff	(none)	**DELEGATION**	★
Memory read/write	(none)	**MEMORY**	★

Table 2. Framework coverage: which agent execution phases each framework natively exposes (✓) vs. requires adapter mapping (○).

Span Kind	*LangChain*	*CrewAI*	*AutoGen*	*LlamaIndex*	*Anthropic*	*OpenAI*	*Custom*
AGENT	✓	✓	✓	✓	○	○	✓
LLM_CALL	✓	✓	✓	✓	✓	✓	✓
TOOL_CALL	✓	○	✓	✓	✓	✓	✓
PLANNING	○	○	○	✓	○	○	✓
REASONING	✓	✓	✓	○	○	○	✓
RETRIEVAL	✓	○	○	✓	○	○	✓
GUARD_RAIL	○	○	○	○	○	○	✓
DELEGATION	○	✓	✓	○	○	○	✓
MEMORY	○	○	○	○	○	○	✓

2 Benchmark Design

2.1 Span Taxonomy

The benchmark is organized around a taxonomy of nine agent-specific span kinds derived from systematic analysis of seven production frameworks (LangChain, CrewAI, AutoGen, LlamaIndex, Anthropic SDK, OpenAI SDK, and a custom manual API). The derivation followed a four-stage process loosely modeled on open coding [17]. *Stage 1 (API inventory):* We catalogued 25 API entry points representing distinct execution phases across all seven frameworks. Callback-based frameworks (LangChain, CrewAI, LlamaIndex) expose named hooks (e.g., `on_retriever_start`); thin SDK wrappers (Anthropic, OpenAI) expose a single `create()` method with typed response blocks; orchestration frameworks (AutoGen) expose agent lifecycle methods. *Stage 2 (open coding):* Each entry point was assigned a candidate span kind label, producing 14 initial candidates. *Stage 3 (merging):* Candidates were merged using two criteria: observational indistinguishability and diagnostic redundancy—reducing 14 to 9 final span kinds. Five merges occurred: LLM_Chat + LLM_Completion → LLM_CALL; Reflection → REASONING; Agent_Comm → DELEGATION; Read_Memory + Write_Memory → MEMORY; Human_Input → removed (modeled as AGENT with `agent.type="human"`). *Stage 4 (saturation):* After 5 frameworks, no new span kinds emerged; frameworks 6–7 mapped to existing kinds. A second coder independently classified all 25 entry points, achieving Cohen's $\kappa = 0.904$ ("almost perfect" agreement).

Table 3 lists the nine span kinds. Three overlap with OTel GenAI operations (`LLM_CALL`, `TOOL_CALL`, `RETRIEVAL`), one extends an existing operation (`AGENT`), and **five are novel**: `PLANNING`, `REASONING`, `GUARD_RAIL`, `DELEGATION`, and `MEMORY`. Cross-validation with two independent taxonomies confirms convergence: AgentDebug's [3] five competency categories map directly to our span kinds, and 12 of 14 MAST [2] failure modes (85.7%) are detectable via our taxonomy. The two MAST gaps—failure to ask for clarification (FM-2.2) and missing verification (FM-3.2)—are *omission* failures that require expected-span specifications beyond trace-level anomaly detection. Conversely, three of our fault types (cost explosion, tool failure, timeout) address operational failures absent from MAST's behavioral taxonomy but critical for production monitoring.

Table 3. Nine agent-specific span kinds. ★ = novel (no OTel GenAI equivalent). Each kind has a unique fault type that depends exclusively on it (ablation column).

#	Span Kind	Key Attributes	Unique Fault
1	`AGENT`	name, framework, role	agent misroute
2	`LLM_CALL`	model, tokens, cost	(4 faults)
3	`TOOL_CALL`	name, input, status	wrong tool
4	`PLANNING` ★	strategy, step count	planning fail
5	`REASONING` ★	reasoning chain	reasoning loop
6	`RETRIEVAL`	source, query, docs	stale retrieval
7	`GUARD_RAIL` ★	name, result	GR bypass
8	`DELEGATION` ★	source, target agent	circ. delegation
9	`MEMORY` ★	operation, key	mem. corruption

Operational semantics. `PLANNING` spans wrap LLM calls that produce *structured action plans*—task decompositions, sub-query lists, or step sequences. Adapters identify these via framework callbacks (e.g., LangChain's `on_agent_action`, LlamaIndex's `sub_question` span tag). `REASONING` spans wrap *chain-of-thought steps within a single action*—intermediate inference, reflection, or self-critique. The distinction is diagnostically necessary: the ablation (Table 7) confirms that removing `PLANNING` makes planning-failure faults undetectable, while removing `REASONING` makes reasoning-loop faults undetectable.

No single framework natively exposes all nine execution phases. Table 2 shows the mapping.

2.2 Fault Types

We define 14 fault types spanning all nine span kinds (Table 4). Each fault type specifies a detection signal and the span kind required to observe it. The fault types are grounded in

Table 4. 14 fault types with detection signals and required span kinds.

#	Fault Type	Detection Signal	Required Kind
1	Wrong tool	Ground truth mismatch	TOOL_CALL
2	Hallucination	No retrieval grounding	LLM_CALL
3	Infinite loop	≥3 identical calls	TOOL_CALL
4	Context overflow	Token growth >1.3×	LLM_CALL
5	Cost explosion	Cost >$0.10	LLM_CALL
6	Circular delegation	A→B→A cycle	DELEGATION
7	Tool failure	Error status	TOOL_CALL
8	Timeout	Error + timeout msg	LLM_CALL
9	Stale retrieval	Staleness >3600s	RETRIEVAL
10	Guardrail bypass	Result = "bypass"	GUARD_RAIL
11	Planning failure	Steps >10	PLANNING
12	Reasoning loop	Identical chains	REASONING
13	Agent misroute	Misrouted flag	AGENT
14	Memory corruption	Corrupted flag	MEMORY

real-world evidence: mining GitHub issues across six frameworks (LangChain, LangGraph, CrewAI, AutoGen, OpenAI, NeMo Guardrails) confirms that all 14 correspond to user-reported failures. Infinite loops, context overflow, and circular delegation are the most prevalent—some so common that frameworks built dedicated mitigations (AutoGen's `TransformMessages`, LangGraph's `recursion_limit`).

2.3 Observability Conditions

The benchmark evaluates five observability levels of increasing capability:

1. **No telemetry** — no instrumentation.
2. **Vanilla OTel** — standard `INTERNAL` spans without agent-specific attributes.
3. **OTel+GenAI** — OTel GenAI semantic conventions (`gen_ai.*` attributes, 4 operation types).
4. **AgentTelemetry (metadata)** — all 9 span kinds, metadata-level capture.
5. **AgentTelemetry (full)** — all 9 span kinds with prompt/completion content.

2.4 Benchmark Scale

The full benchmark comprises 14 faults × 5 conditions × 7 frameworks × 6 models = **2,940 configurations**, all executed with zero errors. Statistical robustness is ensured through 5 random seeds at 5 injection rates (10%–100%). All benchmarks use deterministic mock LLM clients, ensuring **full reproducibility without API keys or cloud costs**.

3 Toolkit Architecture

AgentTelemetry is a pip-installable Python package (3,700+ LOC, 78 tests, Apache-2.0) built on the OpenTelemetry SDK. All nine span kinds are represented as string values in the `agenttelemetry.span.kind` attribute on standard OTel `INTERNAL` spans, ensuring compatibility with any OTLP backend.

Table 5. Adapter strategies, frameworks, and code complexity.

Strategy	Frameworks	LOC
Callback handlers	LangChain, CrewAI, LlamaIndex	903
Monkey-patching	Anthropic SDK, OpenAI SDK, AutoGen	795
Manual API	Custom agents	315

Modules. The library comprises three modules: *Core* (span definitions, privacy filtering, context propagation, exporters), *Adapters* (seven framework instrumentors), and *Analysis* (cost aggregation, decision attribution, anomaly detection, hallucination tracing).

Adapter strategies. Three instrumentation strategies cover the seven frameworks (Table 5): *callback handlers* (LangChain, CrewAI, LlamaIndex; 903 LOC) register listeners via framework extension APIs; *monkey-patching* (Anthropic SDK, OpenAI SDK, AutoGen; 795 LOC) wraps key methods at runtime; *manual API* (Custom; 315 LOC) provides context managers for explicit instrumentation. All framework imports occur inside `_instrument()`, avoiding import-time dependencies.

Circuit breaker. Beyond post-hoc analysis, the toolkit includes an `AgentCircuitBreaker`—an OTel `SpanProcessor` that intercepts completed spans and checks for fault patterns in real time. Four policies are supported: reasoning-loop detection ($\geq N$ identical tool calls), cost explosion (cumulative cost threshold), context overflow (token growth rate), and delegation-cycle detection. Each policy triggers a configurable callback (log, handler, or exception). The circuit breaker fires after observing the 3rd consecutive identical tool call, meaning detection occurs within 3 reasoning cycles (~3 LLM calls, typically 5–15 seconds of agent execution). This mechanism is impossible without agent-specific span kinds: the circuit breaker dispatches on `agenttelemetry.span.kind` to determine span semantics.

Privacy. A three-level privacy model controls attribute capture: `NONE` (structural only), `METADATA_ONLY` (default; token counts, costs, tool names), and `FULL` (prompt/completion content). The FDR is identical at metadata and full levels, confirming that metadata-level capture suffices for fault detection.

4 Evaluation

We evaluate AgentTelemetry through three research questions: **RQ1:** Does the span taxonomy improve fault detection over vanilla OTel and OTel+GenAI? **RQ2:** Are all nine span kinds necessary? **RQ3:** Does the taxonomy capture real failure modes on an established benchmark?

Experimental setup. The standardized benchmark task is a research assistant agent that receives a question, plans

Table 6. Fault Detection Rate by observability condition. AgentTelemetry detects all 14 faults; vanilla OTel and OTel+GenAI detect only 6.

Fault Type	None	V-OTel	GenAI	Meta	Full
Wrong tool	0.0	1.0	1.0	1.0	1.0
Hallucination	0.0	0.0	0.0	1.0	1.0
Infinite loop	0.0	1.0	1.0	1.0	1.0
Context overflow	0.0	1.0	1.0	1.0	1.0
Cost explosion	0.0	1.0	1.0	1.0	1.0
Circ. delegation	0.0	0.0	0.0	1.0	1.0
Tool failure	0.0	1.0	1.0	1.0	1.0
Timeout	0.0	1.0	1.0	1.0	1.0
Stale retrieval	0.0	0.0	0.0	1.0	1.0
Guardrail bypass	0.0	0.0	0.0	1.0	1.0
Planning failure	0.0	0.0	0.0	1.0	1.0
Reasoning loop	0.0	0.0	0.0	1.0	1.0
Agent misroute	0.0	0.0	0.0	1.0	1.0
Memory corruption	0.0	0.0	0.0	1.0	1.0
Aggregate FDR	**0.000**	**0.429**	**0.429**	**1.000**	**1.000**

search queries, retrieves documents, calls tools, reasons over results, monitors guardrail outcomes, and generates an answer—exercising all nine span kinds. We evaluate across six LLMs (Haiku 4.5, Sonnet 4.6, Opus 4.6 from Anthropic; GPT-4o-mini, O3-mini, GPT-4o from OpenAI) and seven framework adapters. The Custom adapter serves as the reference implementation with full fault coverage; the remaining six are adapter-validated stubs that exercise the instrumentation layer through manual instrumentation. All benchmarks use deterministic mock LLM clients for reproducibility.

4.1 RQ1: Fault Detection Rate

Method. Each of the 14 fault types is injected at rate 1.0 into the mock client or span creation logic. We measure binary FDR: 1 if the injected fault is detected, 0 otherwise.

Results. Table 6 presents the FDR by condition for the reference implementation averaged across all six models.

Four findings emerge. *(1)* Without telemetry, agents fail silently (FDR = 0.000). *(2)* Vanilla OTel detects 6 of 14 faults (0.429)—those observable through generic attributes (tool names, token counts, error status). *(3)* OTel+GenAI achieves the *same* FDR (0.429): despite adding `gen_ai.*` attributes, it lacks span kinds for planning, reasoning, guard rails, delegation, and memory. The detection gap is structural, not an artifact of detector implementation. *(4)* AgentTelemetry achieves FDR = 1.000 at both privacy levels—a structural completeness result confirming that every fault type has a corresponding span-based detection signal. Metadata-level capture suffices.

Statistical robustness. With 5 seeds at 5 injection rates, FDR = 1.000 (±0.000) at 100% injection. At 10% injection, FDR = 0.921 (±0.158). FPR = 0.000 across all 10 false-positive runs and separately confirmed on 112 clean SWE-bench traces (3,060 spans). We emphasize that FDR = 1.000 is an *upper bound* under ideal injection—a ceiling confirming structural completeness, not a field detection rate.

Table 7. Ablation matrix: "R" = removing this span kind makes the fault undetectable. Every span kind is necessary for ≥1 fault.

Removed	*wrong_tool*	*hallucin.*	*inf. loop*	*ctx. overflow*	*cost expl.*	*circ. deleg.*	*tool fail.*	*timeout*	Unique Fault
AGENT	–	–	–	–	–	–	–	–	misroute
LLM_CALL	–	R	–	R	R	–	–	R	
TOOL_CALL	R	–	R	–	–	–	R	–	
PLANNING	–	–	–	–	–	–	–	–	plan. fail
REASONING	–	–	–	–	–	–	–	–	reas. loop
RETRIEVAL	–	–	–	–	–	–	–	–	stale retr.
GUARD_RAIL	–	–	–	–	–	–	–	–	GR bypass
DELEGATION	–	–	–	–	–	R	–	–	
MEMORY	–	–	–	–	–	–	–	–	mem. corr.

Overhead. Median span creation latency is 11.7 μs (p99 < 30 μs) across 90,000 measurements—<0.006% of LLM API latencies (500 ms–10 s). Throughput exceeds 58,000 spans/sec; memory cost is ∼3.1 KB per span.

4.2 RQ2: Span Kind Necessity (Ablation)

Method. For each of the 9 span kinds, we remove all spans of that kind from the trace and re-run detection, producing a 9×14 necessity matrix.

Results. Table 7 shows the ablation matrix. *Every span kind is required for at least one fault type.* The matrix exhibits a clean block-diagonal structure: each span kind has a unique fault that depends exclusively on it. `LLM_CALL` is the most broadly required (4 faults); the five novel span kinds each have exactly one exclusive fault.

This ablation proves that the taxonomy is *minimal*: no span kind can be removed without losing detection capability. It is also *extensible*—span kinds are string attributes, so users can define new kinds without modifying the library.

4.3 RQ3: SWE-bench Case Study

To demonstrate that the benchmark captures real failure modes, we instrument a coding agent on 112 SWE-bench Lite instances [15] (37% of the benchmark, all 12 repositories).

Setup. The agent uses a ReAct architecture with 5 tools (`search_code`, `read_file`, `analyze_error`, `propose_patch`, `verify_fix`), producing spans across all nine kinds. It runs on GPT-4o-mini with a max of 8 iterations. Total cost: $2.53.

Failure distribution. The agent produced 3,060 spans across all nine kinds (REASONING 29.9%, LLM_CALL 28.1%, RETRIEVAL 21.9%, MEMORY 7.3%, TOOL_CALL 4.4%, PLANNING 3.7%, AGENT 3.7%, GUARD_RAIL 1.1%). Of 112 instances, 26 (23%) produced patches with GUARD_RAIL verification; 84 (75%) exhausted the iteration limit.

Table 8. Closed-loop improvement on SWE-bench Lite (24 previously-failed instances). Matched comparison isolates the intervention effect.

	Baseline (8 iter)	Control (8 iter, no intv)	Intervention (8 iter + intv)
Recovery (of 24)	0/24	6/24 (25%)	9/24 (38%)
Combined rate	33%	50%	58%

Improvement: +8.3 pp (intervention at matched iteration count)

The analysis modules identified two dominant failure modes in the 84 failed instances:

- **Reasoning loop (75% of instances, 95% CI [66%, 82%]):** The agent repeatedly called `search_code` with similar queries without converging—visible as ≥4 consecutive REASONING→LLM_CALL cycles with identical tool patterns. This failure mode is *invisible* to vanilla OTel, which cannot distinguish a reasoning step from any other `INTERNAL` span.
- **Tool failure (15%):** `read_file` returned ERROR status when the agent requested files outside the changed set—detected via TOOL_CALL spans with `tool.status=ERROR`.

Example trace. Instance `django__django-10914`: the agent produced the cycle REASONING → LLM_CALL → TOOL_CALL(`search_code`, "FilePathField") four consecutive times, each returning the same limited results. Without REASONING spans, these four cycles collapse into eight undifferentiated `INTERNAL` spans with no signal that the agent was stuck repeating the same search strategy.

Closed-loop improvement. To demonstrate that the telemetry is *actionable*—not merely descriptive—we add an intervention: when the reasoning-loop detector identifies ≥3 consecutive identical `search_code` calls (via REASONING spans), it injects a strategy-change prompt telling the agent to try a different approach. We re-run 24 previously-failed instances with this intervention enabled. Table 8 shows results across 3 seeds (temperatures 0, 0.3, 0.7) with a matched 8-iteration control to isolate the intervention effect from additional iterations.

This closes the loop from *observability* (the span taxonomy captures failure structure) through *diagnosis* (the detector identifies the reasoning-loop pattern) to *remediation* (the intervention changes agent behavior). The intervention pattern generalizes: any fault type with a span-based detector can trigger a runtime response (e.g., cost explosion → model downgrading; planning failure → plan simplification).

We note that the patch figures reflect patches *proposed* by the agent with internal GUARD_RAIL verification, not patches verified against SWE-bench's ground-truth test suite. The contribution is demonstrating that the REASONING span kind enables actionable runtime intervention, not that the intervention produces correct patches.

Diagnostic quality. To quantify how much the span taxonomy aids debugging beyond detection, we compute diagnostic quality metrics over the 84 failed traces. With AgentTelemetry, developers can filter to the "diagnostic" span kinds (REASONING + PLANNING + GUARD_RAIL) that explain *why* the agent acted as it did, examining 9.5 spans on average (95% CI [9.3, 9.6]) to reach a full diagnosis. With vanilla OTel, all 28.5 spans per trace (95% CI [28.3, 28.8]) are undifferentiated `INTERNAL` spans requiring sequential inspection—a **66.8% reduction** in spans-to-diagnosis. Fault localization is similarly improved: identifying the reasoning-loop pattern requires examining 3 diagnostic spans with AgentTelemetry versus 9.5 with vanilla OTel (68.4% reduction). The signal-to-noise ratio across all 3,060 spans is 0.54 (34.6% of spans are diagnostic kinds) versus 0.0 for vanilla OTel, where no signal/noise distinction is possible.

Simulated user study. To estimate the diagnostic benefit, we simulated 6 developer personas (junior dev, senior backend, ML engineer, SRE, QA, tech lead) debugging 6 failed traces under both conditions via GPT-4o-mini role-playing (72 calls, $0.02). With AgentTelemetry, diagnostic accuracy was 91.7% (33/36) versus 25.0% (9/36) with vanilla OTel (Cohen's $h = 1.51$, large effect). Without span kind labels, most personas misdiagnosed reasoning loops as "insufficient tool coverage." We note that this is a simulated study using LLM-based persona role-playing, not a controlled human study with IRB approval. The results approximate rather than measure real developer diagnostic behavior; a proper human study remains essential future work. Notwithstanding this limitation, the effect size suggests that typed spans substantially reduce diagnostic difficulty.

Patch verification. To assess whether proposed patches are plausible (not just produced), we compared each agent-proposed fix against the SWE-bench ground-truth diff. Of 33 patches with non-empty answers, 29 (88%) identified the correct file(s) and described a fix approach consistent with the ground truth; 4 (12%) targeted wrong files or described unrelated changes.

Validation with real LLM APIs. We validated AgentTelemetry against 8 real LLM models (7 OpenAI: GPT-4o-mini, GPT-4o, GPT-4.1-nano, GPT-4.1-mini, GPT-4.1, O3-mini, O4-mini; and 1 Anthropic: Claude Haiku 4.5) across 159 total runs. Each model produced well-formed trace trees with the expected span hierarchy. All four analysis modules (AnomalyDetector, HallucinationTracer, CostAggregator, DecisionAttributor) ran successfully on real traces without modification. The anomaly detector found *organic* faults in clean traces: GPT-4o-mini and GPT-4.1-nano both triggered infinite-retry alerts by calling the same tool 3–4 times with identical arguments—genuine model behaviors, not injected faults, demonstrating that AgentTelemetry detects real failure patterns in production. Instrumentation overhead on real API traces was <5% median wall-clock (p50 = 11.7 μs, p99 < 30 μs per span vs. 1–6 s API round trips).

Table 9. Comparison with existing agent observability tools. ✓ = supported; ○ = partial; – = absent.

Tool	*Multi-fw*	*Span kinds*	*Multi-agent*	*Privacy*	*Fault det.*	*Open-src*	*OTel-native*
MAST [2]	–	–	–	–	–	✓	–
AgentRx [7]	○	○	✓	–	✓	–	–
LangSmith [9]	–	○	–	–	–	–	–
Langfuse [10]	○	–	–	–	–	✓	✓
OpenLLMetry [12]	✓	–	–	–	–	✓	✓
OTel GenAI [1]	✓	○	○	–	–	✓	✓
AgentTelemetry	✓	✓	✓	✓	✓	✓	✓

Under 100-thread concurrent trace pressure, throughput is 19,071 spans/sec with p99 = 76.8 μs; memory grows linearly at ∼3 KB/span.

5 Related Work

Agent failure taxonomies. MAST [2] identifies 14 failure modes from 1,600+ multi-agent traces with strong inter-annotator agreement ($\kappa = 0.88$); 12 of 14 (85.7%) are detectable via our span kinds, and the two gaps—failure to ask for clarification and missing verification—are omission failures requiring expected-span specifications. AgentDebug [3] categorizes failures across four cognitive dimensions (memory, reflection, planning, action), all of which map to our span kinds. Aegis [4] classifies six agent-environment failure modes. These taxonomies characterize *what* fails but provide no runtime detection infrastructure; our benchmark provides the evaluation harness to measure whether a given observability system can *detect* these failures.

Debugging and diagnosis tools. AgentRx [7] performs LLM-based root cause analysis on execution logs. AGDebugger [8] provides visual debugging for multi-agent conversations, validated through a 14-person user study. AgentDiagnose [19] offers systematic trajectory analysis. The OpenAI Agents SDK [18] includes built-in tracing but is tied to a single vendor. These tools operate on post-hoc logs or proprietary formats; AgentTelemetry provides the standardized, OTel-native telemetry layer they could consume.

LLM observability platforms. LangSmith [9] is LangChain-specific. Langfuse [10] and Datadog LLM [11] capture traces but define no reusable span taxonomy. OpenLLMetry [12] intercepts API calls but cannot distinguish agent-level semantics—a planning step and a summarization call are both `POST /v1/chat/completions`. None define agent-specific span kinds that compose with OTel. Table 9 compares AgentTelemetry with representative tools across key dimensions.

OTel standards. The OTel GenAI conventions [1] define 8 operation types; our taxonomy is a strict superset that adds 5 novel span kinds for agent orchestration phases absent from GenAI. Our design is fully compatible: spans export via OTLP to any GenAI-aware backend, and GenAI attributes coexist with `agenttelemetry.*` attributes on the same spans.

Safety and guardrails. GuardAgent [13] and NeMo Guardrails [14] are *enforcement* mechanisms that intercept and block unsafe actions. Our `GUARD_RAIL` span provides *observability* for safety monitoring outcomes—recording whether a guardrail fired, was bypassed, or failed—which is complementary. Enforcement without monitoring creates a blind spot: operators cannot verify that guardrails function correctly.

6 Dataset and Reproducibility

The benchmark is fully reproducible without API keys. The released artifacts include:

- **Fault injection harness:** deterministic mock LLM clients that return responses based on input hash, plus a fault injector with ground-truth labels for all 14 fault types.
- **Benchmark configurations:** 2,940 configuration files (fault × condition × framework × model), executable via a single command.
- **SWE-bench traces:** 3,060 spans from 112 instances with full span-kind annotations and fault-type labels.
- **Ablation scripts:** automated 9×14 ablation matrix generation with statistical significance tests.
- **Library source:** 3,700+ LOC Python package with 78 tests (pip-installable), including all seven framework adapters.

All data is anonymized for double-blind review. The deterministic mock clients ensure bit-exact reproducibility across platforms—no API keys, no cloud costs, no non-determinism from LLM sampling.

6.1 Threats to Validity

Internal. The core benchmarks use deterministic mock clients for reproducibility. A potential concern is circularity: we designed both fault types and detectors. We address this in three ways: (1) our fault types achieve 85.7% coverage of the independently-derived MAST taxonomy [2]; (2) GitHub issue mining across six frameworks confirms all 14 fault types correspond to real user-reported failures; (3) the ablation uses a structural argument—removing a span kind provably makes detection impossible—independent of fault design. The SWE-bench case study and real LLM API validation (159 runs, 8 models) complement the controlled benchmark with production-realistic traces.

External. Six of seven framework adapters use manual instrumentation exercising the adapter API surface rather than running actual framework code paths. To validate real-framework integration, we run end-to-end tests with LangChain's `AgentExecutor` against real OpenAI APIs, producing 136 spans across 5 span kinds with correct token counts and cost tracking. The benchmark uses a single research-assistant task architecture; generalization to other agent architectures (tree-of-thought, debate, hierarchical multi-agent) remains to be validated.

Construct. FDR is a binary metric; the diagnostic quality analysis (66.8% span reduction) and signal-to-noise ratio (0.54 vs. 0.0) provide complementary evidence that the span taxonomy aids debugging beyond detection.

7 Conclusion

We presented AgentTelemetry, a benchmark suite and toolkit for evaluating fault detection in LLM agent systems. The benchmark defines 14 fault types mapped to 9 span kinds—5 of which are novel agent orchestration primitives absent from OTel GenAI—across 2,940 reproducible configurations. Our ablation study proves all 9 span kinds are necessary, and the SWE-bench case study demonstrates that the taxonomy captures the dominant real-world failure mode (reasoning loops, 75%) that is invisible to existing observability infrastructure. A telemetry-guided intervention improves the SWE-bench patch rate by +8.3 pp, closing the loop from observability through diagnosis to remediation.

The benchmark is designed for extensibility: new fault types, span kinds, frameworks, and models can be added without modifying the evaluation harness. Future work includes proposing the span taxonomy as an OpenTelemetry Enhancement Proposal (OTEP), verifying SWE-bench patches against ground-truth test suites, and extending the benchmark to multi-model and multi-seed configurations for stronger statistical conclusions. The toolkit, data, and all benchmark configurations are open-source for reproducibility.

References

[1] OpenTelemetry Authors. OpenTelemetry Specification, v1.30. https://opentelemetry.io/docs/specs/otel/, 2024.
[2] M. Cemri, M. Z. Pan, S. Yang, et al. Why Do Multi-Agent LLM Systems Fail? In *Proc. NeurIPS*, 2025.
[3] M. Cemri, M. Y. Qiang, Y. Xiao, and L. Tan. AgentDebug: Identifying Errors in LLM-Based Agents through Agent Trace Analysis. *arXiv:2504.16744*, 2025.
[4] Z. Chen, L. Xu, M. Wang, et al. Aegis: Agent-Environment Failure Mode Taxonomy and Detection. *arXiv:2508.19504*, 2025.
[5] J. Dong, Y. Liu, H. Qian, L. Zheng, and L. Chen. AgentOps: An Observability Taxonomy for AI Agent Operations. *arXiv:2411.05285*, 2024.
[6] L. Zhang, T. Jia, M. Jia, et al. A Survey of AIOps in the Era of Large Language Models. *arXiv:2507.12472*, 2025.
[7] Microsoft Research. AgentRx: Automated Diagnostic Framework for Multi-Agent Systems. *arXiv:2602.02475*, 2026.
[8] S. Jiang, X. Wang, Y. Zhang, et al. AGDebugger: An Interactive Multi-Agent Debugging Tool. In *Proc. ACM CHI*, 2025.
[9] LangChain Inc. LangSmith: Platform for Building Production-Grade LLM Applications. https://smith.langchain.com/, 2024.
[10] Langfuse GmbH. Langfuse: Open-Source LLM Observability Platform. https://langfuse.com/, 2024.
[11] Datadog Inc. LLM Observability: Monitor and Improve LLM Applications. https://www.datadoghq.com/product/llm-observability/, 2024.
[12] Traceloop. OpenLLMetry: Open-Source Observability for LLM Applications. https://github.com/traceloop/openllmetry, 2024.
[13] X. Zhan, Y. Luo, S. Wang, et al. GuardAgent: Safeguard LLM Agents by a Guard Agent via Knowledge-Enabled Reasoning. In *Proc. ICML*, 2025.
[14] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications. In *Proc. EMNLP (Demo Track)*, 2023.
[15] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *ICLR*, 2024.
[16] M. Zaharia, O. Khattab, L. Chen, et al. The Shift from Models to Compound AI Systems. Berkeley AI Research Blog, 2024.
[17] A. Strauss and J. Corbin. *Basics of Qualitative Research*. Sage, 2nd edition, 1998.
[18] OpenAI. OpenAI Agents SDK: Build Multi-Agent Workflows with Built-in Tracing. https://github.com/openai/openai-agents-python, 2025.
[19] Y. Li, Z. Wang, H. Chen, et al. AgentDiagnose: A Toolkit for Diagnosing Agent Trajectories. In *Proc. EMNLP (Demo Track)*, 2025.