Executable Ground Truth: A Closed-Loop Benchmark for Evaluating LLM Agents on Microservice Incident Remediation

Dhatri C [1] **Tadisetty Sai Yashwanth** [1]

Abstract

Most LLM agent benchmarks score the agent's output description, not its effect on the world. We introduce **TraceRoot**, a microservice incident benchmark where an agent investigates structured logs, edits the faulty source file, and has its patch verified by an end-to-end reproducer with no LLM judge. Across six frontier models and five incidents, three reasoning-enabled models achieve 5/5 pass rates; three non-reasoning models score 1/5, failing primarily through non-termination, a failure mode invisible to report-based evaluators. We release all artifacts to support reproducible evaluation.

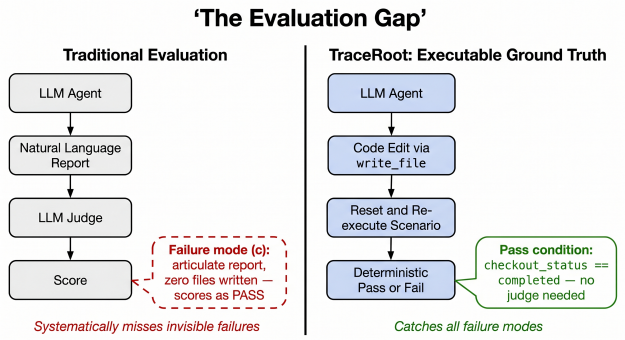

Figure 1. The two evaluation pipelines. **Left:** traditional evaluation delegates scoring to an LLM judge, which cannot detect failure mode (c). **Right:** TraceRoot uses a deterministic reproducer. The pass condition is a property of the running system, not the agent's report.

1. Introduction

A growing literature evaluates LLM agents on tasks requiring multi-step reasoning, tool use, and planning. Incident response is a canonical example: the agent must gather evidence from logs, localize a fault, apply a fix, and verify the result. Most existing benchmarks stop at *diagnosis*, measuring whether the model produces a plausible natural-language account of the root cause. Scoring is then delegated to an LLM judge or keyword match against a gold summary sentence.

This creates a systematic evaluation gap with two components.

The description-vs.-execution gap. A model can write a persuasive explanation of a fix that, when applied, still fails. Prose can be correct in intent and incorrect in code simultaneously. Only running the repaired system reveals whether the fix works.

The non-termination blind spot. An agent that queries logs indefinitely without committing to a fix receives the same "no answer" penalty as one that answers incorrectly. In practice the two failure modes have very different production implications and are caused by different underlying deficiencies. LLM-judged benchmarks conflate them.

We address both with a benchmark in which the scoring function is the reproducer itself. After the agent edits the source code, the scenario is re-executed end-to-end. The pass condition is a property of the *running system*, not a property of the agent's report. This design principle, **executable ground truth**, is the central methodological contribution of this paper. Figure 1 contrasts the two evaluation pipelines.

Our secondary contribution is a concrete empirical demonstration of what the evaluation gap hides: when scoring moves from prose to execution, the apparent capability of non-reasoning models collapses from plausible-sounding diagnoses to 1/5 pass rates, while the dominant failure mode is non-termination rather than incorrect patching.

2. TraceRoot: Benchmark Design

2.1. Design Principles for Evaluable Benchmarks

We identify four properties a benchmark must have to support reliable, structured empirical evaluation of tool-using

[1]Turilabs, Bangalore, India. Correspondence to: Dhatri C <dhatric@turilabs.in>, Tadisetty Sai Yashwanth <sai@turilabs.in>.

Accepted to the Workshop on Combining Theory and Benchmarks: Towards A Virtuous Cycle to Understand and Guarantee Foundation Model Performance at ICML 2026, Seoul, South Korea.
Copyright 2026 by the author(s).

agents.

1. **Determinism.** Every incident must be triggered by a named flag and produce a stable, reproducible log corpus. Non-determinism in the environment makes results model-confounded rather than just model-dependent.
2. **Machine-verifiable pass conditions.** The pass condition must be computable without a judge. In TraceRoot, each condition is a specific property of the system state after execution (e.g., checkoutstatus == "completed").
3. **Decomposable capability structure.** The task should require a sequence of distinct sub-capabilities, each observable in the tool trace, so that failure can be localized to a specific stage rather than attributed globally to "the model."
4. **Calibrated difficulty with misleading signals.** Every incident contains at least one symptom pointing to the wrong service. This reflects realistic incident data and prevents shallow pattern-matching from mimicking genuine causal reasoning.

2.2. System

Five Python/FastAPI services form a checkout system: `api-gateway`, `order-service`, `inventory-service`, `payment-service`, and `reconciliation-worker`. A `log-manager` ingests structured JSON logs from all components into SQLite. The agent accesses logs through seven constrained query tools and edits source through three file tools (`list_service_files`, `read_file`, `write_file`). Writes are path-validated to service directories only.

2.3. Incident Catalog

Five incidents span five distinct reasoning sub-capabilities (Table 1).

Table 1. TraceRoot incident catalog.

ID	Root Cause	Capability Tested
I1	`api-gateway`	Cross-service correlation
I2	`order-service`	Compensation; temporal displacement
I3	`order-service`	Idempotency key lifecycle
I4	`reconciliation-worker`	Boundary condition / off-by-one
I5	`order-service`	Contradictory cross-service state

Each incident is specified in a YAML ground-truth file with `root_cause_service`, `expected_log_signals`, `misleading_signals`, `required_reasoning_steps`, and `expected_remediation`. Ground truth is fully machine-readable; no free-text evaluation is required at any stage.

2.4. Verification Protocol

Before every model run, the harness resets the codebase to the buggy baseline via `git checkout`. After the agent terminates, `verify_scenario.py` executes in a fresh temporary database, constructs the scenario-specific checkout request, and checks the deterministic pass condition. All five incidents fail on the unmodified codebase; any `passed=True` in a subsequent run is attributable solely to the agent's edits. All artifacts (prompt, tool trace, final report, verification result) are persisted for audit.

3. Experiments

3.1. Setup

Six models were evaluated (max 15 tool-call iterations per run, 30 total runs): `gpt-5.4`, `gpt-5`, `gpt-4.1` (OpenAI); `grok-4.20-0309-reasoning`, `grok-4-1-fast-reasoning`, `grok-4-1-fast-non-reasoning` (xAI). The first three are reasoning-enabled; the latter three are not (or are older non-reasoning variants).

3.2. Main Results

✓ = reproducer passed; × = reproducer failed; ERR = budget exhausted, no final report.

Table 2. Pass/fail matrix across six models and five incidents.

Model	I1	I2	I3	I4	I5	Rate
`gpt-5.4`	✓	✓	✓	✓	✓	**5/5**
`grok-4.20-R`	✓	✓	✓	✓	✓	**5/5**
`grok-4-1-fast-R`	✓	✓	✓	✓	✓	**5/5**
`grok-4-1-fast-NR`	×	×	×(c)	✓	×	1/5
`gpt-4.1`	×	ERR	ERR	×	✓	1/5
`gpt-5`	ERR	ERR	ERR	ERR	✓	1/5

The split is not along vendor or recency lines (`gpt-5`, the most recent OpenAI model evaluated, ties for last). The decisive axis is chain-of-thought reasoning. Figure 2 annotates each failing cell with its failure mode.

3.3. Fine-Grained Failure Taxonomy

Failures decompose into three distinct modes.

(a) Non-termination. `gpt-5` on 4/5 incidents; `gpt-4.1` on 2/5. These agents re-query logs and re-read files until the 15-turn cap is hit, never calling `write_file`. The model lacks a mechanism to assess when evidence is sufficient to commit to an action.

(b) Wrong patch. `gpt-4.1` on I1; `grok-4-1-fast-NR` on I1, I2, I5. The agent writes a file but the edit is incomplete. On I2, the non-reasoning

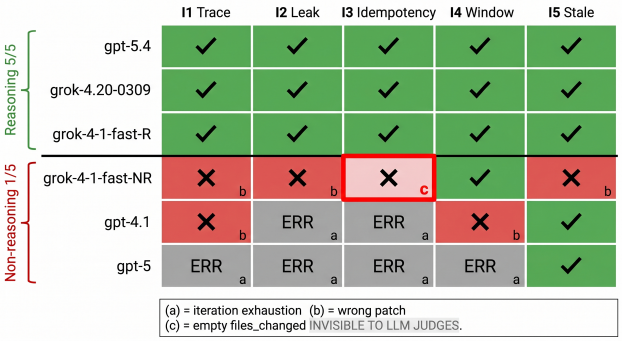

Figure 2. Pass/fail matrix annotated with failure modes. Green (✓) = reproducer passed; red (×) = reproducer failed; gray (ERR) = budget exhausted. The (c) cell (`grok-4-1-fast-NR` on I3) is the most dangerous: an articulate diagnosis report with zero files written, invisible to any report-based evaluator.

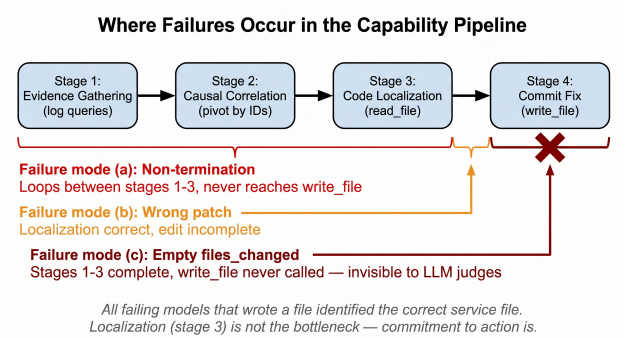

Figure 3. Failure modes localized against the four-stage capability pipeline. All three modes collapse at or before the commit stage (Stage 4). Stages 1–3 (investigation and localization) are not the bottleneck.

grok correctly describes the missing compensation call in its report text, but the emitted file still lacks it. An LLM judge evaluating the report would call this a success.

(c) Empty `files_changed`. `grok-4-1-fast-NR` on I3 (after 694.5s and 35 tool calls): the final JSON report contains `files_changed: []` and no file was written. The agent's diagnosis is articulate; the system is untouched. This is structurally invisible to any evaluation that scores the report rather than the system state.

Failure mode (c) demonstrates concretely why executable ground truth is not merely a quality improvement over LLM judging: it is a different measurement instrument that surfaces categorically different failure evidence.

3.4. Controlled Capability Comparison

The cleanest controlled comparison is `grok-4-1-fast-reasoning` vs. `grok-4-1-fast-non-reasoning`: same base model, same provider, same API surface, differing only in chain-of-thought activation.

Table 3. Reasoning vs. non-reasoning on the same base model.

Variant	Pass Rate	Tool Calls	Latency
Reasoning	**5/5**	21.6	58s
Non-reasoning	1/5	27.8	158s[†]

[†]Inflated by 694.5s stall on I3. On non-stalling runs, non-reasoning averages 24s.

Enabling reasoning simultaneously *reduced* tool use, *eliminated* runaway loops, and *increased* pass rate by 5×. The non-reasoning model uses more tool calls while solving fewer incidents: investigation volume is not a proxy for planning progress.

4. Discussion

4.1. The Evaluation Gap is Quantifiable

In a hypothetical diagnosis-only evaluation, failure mode (c) would be scored as a success, and failure mode (a) would receive a partial penalty at best. Our benchmark allows us to quantify what this gap costs: at least one run per non-reasoning model would be incorrectly classified as a success by a report-based evaluator. More precisely, `grok-4-1-fast-NR` on I3 produces a report that correctly identifies the idempotency regeneration bug and would score well on any rubric, yet the system remains broken.

4.2. Benchmark Design as Hypothesis Infrastructure

The four capability decomposition (evidence gathering → causal correlation → code localization → correct editing) allows us to state and test a precise hypothesis: *non-reasoning models fail at stage 4 (committing a write), not at stages 1–3 (investigation).* The tool trace confirms this: all models that wrote a file identified the correct service file. File localization is not what separates the groups; commitment to a write action is. Figure 3 maps each failure mode to the stage at which it breaks down.

This kind of localized failure attribution is only possible when the benchmark is designed with decomposable capability structure. Benchmark design should anticipate the hypotheses it will be used to test, not just the tasks it will score.

4.3. Limitations

Each incident requires editing one file; real bugs often span multiple files. The tool surface is narrower than Loki or Datadog. Models that detect and exploit the scenario flag string can inflate scores without genuine causal reasoning. Each (model, incident) pair was run once; multi-seed runs

are needed for tighter estimates on borderline incidents.

5. Related Work

RCA benchmarks. OpenRCA (Chen et al., 2025) provides 335 failures from enterprise systems with 68 GB of telemetry; even the best agent solves only 11.34%, and scoring uses LLM evaluation. Our benchmark differs in using machine-verifiable pass conditions and in quantifying failure modes rather than just aggregate scores. RCAgent (Xu et al., 2024) targets industrial deployment rather than controlled evaluation; it does not use a closed-loop verifier.

Software engineering agents. SWE-bench (Jimenez et al., 2024) is the closest antecedent in spirit: patches are verified against repository tests, not by an LLM. Our benchmark differs in input modality (structured telemetry vs. issue text) and in the deliberate design of misleading signals requiring multi-hop causal reasoning.

Evaluation methodology. The critique of LLM-as-judge evaluation has been raised in generative NLP contexts (Liu et al., 2023); we provide a concrete instantiation in the agentic tool-use setting and demonstrate a specific class of failure (mode (c)) that is structurally undetectable by judge-based methods. Chain-of-thought as an enabling capability for agent planning is established in Wei et al. (2022) and Yao et al. (2023).

6. Conclusion

We introduced TraceRoot and a fix-benchmark harness with executable ground truth: success is defined by running the repaired system, not by evaluating the repair report. The benchmark surfaces a failure mode: authoritative-looking diagnostic output with no actual system change, which report-based evaluation cannot detect. Across 30 runs, chain-of-thought reasoning was the single strongest predictor of benchmark success, with the controlled within-family comparison showing a 5× pass rate improvement at 2.5× average latency. We release all artifacts to support reproducible evaluation in this setting.

A fix is a fix only when it runs.

References

Chen, Y., Zhong, H., Ma, M., Kang, Y., Qin, X., Rajmohan, S., Zhang, D., et al. OpenRCA: Can large language models locate the root cause of software failures? In *Proceedings of the 13th International Conference on Learning Representations (ICLR 2025)*, 2025.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)*, 2024.

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, 2022.

Xu, J., Su, Y., Chen, X., Li, Z., Weng, T., Zhang, H., Nie, J., Lin, X., et al. RCAgent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM 2024)*, 2024.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*, 2023.