Compositional Investigation: Why Reasoning Enables Tool-Using Agents to Fix What They Diagnose

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

Abstract

We introduce `rca_llms`, a closed-loop benchmark in which an agent must diagnose and *patch* microservice bugs by composing four sub-skills (evidence gathering, cross-service correlation, code localization, and code editing), with correctness verified by a deterministic reproducer rather than an LLM judge. Across six frontier models and five multi-hop incidents, all three reasoning variants achieve 5/5 pass rates while all three non-reasoning variants score 1/5, with most failures being non-terminations. The same base model (`grok-4-1-fast`) scores 5/5 with reasoning and 1/5 without, isolating chain-of-thought as a *compositional controller* that commits to sub-conclusions and advances the pipeline, rather than a simple accuracy booster.

1. Introduction

Root cause analysis (RCA) in a multi-service system is not a single-step inference problem. A production engineer investigating an incident must: (1) gather relevant log signals from multiple services, (2) correlate events across service boundaries using shared entity identifiers, (3) localize the responsible code path, and (4) produce and apply a precise fix. Each step is a distinct sub-skill, and each step's scope is shaped by the conclusions drawn in the previous one. This is the canonical structure of a *compositional task*: complex competence achieved by chaining simpler competences in a goal-directed sequence.

The central question of compositional learning for agents is whether models can systematically chain such sub-goals rather than executing each in isolation or collapsing them into pattern matching. The prevailing evaluation methodology for "AI SRE" systems does not test this: it measures whether a model can produce a plausible diagnostic narrative, scored either by string matching or by another LLM. Narrative production rewards the first two sub-skills in the chain (gather and correlate) but does not require the final two (localize and fix), and it does not verify that the chain was executed in the correct compositional order.

We address this gap with a *fix benchmark*: the agent's tool-calling sequence and code edit are preserved, the patched system is executed end-to-end, and the pass condition is checked programmatically. There is no judge. A correct fix requires composing all four sub-skills without failing at any stage. This design makes compositional failure directly observable: a model that diagnoses correctly but emits an incorrect edit has failed at localization or editing; a model that never emits a final report has failed at *termination*, i.e., at deciding when the chain is complete enough to act.

Our findings reveal a sharp binary: reasoning-enabled models compose the full chain on every incident; non-reasoning models cannot reliably terminate the chain. The gap is not between large and small models or between vendors. The *same* base model, `grok-4-1-fast`, scores 5/5 with reasoning and 1/5 without it. This isolates reasoning as the mechanism that enables compositional execution of multi-step agent tasks.

2. The RCA Playground as a Compositional Task

2.1. System Overview

We construct a minimal but realistic 4-service checkout system: `api-gateway`, `order-service`, `inventory-service`, and `payment-service`, backed by a `reconciliation-worker` for async cleanup. A `log-manager` component ingests structured JSON logs into SQLite and exposes a constrained query surface to the agent.

All services emit structured JSON with shared fields including `timestamp`, `service`, `request_id`, `order_id`, `payment_id`, `idempotency_key`, `scenario_id`, `operation`, `event_type`, `error_code`, and a `metadata` object. The structured log schema enables sys-

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

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. Copyright 2026 by the author(s).

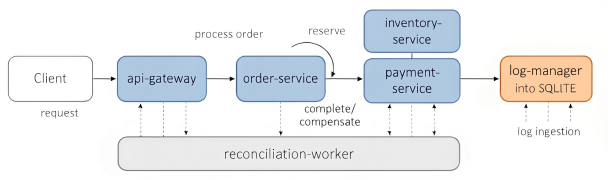


Figure 1. Playground architecture. The synchronous checkout path (solid arrows) is the primary investigation surface. The reconciliation-worker (dashed) introduces async failures tested in I4. All components emit structured JSON logs ingested into SQLite by the log-manager.

tematic correlation across services through shared entity identifiers.

2.2. Incidents as Compositional Test Cases

Five deterministic incidents test the agent's ability to compose distinct investigation sub-skills. Each incident contains at least one misleading clue; the true root cause is always temporally or causally displaced from the visible symptom.

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` fields. Scenarios are activated via explicit flags (`scenario_id` in the request), making every bug deterministic and reproducible.

2.3. The Compositional Task Decomposition

Incident remediation decomposes into four sub-goals that must be executed in sequence:

1. **Evidence gathering:** call log tools with the right service, time window, and entity filter.
2. **Cross-service correlation:** stitch events through shared IDs (`request_id`, `order_id`, `idempotency_key`) to find the true causal chain, not just the symptom.

Table 1. Incident catalog. Each incident targets a distinct compositional sub-skill.

ID	Title	Root cause	Compositional challenge
I1	Missing Trace Prop.	api-gateway	Symptom at payment; cause upstream; pivot required
I2	Inventory Leak	order-service	Cause (missing compensation) precedes symptom
I3	Duplicate Payment	order-service	Idempotency evidence dispersed across retry logs
I4	Window Bug	recon.-worker	Silent failure; no error logs; boundary reasoning
I5	Stale Read	order-service	Contradictory cross-service signals

3. **Code localization:** translate the log pattern to a specific file and function.
4. **Minimal correct editing:** produce a complete, syntactically valid corrected file that fixes the identified bug without regressions.

Sub-goals 1–2 are purely investigative; sub-goals 3–4 require committing to a hypothesis and acting on it. Failure at any sub-goal propagates: a wrong pivot in correlation will localize the wrong code path; a correct localization with an incomplete edit still fails the reproducer.

3. Benchmark Harness

3.1. Tool Surface

In the fix benchmark, the agent receives two tool sets simultaneously.

Log query tools: `search_logs`, `get_logs_by_request_id`, `get_logs_by_order_id`, `get_related_ids`, `count_errors_by_service`, `timeline`, `list_recent_anomalies`.

File tools (path-validated to service directories only): `list_service_files`, `read_file`, `write_file`.

The surface is intentionally narrow: it mirrors the affordances an engineer actually uses on an observability platform, avoiding free-form SQL or unrestricted filesystem access.

3.2. Agent Contract and Budget

The prompt frames the model as a senior engineer fixing a production bug. It provides architecture context, inci-

dent ID, scenario ID, symptom statement, and time window. The agent must investigate with log tools, read relevant source files, apply the fix via `write_file` with complete corrected content, and emit a structured JSON object with `root_cause`, `files_changed`, and `explanation`.

The harness enforces a `max_iterations=15` budget (assistant turns). Exhaustion without a final report is recorded as a run error.

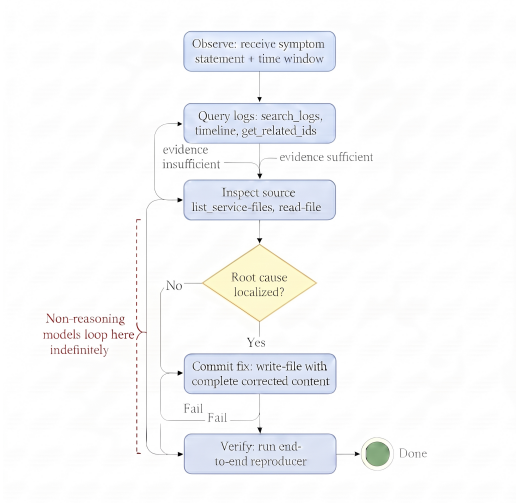


Figure 2. Agent fix loop. Reasoning models traverse this flowchart in a single directed pass; non-reasoning models loop on the evidence-gathering branch indefinitely, exhausting the iteration budget without committing to a fix.

3.3. Deterministic Verification

Before every run, `git checkout -- services/` restores the buggy baseline. After the agent terminates, `evaluation/verify_scenario.py` executes the patched system end-to-end in a fresh database. Pass conditions are: I1, I3, I5: `checkout_status == "completed"`; I2: `reservation_status == "released"` post-checkout; I4: `order_status == "completed"` after `run_once()`.

All five bugs are confirmed present in the unmodified baseline before any model run. A `passed=True` result is exclusively attributable to the agent's edits.

4. Results

4.1. Models

Six models were evaluated across two vendors (Table 2).

4.2. Main Results

The result is a clean binary (Table 3, Figure 3). All reasoning variants pass every incident; all non-reasoning variants score 1/5 with most failures being non-terminations. The split does not follow vendor or model size: `gpt-5`, the most recent OpenAI model in the set, ties for last with `grok-4-1-fast-non-reasoning`. `grok-4-1-fast-reasoning`, a fast-tier model, matches `gpt-5.4`.

Table 2. Models evaluated. All accessed via OpenAI-compatible API.

Label	Provider	Reasoning
`grok-4-1-fast-reasoning`	xAI	✓
`grok-4-1-fast-non-reasoning`	xAI	×
`grok-4.20-reasoning`	xAI	✓
`gpt-4.1`	OpenAI	×
`gpt-5.4`	OpenAI	✓
`gpt-5`	OpenAI	×

Table 3. Pass/fail matrix (6 models × 5 incidents). ✓ = reproducer passed; × = reproducer failed after edit; ERR = budget exhausted without final report.

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`	×	×	ERR	✓	×	1/5
`gpt-4.1`	×	ERR	ERR	×	✓	1/5
`gpt-5`	ERR	ERR	ERR	ERR	✓	1/5

4.3. Within-Family Controlled Comparison

The cleanest experimental contrast is `grok-4-1-fast-reasoning` vs. `grok-4-1-fast-non-reasoning`: same base model, same provider, same API, differing only in chain-of-thought availability.

Table 4. `grok-4-1-fast`: reasoning vs. non-reasoning.

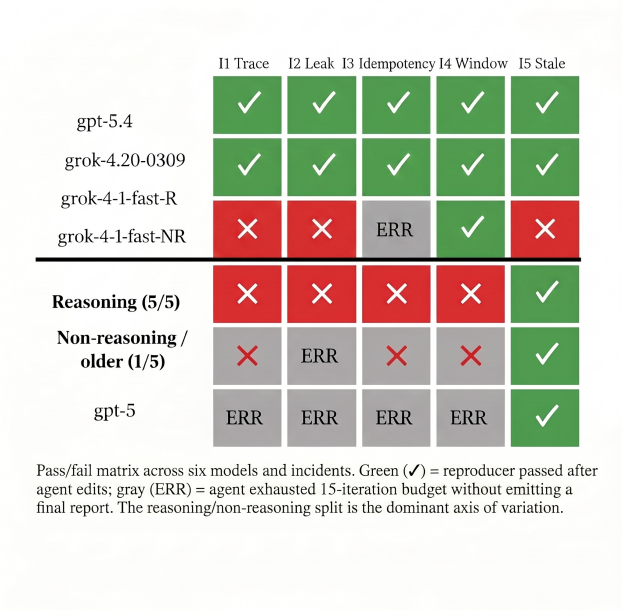
Variant	Pass rate	Avg tools	Avg latency
reasoning	5/5	21.6	≈69 s
non-reasoning	1/5	27.8	≈160 s

Reasoning reduces tool calls by 22% while increasing pass rate by 400% (Table 4, Figure 4). Counterintuitively, non-reasoning models issue *more* tool calls while making *less* investigative progress, consistent with a model that lacks a termination signal and defaults to defensive re-querying.

4.4. Failure Mode Analysis

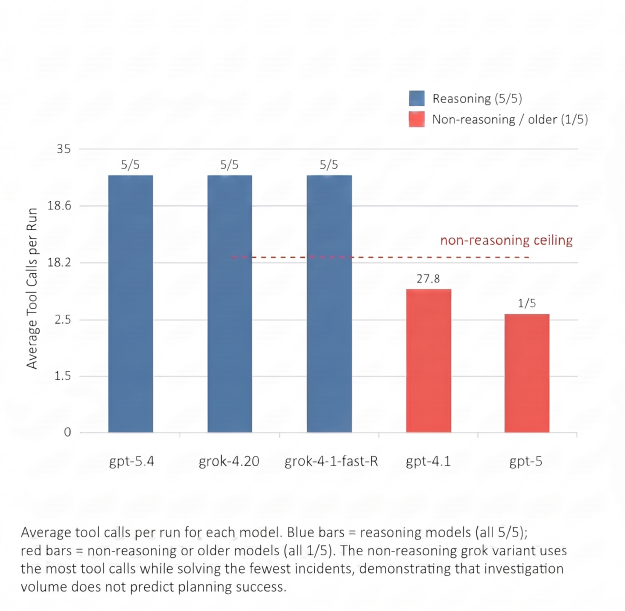
Failures fall into three categories, in decreasing frequency:

(a) Iteration exhaustion without a final report (9/15 failing runs). The agent keeps issuing tool calls (often re-reading the same files or querying the same service) until



Pass/fail matrix across six models and incidents. Green (✓) = reproducer passed after agent edits; gray (ERR) = agent exhausted 15-iteration budget without emitting a final report. The reasoning/non-reasoning split is the dominant axis of variation.

Figure 3. Pass/fail matrix visualised as a heatmap. The reasoning/non-reasoning split is the dominant axis of variation, orthogonal to vendor and model generation.



Average tool calls per run for each model. Blue bars = reasoning models (all 5/5); red bars = non-reasoning or older models (all 1/5). The non-reasoning grok variant uses the most tool calls while solving the fewest incidents, demonstrating that investigation volume does not predict planning success.

Figure 4. Average tool calls per run grouped by reasoning capability. Reasoning models (blue) use fewer tool calls than non-reasoning and older models (red) while achieving 5/5 pass rates. Investigation volume is not a proxy for planning success.

the 15-turn cap is hit. This is a compositional control failure: the model cannot determine when evidence gathering is sufficient to commit to the next sub-goal.

(b) Edit applied but reproducer fails (5/15). The model localizes the correct file and issues `write_file`, but the edit is incomplete. On I2, the non-reasoning Grok variant correctly describes the missing-release bug in prose but emits a file that still lacks the compensation call on the triggering scenario branch. The diagnostic output appears correct; the code is wrong.

(c) Correct identification but empty `files_changed` (1/15). The agent explains the bug accurately in its final report but never writes the file. The worst category: high diagnostic confidence, zero operational effect.

5. Discussion: Reasoning as a Compositional Controller

5.1. Chain-of-Thought as a Sub-Goal Commitment Mechanism

The pattern across failure modes points to a single underlying deficit: non-reasoning models cannot reliably commit to sub-conclusions and advance the compositional chain. They gather evidence but do not transition to localization; they localize but do not commit to an edit; they emit prose but not code. Reasoning-enabled models do the opposite: intermediate chain-of-thought provides an explicit record of what has been established and what remains open, which triggers commitment when the evidence is sufficient.

This is not the standard view of chain-of-thought as accuracy enhancement through more computation. The reasoning/non-reasoning comparison at fixed model size (`grok-4-1-fast`) shows no evidence that non-reasoning models fail *at understanding the problem*; they often produce correct diagnostic narratives. They fail at *acting on that understanding* within the compositional pipeline. Chain-of-thought here plays the role of a planner that tracks sub-goal state, not a calculator that improves individual reasoning steps.

5.2. Termination as Compositional Competence

The evaluation literature typically treats non-termination as an edge case or timeout artifact. We argue it should be treated as a compositional failure mode in its own right. In a compositional task, knowing *when a sub-goal is complete* is itself a competence. A model that cannot determine when evidence gathering is sufficient to proceed to correlation has failed at composing those two sub-goals, regardless of whether it could execute each in isolation.

Failure mode (a) is particularly diagnostic. The failing models are not stuck because they lack relevant information; several correctly identify the relevant service and event within 5–6 tool calls and then keep querying the same information. What they lack is the meta-level signal to transition: "the evidence for hypothesis H is now sufficient; I will act."

5.3. Implications for Compositional Agent Design

Sub-goal state representation matters as much as per-sub-goal capability. Models that execute each investigation step individually may still fail the full chain if they lack a mechanism to track which steps have been completed and at what confidence level. This points toward explicit goal-tracking architectures (hierarchical planning modules, scratchpads with commitment markers, or structured decomposition prompts) as a complement to raw capability.

Closed-loop verification is not optional. Failure mode (b) would be invisible to any evaluation that scores by LLM judgment or text matching. The edited file looks plausible; the reported root cause is correct; only the reproducer reveals that the compensation branch was not fixed on the exact scenario path. For safety-critical settings, narrative-only scoring systematically overestimates agent competence on the task that actually matters: producing a working fix.

Efficiency and reliability are positively correlated. Successful models use fewer tool calls than failing models while achieving higher pass rates. This contradicts the intuition that more investigation should lead to better fixes. It suggests that compositional structure in the investigation reduces redundant exploration: once a sub-goal is established, the model moves on rather than revisiting it.

5.4. Connection to Compositional Generalization

The incidents in this benchmark require systematic compositional generalization: the agent must identify the correct sub-goal *composition order* for a novel incident it has not seen before, using a fixed set of primitive tools. Incidents I4 (async boundary reasoning) and I5 (contradictory cross-service state) require pivoting the investigation in directions not suggested by the initial symptom. The analogy to compositional generalization is that the agent must recombine known sub-skills (time-window reasoning, state-consistency checking) in a novel order appropriate to the incident structure.

The success of reasoning models on these incidents, and the failure modes of non-reasoning models (which often pursue the symptom-indicated service rather than pivoting), is consistent with the hypothesis that chain-of-thought enables productive recomposition: the model records and revises its compositional plan as evidence updates, rather than committing early to a symptom-driven path.

6. Related Work

LLM-based incident diagnosis. Prior work on log-based LLM agents focuses on diagnostic narrative generation, scored by LLM judges or keyword matching against gold summaries (Chen et al., 2024; Shan et al., 2024). These evaluations are insufficient for the fix task: a syntactically correct narrative does not imply a semantically correct code edit.

Software engineering agent benchmarks. SWE-bench (Jimenez et al., 2024) and its descendants evaluate patch correctness on GitHub issues using unit tests. The agent reads issue text and code but not operational logs; incidents are drawn from open-source issue trackers rather than production observability telemetry. `rca_llms` combines log-based investigation with code editing, reproducing the full operational loop that SWE-bench omits.

Compositional reasoning benchmarks. ARC (Chollet, 2019), SCAN (Lake & Baroni, 2018), and CLUTRR (Sinha et al., 2019) test compositional generalization on structured synthetic tasks. Our benchmark extends this evaluation mode to a realistic agentic setting where compositionality is a property of the agent's *action sequence*, not the input/output structure.

Chain-of-thought and reasoning. Wei et al. (2022) and Kojima et al. (2022) establish that chain-of-thought improves multi-step reasoning on math and symbolic tasks. Our results extend this finding to compositional planning over tool-calling traces. The improvement is qualitative (0% $\rightarrow$ 100% within family), not incremental, suggesting a phase transition in compositional task completion rather than a smooth accuracy boost.

7. Conclusion

We have presented `rca_llms`, a benchmark for evaluating tool-using agents on the full incident remediation cycle: from log investigation through code fix to programmatic verification. The benchmark operationalizes incident remediation as a compositional task: four sub-goals that must be chained in order, each contingent on the previous.

The main empirical finding is sharp and clean: reasoning-enabled models compose all four sub-goals on every incident; non-reasoning models of the same size and from the same provider fail 80% of the time, predominantly by failing to terminate the investigation and commit to an edit. We interpret reasoning as a compositional controller that tracks sub-goal state and enables commitment, not merely as an accuracy booster on individual steps.

For the compositional learning community, this result identifies tool-using agent benchmarks as a productive evaluation modality for studying compositional plan execution, one where compositional failure is directly observable in the agent's action trace and where the evaluation signal is a deterministic program rather than a human judgment. The benchmark is fully reproducible and we encourage its extension to larger incident catalogs, multi-file fixes, and

scenario-free variants.

For practitioners, the recommendation is equally direct: for tool-using incident agents, choose a reasoning model and close the loop with a programmatic verifier. Diagnosis is not a fix. A fix is a fix only when it runs.

References

Chen, Y. et al. RCA-Eval: Benchmarking large language models for root cause analysis of microservice incidents. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, 2024.

Chollet, F. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

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? *arXiv preprint arXiv:2310.06770*, 2024.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35, 2022.

Lake, B. M. and Baroni, M. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 2879–2888, 2018.

Shan, J. et al. Log-based anomaly detection with large language models. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2024.

Sinha, K., Sodhani, S., Dong, J., Pineau, J., and Hamilton, W. L. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

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. *Advances in Neural Information Processing Systems*, 35, 2022.