

Where Does the Accuracy of LLM-Based Root-Cause Analysis Come From? An Empirical Study on Microservice Benchmarks

Muhammad Saad Hussain

muhammadsaadhussain49@gmail.com

The University of Lahore <https://orcid.org/0009-0003-9132-9986>

Alishba Farrukh

The university of lahore

Research Article

Keywords: AIOps, root-cause analysis, large language models, microservices, empirical study, pre-registration

Posted Date: October 19th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11401166/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare potential competing interests as follows: The authors developed the system under study. They declare no other competing interests.

Where Does the Accuracy of LLM-Based Root-Cause Analysis Come From? An Empirical Study on Microservice Benchmarks

Muhammad Saad Hussain^{1*} and Alishba Farrukh¹

^{1*}Department of Computer Science, The University of Lahore,
Sargodha, Pakistan.

*Corresponding author(s). E-mail(s):

muhammadsaadhussain49@gmail.com;

Contributing authors: alishba.farrukh@cs.uol.edu.pk;

Abstract

Context. In many AIOps pipelines, detectors summarise telemetry and an LLM names the faulty service. **Objective.** To determine how much of a root-cause analysis agent’s accuracy comes from its detectors, the text its pipeline adds, and the model. **Method.** A pre-registered study of one open-source agent on RCAEval RE2 (270 cases) and RE3 (90 code-level faults) with three open-weight models. We compare the agent with its pipeline, classifiers, a SimpleRCA-style heuristic and RCAEval baselines; vary only the text (interpretive NOTEs, a false NOTE, raw logs, service names); re-test the main findings in a generic pipeline; and cross the two pipelines’ evidence and prompts. **Results.** On RE2 the agent (AC@1 0.44) trails its detector count (0.60) and N-Sigma (0.80). On RE3, removing raw log text cost it 24 points. A generic pipeline raised the same model to 0.87 on RE2, level with its label-free ranking; a 2×2 traced most of this difference to the evidence shown (+0.30) and less to the prompt (+0.05), but a two-minute onset error dropped it to 0.27. A false claim moved the answer in both pipelines; in the agent, a decoy was named first in 21–42% of flagged RE2 cases (about 4% without it). **Conclusion.** The model’s accuracy tracked what its pipeline extracted and at most modestly exceeded a label-free reading of it; asserted text pulled answers off the evidence. Evaluations should compare the model with a label-free reading of its input and across context designs, and test the text with decoys, ablations and onset shifts.

Keywords: AIOps, root-cause analysis, large language models, microservices, empirical study, pre-registration

1 Introduction

Large language models (LLMs) are increasingly placed at the end of AIOps pipelines. Deterministic detectors watch metrics, traces and logs, turn what they see into a short textual context, and an LLM reads that context and names the faulty service (Chen et al. 2024; Wang et al. 2024, 2025; Zhang et al. 2025a). When such a system is evaluated, its overall accuracy is usually credited to the model. Yet the model sees only what the pipeline chose to show it. Some pipelines also add interpretive sentences of their own: hints that a memory reading means memory pressure, or that an error message points to a failed dependency. Whether the accuracy comes from the model’s reasoning, from the detectors, or from those sentences is rarely measured.

Two lines of evidence suggest that the question matters. First, simple methods do well on public root-cause analysis (RCA) benchmarks. On RCAEval, simple statistical methods such as N-Sigma are among the most accurate (Pham et al. 2024b, 2025), and a rule-based count of alerts performs comparably to or better than state-of-the-art methods on four public benchmarks, RCAEval among them (Fang et al. 2026b). Second, LLMs are sensitive to what their context says. Irrelevant text distracts them (Shi et al. 2023), a biasing cue can flip an answer without being mentioned in the explanation (Turpin et al. 2023), and formatting alone moves accuracy (Sclar et al. 2024). Several studies run LLMs or LLM agents on RCAEval and report their accuracy against other methods (Pham 2026; Tian et al. 2025; Muhammad et al. 2026; Huang et al. 2026). None of them removes interpretive hints while holding the evidence fixed or plants a false or neutral hint, and none compares the model with its own pipeline’s detector output or with a classifier given the same inputs.

We study one such system, an open-source multi-signal AIOps agent that we built and evaluated on a live deployment (Hussain and Farrukh 2026b), on RCAEval. The study was pre-registered. Before any RE2 fault window reached a detector, a baseline or a model, we committed the research questions, predictions, comparators, statistics and decision rules to a version-controlled repository and tagged the commit. Before the freeze, only the file structure and fault-free windows of the data were inspected, together with a 21-case RE1 and RE3 pilot whose accuracy was not inspected. Five follow-up studies were each pre-registered in the same way before their own runs, and GitHub’s server-side log confirms that every stage reached GitHub before its runs began. On the 270 cases of RE2 we compare the agent with:

- its own pipeline without the model;
- classifiers trained on exactly the features the prompt shows;
- a SimpleRCA-style heuristic;
- the RCA methods RCAEval ships.

We then change only the text: we remove the pipeline’s interpretive NOTES, insert a false one (a *decoy*) or a neutral one (a *placebo*), remove the raw log text, and hide or permute service names. We use three open-weight models, replicate the study on RE3’s 90 code-level faults, re-test three exploratory RE2 findings as pre-registered hypotheses on RE3, re-test the headline findings in a generic pipeline that shares with the agent only the data, the window, the candidate set, the scoring, the error

keywords and the seeded decoy service, and finally cross the two pipelines’ evidence and prompts. We label each finding below by the status of its evidence.

Our main findings are:

1. *Confirmatory (RE2)*. The deployed model (AC@1 0.44) is less accurate than its own pipeline’s detector count (0.60), than a logistic regression on the features it sees (0.63; pre-registered non-inferiority supported, and the classifier is superior), and than N-Sigma (0.80).
2. *Confirmatory, replicated on RE3*. A false interpretive sentence moves the answer. A decoy NOTE made the three models name the decoy service first in 21–42% of the 245 flagged RE2 cases, against about 4% when the decoy was absent. For the deployed model, a placebo naming the same service without a claim did so in 7%: the claim, not the mention, moves the answer.
3. *Replication (RE3), mixed*. On RE3’s code-level faults the classifier result does not replicate. The model’s AC@1 (0.60) is above its detector count (0.38) and the same-feature classifier (0.42), but neither difference is significant after Holm correction, and the model stays below N-Sigma (0.73; n.s.).
4. *Exploratory (RE2), with pre-registered re-tests on RE3*. Three post hoc findings were re-tested:
 - Suppressing log alarms that already fired before the fault raised the model from 0.44 to 0.60 on RE2, level with its detectors. The RE3 re-test did not support this.
 - Hiding every service name did not lower accuracy on RE2. On RE3 it lowered the deployed model’s accuracy by 21 points, so the RE3 re-test did not support this either.
 - Placing the onset two minutes late collapses the strongest statistical baselines, extending the observation of [Pham et al. \(2024b\)](#) that N-Sigma is sensitive to the onset time: N-Sigma fell from 0.80 to 0.07 and the SimpleRCA-style heuristic from 0.81 to 0.23, while the model fell from 0.44 to 0.33. The pre-registered RE3 test confirmed this for both methods.
5. *Pre-registered follow-up: log text and names (RE3)*. On RE3 the model uses the raw log text: removing it, while keeping everything the log detector reports, lowered the model from 0.60 to 0.36; on RE2 no change was detected. Descriptively, without the text the model’s non-significant lead over its detector count disappears. Deranging the service names cost 0.20, so names that denote the right service matter; no difference was detected between deranged names and neutral labels.
6. *Pre-registered follow-up: a generic pipeline*. In a second pipeline that we built from a plain statistical summary of the same data, the decoy effect reappears (H7a): one sentence asserting a wrong root cause made `gpt-oss-120b` name that service first in 62% of all RE2 cases (`qwen3.8-27b`, 24%). The shortfall against the pipeline’s own label-free ranking (finding 1) and the dependence on log text (finding 5) were not detected there (H7b–H7d), so by the pre-registered rule we report them as specific to our agent. Descriptively, the same model reaches 0.87 on RE2 and 0.89 on RE3 in that pipeline, close to the label-free ranking of its own summary (0.83 and 0.80).

7. *Pre-registered follow-up: evidence or prompt.* Crossing the two pipelines’ user messages (the evidence) with their system prompts showed that the evidence carries most of the difference: it raised accuracy on the flagged cases by 0.30, the prompt by 0.05 (H8a, H8b). The generic pipeline, however, depends on the onset like the baselines it resembles: two minutes late, it fell from 0.87 to 0.27 on RE2 (H8c), close to or below the deployed agent’s model under the same shift (0.33, descriptive).

This paper makes four contributions:

- A controlled decomposition of where an LLM RCA agent’s accuracy comes from, separating the detectors, the interpretive text and the model. With the deployed model on RE2, the model was less accurate than its agent’s own detector count and not detectably different from a label-free ranking of the generic pipeline’s summary; elsewhere it was at most modestly ahead of such rankings. Crossing the two pipelines’ evidence and prompts showed that the evidence accounts for most of the difference between them and the prompt for a smaller part.
- A staged pre-registered design for such studies, with a server-verified timeline: a main study, a replication on a second benchmark suite, pre-registered re-tests of exploratory findings, and text ablations that separate what the model reads from what its detectors measure.
- Evidence that changes how such systems should be evaluated. We argue for comparing the model with a label-free reading of its own input and across context designs, reporting the detector-only and same-feature baselines, ablating and stress-testing the text a pipeline adds, and reporting sensitivity to the onset time, for the pipeline as well as for the baselines.
- A replication package with the protocol and every pre-registered stage, every prompt, raw response and per-case ranking, and a script that reruns every analysis without model calls.

Section 2 reviews related work and Section 3 describes the system. Section 4 gives the study design and Section 5 the results. Section 6 discusses them, Section 7 the threats to validity, and Section 8 concludes.

2 Background and Related Work

2.1 Root-cause analysis for microservices and its benchmarks

AIOps applies data-driven methods to the operation of large software services, and incident management is a central task (Dang et al. 2019; Chen et al. 2020; Notaro et al. 2021; Remil et al. 2024). For microservice systems, root-cause analysis (RCA) asks which service, and sometimes which metric or code location, caused an observed failure (Soldani and Brogi 2022; Zhang et al. 2024a). Established methods range from statistical change detection, such as N-Sigma and BARO (Pham et al. 2024a), to causal discovery and inference, such as RCD (Ikram et al. 2022) and CIRCA (Li et al. 2022a), multi-source methods such as Nezha (Yu et al. 2023) and TORAI (Pham et al. 2026), and supervised models such as Eadro (Lee et al. 2023) and DejaVu (Li et al. 2022b).

RCAEval (Pham et al. 2025) collects 735 cases from three open-source systems with injected faults, together with reference implementations of many established methods and a common evaluation. It evaluates no LLM-based method. Studies on and around it have repeatedly found that simple methods hold their own. Pham et al. (2024b) found most causal-discovery methods close to random, with N-Sigma and BARO fast and accurate. Fang et al. (2026b) showed that SimpleRCA, a rule-based count of metric, trace and log alerts per service, performs comparably to or better than state-of-the-art methods on four public benchmarks, including RCAEval RE2 and RE3. Hu et al. (2026) showed that pooled leaderboards hide system-specific winners, with effects that change sign between systems. We follow these lessons: we include N-Sigma and a SimpleRCA-style heuristic, report every result per system, and use paired, cluster-level tests.

2.2 LLMs and LLM agents for root-cause analysis

Early industrial work used LLMs on the text of cloud incidents: fine-tuned models that recommend root causes and mitigations (Ahmed et al. 2023), in-context learning over past incidents (Zhang et al. 2024c), and systems in which handlers collect diagnostics and an LLM names the category (Chen et al. 2024). Agents that call tools followed, in production (Wang et al. 2024; Roy et al. 2024) and in research prototypes, some of them multi-agent (Zhang et al. 2024b; Wang et al. 2025; Zhang et al. 2025b; Naakka et al. 2026); surveys trace the shift (Zhang et al. 2025a). The architecture we study belongs to the group in which deterministic components summarise telemetry and the LLM interprets the summary (Chen et al. 2024; Wang et al. 2025).

Benchmarks for LLM agents report low absolute success: the best agent on OpenRCA solved about 11% of its failures (Xu et al. 2025), and agents on ITBench resolved 11.4% of SRE scenarios (Jha et al. 2025); see also AIOpsLab (Chen et al. 2025), CloudOpsBench (Wang et al. 2026) and a multi-dataset benchmark that warns that outcome scoring can reward keyword matching (Cai et al. 2026). OpenRCA 2.0 found that over a third (38.5%) of correct service-level answers were not grounded in a verified propagation path (Fang et al. 2026a).

Four studies run LLMs on RCAEval. Pham (Pham 2026) reports an LLM agent (RCLAgent) as a baseline at 0.10 top-1, far below a statistical ranker. GALA re-ranks BARO’s output with an LLM on the 90 RE2-OB cases and raises top-1 from 14% to 42% (Tian et al. 2025). DecompRCA audits four benchmarks, RCAEval among them, and splits accuracy into retrieval (is the root among the candidates?) and re-ranking (is it ranked first?); statistical baselines mostly fail at re-ranking, and an LLM that re-ranks the candidates of a multi-signal retriever improves top-1 over the best baseline (Muhammad et al. 2026). OpsHarness evaluates coding agents on the 270 RE2 cases, with LLM baselines only (Huang et al. 2026). GALA and DecompRCA compare an LLM with statistical rankers on the same candidates. We extend that comparison to the detector output an agent is actually given and to a classifier trained on the features it sees, and, unlike all four, we remove interpretive hints while holding the evidence fixed and plant false or neutral ones.

The closest work on what the model is told is by Luo et al. (2026), who vary the tools given to an agent on RCAEval RE1, from a bare Python interface to tools that

return candidate root causes. An agent given only diagnostic tools localised slightly less often than one given only the Python interface (AC@1 82.8% against 85.4%), agents overrode the tools’ candidates more often than they corrected them, and the ranking of tool designs changed across systems. They compare the agent’s final answer with the first candidate its diagnostic tools return, but do not plant false information or separate the evidence an agent is shown from its prompt. [Riddell et al. \(2026\)](#) withhold whole data sources and find that removing traces sometimes helps; [Kim et al. \(2026\)](#) trace most agent failures on OpenRCA to hallucinated interpretation, which prompting does not fix; and [Gopal and Krishnan \(2026\)](#) find that the needed evidence is usually present and that injected knowledge partly compensates for weak models. We study the prompt level that these works leave open: interpretive sentences that a pipeline writes into the context. We remove them, plant false ones, plant neutral ones, and measure how often the model’s top-1 follows.

2.3 Simple baselines, shortcuts and prompt sensitivity

Software engineering has a tradition of checking complex models against simple ones ([Fu and Menzies 2017](#); [Hellendoorn and Devanbu 2017](#); [Liu et al. 2018](#); [Le and Zhang 2022](#); [Yu et al. 2024](#)), and of showing that benchmarks can be solved by simple means ([Landauer et al. 2024](#); [Wu and Keogh 2021](#)). Machine-learning research names the underlying risk shortcut learning: a model can succeed through a cue that is correlated with the label rather than through the intended reasoning ([Geirhos et al. 2020](#)); see also ([Arp et al. 2022](#); [Li et al. 2023](#)). For LLMs, irrelevant context distracts the model ([Shi et al. 2023](#)), a biasing feature in the prompt can flip the answer while the explanation does not mention it ([Turpin et al. 2023](#)), and formatting alone moves accuracy ([Sclar et al. 2024](#)). Our decoy, placebo and order-shuffle conditions are tests of these effects in an operational setting.

2.4 Empirical methods for studies involving LLMs

Studies that use LLMs face threats from non-determinism, model drift and contamination ([Sallou et al. 2024](#); [Ouyang et al. 2025](#); [Zhou et al. 2025](#); [Cao et al. 2024](#)). Guidelines ask authors to report models and settings, release prompts and outputs, include an open model, and use suitable baselines ([Wagner et al. 2025](#); [Baltes et al. 2025](#)); general empirical standards ask for full reporting of benchmark studies ([Ralph et al. 2020](#)), and statistical guidance warns against single runs of stochastic methods ([Arcuri and Briand 2011](#)). Registered reports are one response to flexibility in analysis ([Ernst and Baldassarre 2023](#)). We pre-registered our hypotheses, analysis and decision rules, pre-registered each follow-up study before its runs, used only open-weight models, measured run-to-run variation for the deployed model over three full runs, and report how the results change when service names are hidden.

3 The System Under Study

The system studied here is an open-source AIOps pipeline for Kubernetes that we built and evaluated live on an Online Boutique deployment ([Hussain and Farrukh](#)

2026b). It is typical of a common design: deterministic detectors turn raw telemetry into a short textual context, and a large language model (LLM) reads that context and names the root cause. We describe only the parts that the benchmark study exercises. Section 4.3 lists every change made to run them on offline benchmark data.

3.1 Detectors

Metrics

Each monitored service has its own Isolation Forest (Liu et al. 2008) (scikit-learn (Pedregosa et al. 2011), contamination 0.1, seed 42), fitted on a fault-free baseline. A row has four features: the CPU rate, its rolling standard deviation over six samples, the memory in MB, and an error rate. The anomaly score is the raw Isolation Forest score divided by the 95th percentile of the scores on the baseline, so a value above 1 is more unusual than 95% of the service’s own baseline. A crossing counts only if the service’s mean CPU is at least $\max(0.005 \text{ cores}, 1.5 p95_s)$, where $p95_s$ is the 95th percentile of its baseline CPU. Two rules flag a service independently of the forest: a *memory-pressure* trigger (memory at least 1.2 times the baseline mean and at least 10 MB above it) and a *restart* trigger (a pod younger than 300 s). The scorer also reports CPU and memory relative to the baseline and, when known, to the configured limit.

Traces

For each service the analyser takes up to 100 recent traces that contain it and pools the durations of their spans. A service is flagged if its p99 latency exceeds 2 s or, failing that, if more than 5% of its spans are errors.

Logs

For each service the parser takes up to 500 lines from the last 5 minutes and classes a line as an error if it contains one of *error*, *exception*, *failed*, *refused* or *timeout*. Error lines are clustered into templates with Drain3, an implementation of Drain (He et al. 2017). The parser also scans error lines for *dependency hints*, words that point to a datastore, to gRPC failures or to name resolution. A service is flagged if it has more than 10 error lines in the window, at least one dependency hint, or a template not seen before.

3.2 Context and interpretive annotations

A context builder merges the three result sets. Each flagged (service, signal) pair is one anomaly entry, and their number sets the severity, from LOW (one) to CRITICAL (four or more). The builder writes a textual summary for the model: the severity, an **AFFECTED SERVICES** line, one line per metric, trace and log anomaly (with the anomaly score, relative CPU and memory, latency, and a sample error line), a list of services flagged by more than one signal, and a fixed task instruction.

While building the summary, the builder appends NOTE lines when certain patterns occur. These *interpretive annotations* were added during development to supply interpretations the model had missed. Four kinds prescribe a cause: a crash or restart

(two kinds), memory pressure, and a failed dependency; two of them also name the expected verdict outright. A fifth kind labels a CPU reading near the configured limit as saturation. On the benchmark only two kinds can fire (Section 4.3):

- **Memory pressure**, attached to a service that meets the memory trigger: “Treat this as a RESOURCE/MEMORY-PRESSURE event ... This is a confirmed anomaly (verdict=ANOMALY_CONFIRMED), not a false positive.”
- **Dependency**, attached to a log anomaly with a dependency hint: “this error text specifically references $\langle \text{service} \rangle$ – ... consider whether the TRUE root cause is that dependency being unavailable.”

3.3 Diagnosis agent

The agent is called only when the context holds at least one anomaly entry; otherwise the cycle is recorded as normal. The system prompt gives the agent its role, names the data sources and the services, states that a service missing from the telemetry is itself evidence, and fixes a JSON response schema: a verdict, a confidence, the affected services, a root-cause narrative, evidence and remediation steps. The user message is the context summary followed by the per-service records. The deployed configuration uses temperature 0.1, at most 3,000 output tokens, one retry when the response breaks the schema, and five samples (votes) per cycle, of which the majority verdict is kept. An advisory gate then routes each diagnosis to an operator; no command is executed automatically. The gate and remediation are outside the scope of this study.

4 Study Design

The study was pre-registered. Its research questions, predictions, comparators, statistics and decision rules were committed to a version-controlled repository and tagged before any RE2 fault window reached a detector, a baseline or a model. Before the freeze, only the file structure and the fault-free windows of the data were inspected (to fix units and parsing), and a pilot on 15 RE1 and 6 RE3 cases tested the pipeline; its accuracy was not inspected (Section 4.8). This section follows the protocol. Later stages were each pre-registered before their own runs, and departures in execution are stated here and in a numbered amendment (Section 4.8).

4.1 Research questions and predictions

RQ0 (coverage). How many benchmark cases trigger at least one detector, and how many carry an interpretive NOTE?

RQ1 (state of the art). How accurately does the agent localise the root-cause service compared with established RCA methods, a SimpleRCA-style heuristic (Fang et al. 2026b) and the pipeline’s own detectors? We made no directional prediction.

RQ2 (features). Does a classifier trained on the features the agent sees localise the root cause as accurately as the agent? *P2 (non-inferiority)*: the lower 95% bound of the AC@1 difference (classifier minus agent) lies above -0.05 .

RQ3 (annotations). How far does interpretive text decide the agent’s answer?

- *P3a*: on cases whose context carries a NOTE, removing the NOTES changes the top-1 service more often than re-running the same text does.
- *P3b (central)*: a NOTE claiming that a non-root service is the problem (a *decoy*) makes the model name that service first more often than without it. As a secondary test, for the deployed model, also more often than a *placebo* NOTE that names the same service without a claim.
- *P3c*: on NOTE cases, the three models agree on the top-1 service more often with NOTES than without them.

RQ4 (memorisation). Does replacing service names with neutral labels change the deployed model’s AC@1? We predicted no direction and test equivalence: the 90% interval of the difference lies within ± 0.05 .

P2 and P3 come from an earlier, smaller study of the same system on live faults (Hussain and Farrukh 2026b), in which removing the NOTES reversed 17 of 20 crash verdicts and a logistic regression on the same features matched every model outside the crash scenario. That study had one system, a small fault set, and no established RCA baseline; this study addresses all three. That study has not been peer reviewed; its data and results are archived with the system (Hussain and Farrukh 2026b). The protocol also listed an RQ5, which reports that live study as field evidence with no new analysis. We do not report RQ5 as a separate result here, because the earlier study is not yet published; we refer to it only where it bears on our results. This departs from the protocol, which planned to report RQ5 as published.

4.2 Benchmark and cases

RE2

The confirmatory set is the RE2 suite of RCAEval (Pham et al. 2025): 270 cases from three microservice systems, Online Boutique (OB), Sock Shop (SS) and Train Ticket (TT) (Zhou et al. 2021). Each system has five root-cause services, six injected faults (CPU, memory, disk, socket, delay and packet loss) and three repetitions of each, with metrics sampled every second, logs, and traces for OB and TT. Each case holds 12 minutes of data before the injection and 12 after. We used RCAEval’s Zenodo release with its code pinned to commit 259ea41, and recorded the hash of every data file before the freeze.

Ground truth and candidates

The root-cause service is taken from the case folder name, with database siblings (-db) folded into their owner, as RCAEval does. It is used only for scoring. The candidate set of a case is every service with a metric column, which is the set RCAEval scores; every method, and chance, ranks the same set.

Data properties noted before the freeze

Sock Shop has no traces, and Train Ticket spans carry no status code, so trace errors cannot fire there. Two OB cases have only 210 and 284s of data after the injection; they are kept, and every method sees the same shortened window.

Table 1 Differences between the live system and the benchmark port. The full list (A1–A14, B1–B6) is in the protocol.

Live system	Benchmark port	Consequence
60-min fault-free baseline per forest pod age, CPU limits from the cluster	the 10 min before the injection not in RCAEval	shared window for every method restart and CPU-saturation NOTES cannot fire; only memory and dependency NOTES can
Prometheus rates at a 30-s step traces: error tag five hard-coded services AFFECTED SERVICES in hash order	RCAEval 1-s samples averaged into 30-s bins gRPC code 1–16 or HTTP ≥ 500 every service with a CPU column same code, fixed hash seed	same resolution; gaps are not filled codes in RCAEval mix both names matched by longest prefix reproducible, order still arbitrary
prompt lists services with ports and dependencies JSON schema	services only adds <code>root_cause_service</code> and a ranked list (up to 5)	topology hints are lost needed for AC@k
five votes per cycle	one vote	checked on 30 seeded cases, 27 of them flagged (Section 4.5)

RE3 (replication)

After the RE2 results were known, we pre-registered a replication on RCAEval’s RE3 suite: 90 cases with code-level faults (OB f1–f5, SS and TT f1–f4), three or four repetitions per service and fault. One RE3-TT folder holds a second batch of one service and fault (`ts-route-service_f3.1`); its three cases join the same cluster. Six RE3 cases had been run in the pilot, with their accuracy not computed (their responses were checked only for name matching); the primary RE3 analysis uses all 90 cases, and a sensitivity analysis excludes the six.

4.3 Running the system on benchmark data

The detectors were ported to offline data and return records in the live schema; the deployed context builder is imported and called unchanged, with its clock frozen to the cycle time. Table 1 lists the differences from the live system that matter for interpretation. Each case is one cycle, run ten minutes after the injection; every method sees the same 20-minute window (RCAEval’s `--length 20`).

From response to ranking

A response is turned into a full ranking of the candidates: the ranked list, then the single root-cause service, then the affected services, each name folded and kept only if it is a candidate. The remaining candidates follow in a seeded random order that is fixed per case and shared by every method, so unlisted services earn no more than chance.

Gating

As deployed, the model is called only when a detector fires. Unflagged cases receive the case’s chance ranking and stay in every denominator; we also report every score on flagged cases only.

Table 2 Prompt conditions. Conditions of the frozen protocol in the first block; conditions added by Amendment 1 and the RE3 replication in the second (exploratory on RE2; anon_full and baseline-aware were later the conditions of the pre-registered RE3 tests H5b and H5a); the conditions of the pre-registered text ablation (H6) in the third; and those of the generic pipeline (H7) in the fourth; the fifth block crosses the two pipelines’ prompts and evidence (H8).

Condition	Text	Models
annotated	the deployed context	all three
annotated, rerun	the same text again, on NOTE cases (within-case noise for P3a)	all three
stripped	every line beginning NOTE: removed	all three
decoy	stripped plus one decoy NOTE about a seeded non-root service	all three
placebo	stripped plus “NOTE: (service) is one of the services deployed in this cluster”	deployed
anon	service names replaced by service-NN in the structured text	deployed
vote check	annotated with five votes on 30 seeded cases	deployed
annotated, two reruns	run-to-run variation (E1)	deployed
shuffled	order of the services in the AFFECTED SERVICES line and in the user message’s affected-services list permuted, two seeds (E2)	deployed
baseline-aware	an error-count alarm fires only if the count also exceeds twice that of the 5 minutes before injection; dependency hints already present then are dropped (E3)	deployed
anon_full	every spelling of every service name and every system name replaced, raw log text included	all three
nolog	all raw log text removed (sample error lines, mined log templates); the log detector’s flags, counts and reasons kept	all three
permuted	real service names deranged among the candidates, every spelling, everywhere; answers mapped back before scoring	all three
g_full	generic summary of metrics, error-log counts with example lines, and traces; plain prompt	deployed, qwen3.8-27b
g_nolog	g_full without the example error lines	deployed, qwen3.8-27b
g_decoy	g_full plus one sentence naming the decoy service as the root cause	deployed, qwen3.8-27b
x_dsys_guser	deployed system prompt with the generic user message	deployed
x_gsys_duser	generic system prompt with the deployed user message	deployed
g_full, onset ± 120 s	generic summary rebuilt with the onset two minutes early or late	deployed

4.4 Prompt conditions

Table 2 lists the conditions. All differ from the deployed context in the text alone; the detector output is the same.

Names used in the paper

Unless a pipeline or model is named, *the model* and *the deployed model* mean **gpt-oss-120b** in the deployed agent with the **annotated** context. The *same-feature classifier* is the logistic regression on the features the prompt shows (**1r_visible**). *Multi-source* marks RCAEval’s multi-source variants of N-Sigma, BARO and CIRCA

(prefix `mm` in the package). The *detector count*, *first-listed* and *largest-anomaly* orderings are the agent’s pipeline without the model, and `summary_count` is the generic pipeline’s equivalent. `baseline-aware` is the condition that suppresses log alarms already present before the fault.

Decoy and placebo

The decoy service is drawn uniformly from the candidates other than the root cause, with a seed derived from the case identifier. The decoy text is drawn uniformly from the two deployed `NOTE` kinds that can occur on the benchmark, memory pressure and dependency, and is the deployed builder’s own wording with the decoy’s name inserted. The placebo names the same service without any claim. Each inserted `NOTE` is its own block, placed where the builder places its `NOTES`. The pre-registered harm outcome is how often the decoy is named first among the cases the model gets right without it.

Anonymisation

`anon` replaces every service name in the structured text, but in raw log lines only names that cannot be ordinary words (those with a hyphen or ending in “service”). The pre-registered leak count, which looks for the exact names, found an original name in 86 of 90 Sock Shop prompts, mostly through the service name that Spring Sleuth writes at the start of every log line (`[carts,,]`). Because this left RQ4 unable to separate memorisation from leakage, the RE3 replication added `anon_full` (exploratory on RE2). It uses the same `service-NN` labels and replaces, everywhere and case-insensitively, the exact name, the name without Train Ticket’s `ts-` prefix, and both with hyphens removed or replaced by an underscore or a space. Forms of ten or more characters are also replaced inside longer identifiers (`InsidePaymentServiceImpl`); shorter forms only as whole tokens. System names (Sock Shop, Online Boutique, Train Ticket and their code names) become `app`. It also defined a *strict* leak detector that searches for all of these forms. Under it, `anon_full` leaves no name in any of the 245 flagged RE2 prompts, against 87 Sock Shop and 79 Train Ticket prompts (166 in all) under `anon`. Because the mapping from names to labels is consistent, a log line that named the failing service still names its label: `anon_full` removes what a name means, not which service a line refers to. Ordinary words such as “order” or “payment” in a log message also remain.

Log text and names (H6)

After every result above had been seen, and before any run of the two conditions, a follow-up study pre-registered a direct test of the explanation that RE3 had left open. `nolog` removes every piece of raw log text from the prompt: the sample error line, the mined log templates in the summary, and the same fields in the anomalous-logs block. What the log detector reports stays: which services it flags, their error and log counts, its other reasons, and the dependency keywords. `permuted` deranges the real service names among the candidate services with a seed derived from the case, so that no service keeps its own name, and rewrites every spelling of every name everywhere with the matching rules of `anon_full`, keeping each spelling’s form and case (`InsidePaymentServiceImpl` becomes, say, `SecurityServiceImpl`). The

model’s answers are mapped back through the permutation before scoring. Names therefore still look like names and a log line still points at the same, renamed, service; only the correspondence between a name and the service it really is is broken. Two checks are stored with every call: for **nolog**, how many of the context’s raw log strings remain in the prompt, and for **permuted**, whether the prompt with every name removed equals the original prompt with every name removed. The four pre-registered tests are given with the results (Section 5.8).

A generic pipeline (H7)

Every condition above changes the text of one agent, which we built. A follow-up study re-tests the headline findings in a second pipeline that shares nothing with ours except the data, the 20-minute window, the candidate set, the scoring, the error keywords and the seeded decoy service. For every candidate service it reports three things:

- every metric column whose mean in the ten minutes after the injection differs from the mean before by at least three standard deviations of the before window (the four largest per service);
- for services whose error-line count rose, the counts before and after and up to three distinct example error lines;
- for services whose p95 span duration at least doubled or whose error spans rose, those figures.

A plain prompt asks which service the fault was injected into, and the model is called on every case, with no gate. Three conditions change the text: **g_full** is the summary; **g_nolog** removes the example error lines and keeps the counts; and **g_decoy** adds one sentence stating that the monitoring system’s analysis indicates the main study’s decoy service is the root cause. The label-free analogue of the detector count, **summary_count**, ranks services by what the summary itself shows: the number of metrics beyond 3σ , plus one for a rise in error lines, plus one for a trace flag. The deployed model is confirmatory and **qwen3.8-27b** descriptive. We designed this summary after the earlier results had shown how well per-column statistics such as N-Sigma’s do on RCAEval, and it resembles them; like them, it compares the windows before and after the given onset. The pipelines also differ in more than the evidence they show: the generic prompt states that one fault was injected into one service, asks for a simpler answer, and has no severity gate. Its comparison with the deployed agent therefore measures the effect of the whole pipeline, not of the evidence alone.

Evidence or prompt, and the onset (H8)

A further pre-registered follow-up study separates the two. On the deployed agent’s flagged cases, it crosses the user message of each pipeline (the *evidence*: what the pipeline extracts and shows) with the system message of each (the *prompt*: persona, instructions and answer schema). Two cells already existed (**annotated** and **g_full**); **x_dsyst_guser** and **x_gsys_duser** are the crossed ones (Table 2). Restricting all four cells to the flagged cases removes the gate as a difference. The evidence effect is the mean of the two cells with the generic evidence minus the mean of the two with the deployed evidence; the prompt effect is defined in the same way. The study also rebuilds the

generic summaries with the onset shifted by ± 120 s, as in Amendment 4 and the RE3 re-tests, and runs `g.full` on them. Its tests are H8a (the evidence effect is positive), H8b (the prompt effect, two-sided) and H8c (the generic pipeline loses accuracy when the onset is 120 s late), Holm-adjusted together.

4.5 Models

We used the three models of the live study: `gpt-oss-120b` (the deployed model), `gpt-oss-20b` (same family, smaller) and `qwen3.8-27b` (a different family). All three are open-weight and were served by one provider (Groq) at temperature 0.1 with at most 3,000 output tokens, reasoning at the provider default, and one schema retry, as deployed. Each case is an independent call with no history. The deployed system draws five votes per cycle; for cost we drew one, and checked the choice on 30 seeded cases, where the first vote’s top-1 equalled the unique five-vote plurality in 25 of 27 flagged cases.

4.6 Comparators

The pipeline without the model

`detector_count` ranks services by the number of signals that flag them, with ties broken by the anomaly score shown in the prompt. `first_affected` uses only the order of the `AFFECTED SERVICES` line. `largest_anomaly` ranks by the Isolation Forest score alone. None uses a label.

Same-feature classifiers

One row per candidate and case, with 13 features the prompt shows (metric, trace and log flags, anomaly score, memory trigger, relative CPU and memory, p99 latency, trace error rate, log error count, dependency hint, new templates, number of signals), label = root cause. Values the prompt does not show are masked to zero. `lr_visible` is a class-balanced logistic regression evaluated with five-fold cross validation grouped by (system, root service, fault), so the three repetitions of a fault never straddle a split. We also report a random forest (`rf_visible`), the logistic regression on unmasked features (`lr_all`), a leave-one-system-out version (`lr_visible_loso`), and, on RE3, the RE2-trained model applied unchanged (`lr_visible_re2train`), which uses no RE3 label. These classifiers are references for what the agent’s inputs support, not proposed RCA methods: `detector_count` is the label-free comparison, and the leave-one-system-out model the strictest supervised one.

SimpleRCA-style heuristic

Our reimplement of the alert-counting heuristic of Fang et al. (2026b): one alert per metric column whose post-injection mean leaves the pre-injection mean $\pm 3\sigma$ (any change if $\sigma = 0$), one if a service’s post-injection p95 span duration is at least three times its pre-injection p95, and one if its error-keyword log lines are at least 5 and at least three times the pre-injection count. Services are ranked by alerts, with seeded ties.

RCAEval baselines

We ran RCAEval’s own implementations with `main.py --length 20` and a two-hour limit per method and system. The pre-registered primary methods are N-Sigma, BARO (Pham et al. 2024a), CIRCA (Li et al. 2022a), ϵ -Diagnosis, MicroCause, and the multi-source variants of N-Sigma, BARO and CIRCA; TraceRCA, MicroRank and CausalRCA are secondary. A method that fails or times out keeps its finished cases and the rest are scored as chance. Executing these methods needed repairs, made through Amendment 2 and the RE3 replication before the runs they govern (Section 4.8). Four are source patches to the pinned RCAEval code, each recorded with the file’s hash before and after: P1 removes a statement in `main.py` that stopped a whole run at the first failing case, against the protocol’s per-case rule; P2 replaces, for MicroCause, a tigramite call whose behaviour changed in version 5 by a verbatim copy of the pinned version’s function; P3 reads one partial correlation with `.iloc[0]` under pandas 3 (the same number); and P4 lets `main.py` parse RE3’s batch-folder names. Three are environment changes: installing PyRCA (pinned commit) for ϵ -Diagnosis, adding its dependency `dill`, and pinning `setuptools` below 81 so that PyRCA imports. None changes a package used by our pipeline or by the methods already run. Table B1 (Appendix B) gives the outcome. TraceRCA and MicroRank cannot run through `main.py` at the pinned commit (it passes them the metric table rather than spans), and on RE3 the multi-source variants lack the preprocessed log and trace files they read. We report these as not runnable rather than write our own adapters. Finally, RCAEval’s harness passes MicroCause the root service’s own latency as its starting metric when that column exists; MicroCause’s numbers therefore use ground-truth information and are not interpreted as label-free.

4.7 Measures and statistical analysis

Measures

AC@1, AC@3, AC@5 and Avg@5 (the mean of AC@1 to AC@5) at service level, computed with RCAEval’s own evaluator, per system and pooled, on all cases and on flagged cases. Chance Avg@5 is computed over the same candidate sets. The confirmatory statistic for every comparison is the paired difference in AC@1 between two methods (or two conditions) on the same cases.

Inference

The three repetitions of a (system, root service, fault) are not independent, so inference is done over these clusters (90 on RE2, about 30 on RE3): a 95% cluster-bootstrap interval of the paired difference (2,000 resamples) and a two-sided cluster sign-flip permutation test (10,000 permutations), both seeded. Where few clusters differ, we also report the exact permutation p-value by enumeration (Amendment 1); verdicts follow the pre-registered Monte Carlo value, and a test whose two values fall on different sides of 0.05 is reported as borderline. Exact McNemar tests are reported alongside as an unclustered secondary analysis.

Families and decision rules

Holm’s procedure (Holm 1979) is applied within each family: the 13 RQ1/RQ2 comparisons of the deployed model with the same-feature classifier (`lr_visible`), the four label-free pipeline and heuristic comparators and the eight primary RCAEval methods; P3a and P3b with one test per model each; P3c, RQ4 and decoy versus placebo as single tests. P2 is supported if the lower 95% bound of (same-feature classifier – agent) exceeds -0.05 , contradicted if the upper bound is below zero with $p < 0.05$, and inconclusive otherwise. P3a–P3c are supported if the Holm-adjusted $p < 0.05$ in the predicted direction. For P3b, a model that names the decoy first in more than half of the decoy cases is said to let the decoy outweigh the evidence.

Sensitivity

Assuming 30% discordant cases, the smallest observed difference that can reach $p < 0.05$ is about 7 points on 270 pooled cases, 10 points after Holm over 13 comparisons, and 14 points on 90 cases. Smaller differences are reported as not detected, not as absent.

Replication rule (RE3)

A comparison that met its decision rule on RE2 is *replicated* if RE3 gives the same sign and meets the same rule, *same direction*, *not significant* if only the sign agrees, and *opposite* otherwise; RE2 comparisons that did not meet their rule are reported with the RE3 result as it is. P2 is replicated if its RE3 lower bound exceeds -0.05 .

4.8 Pre-registration, follow-up studies and execution

The study ran in stages, each committed and tagged in a version-controlled GitHub repository before its own runs (Table 3). The repository was private while the study ran and was made public on 14 October 2026. GitHub’s server-side event log, retrieved on 16 October 2026, records when each tagged commit reached GitHub. For every stage this was before the first timestamp in that stage’s run log, by between 23 seconds and 71 minutes (Table 3). The main study on RE2 followed the frozen protocol. Five follow-up studies were each pre-registered, with their hypotheses, statistics and decision rules, in a dated addition to the protocol that was committed and tagged before any of their runs: the replication on RE3 (Section 4.2), RE3 tests of three exploratory RE2 findings (H5), an ablation of log text and service names (H6), a generic pipeline (H7), and a separation of evidence from prompt with an onset test of the generic pipeline (H8). Each was designed after earlier results had been seen, which Table 3 records, and the new conditions each one tests were run only after it was fixed; comparison arms that already existed, such as `annotated` or `g_full`, are reused and marked as such (for H8, two of the four cells). The six RE3 pilot cases were run before the freeze, with their accuracy not inspected. Three further additions are amendments in the narrow sense: an exact permutation p-value and a list of exploratory analyses (Amendment 1), the repairs needed to run RCAEval’s baselines (Amendment 2), and heuristic variants with a shifted onset, all exploratory (Amendment 4). The repository tags all eight additions `amendment-1` to `amendment-8` in the order they were made.

Table 3 The pre-registration record: the frozen protocol, the five pre-registered follow-up studies and the three amendments, with what had been seen when each was committed. Tags refer to the record repository. Last column: when the tagged commit reached GitHub, from GitHub’s server-side event log (retrieved 16 October 2026), and the first timestamp in the run log of that stage.

Tag	Content	Seen at commit	Pushed / first run (UTC)
<code>protocol-v3-frozen-2</code>	protocol, code, unit constants	pilot on 21 RE1/RE3 cases (accuracy not inspected); no RE2 fault window	10 Oct 11:35 / 12:46
<code>amendment-1</code>	<i>Amendment:</i> exact permutation p ; exploratory analyses X1–X8 and runs E1–E3	confirmatory RE2 results except the RCAEval baselines	10 Oct 15:01 / 15:11
<code>amendment-2</code>	<i>Amendment:</i> baseline execution repairs (Section 4.6); reruns	first-run baseline results; reruns not seen	11 Oct 03:04 / 03:08
execution log	environment changes; rule for methods with no ranking	as above	11 Oct 03:15 / –
<code>amendment-3</code>	<i>Follow-up:</i> RE3 replication; <code>anon_full</code>	all RE2 results except the Amendment 2 reruns; of RE3, only the 6 pilot cases (accuracy not inspected)	11 Oct 03:41:27 / 03:41:50
<code>amendment-4</code>	<i>Amendment:</i> SimpleRCA variants; onset shifted by ± 120 s (exploratory)	all RE2 results except the accuracy of the Amendment 2 reruns and <code>anon_full</code> ; no RE3 accuracy	11 Oct 11:21 / 11:22
<code>amendment-5</code>	<i>Follow-up:</i> RE3 tests of three RE2 exploratory findings (H5)	all RE2 and RE3 replication results; no run of the tested conditions	12 Oct 03:48 / 03:49
<code>amendment-6</code>	<i>Follow-up:</i> log-text ablation and name permutation (H6a–H6c)	every result above, including the review numbers; no run of <code>nolog</code> or <code>permuted</code>	12 Oct 14:44 / 14:51
<code>amendment-7</code>	<i>Follow-up:</i> a generic pipeline (H7a–H7d)	every result above; the generic pipeline not yet built on benchmark data	12 Oct 17:31:44 / 17:32:26
<code>amendment-8</code>	<i>Follow-up:</i> evidence $\times$ prompt; generic onset shift (H8a–H8c)	every result above and an AI-assisted pre-submission review (Declarations); no run of the crossed cells or shifted summaries	13 Oct 06:53 / 06:56

No addition changed a confirmatory hypothesis, statistic or decision rule. One did change confirmatory *data*: the RCAEval rankings in the RQ1 family come from the Amendment 2 reruns, which were specified after the first-run results had been computed and seen, because four baselines had not run at all and two had stopped partway. The amendment fixed the repairs and the scoring rule before any rerun result was seen, and the first-run results are reported as a sensitivity analysis.

The protocol names its freeze tag `protocol-v3-frozen`. That tag (10 October, 11:30 UTC) lacked the protocol’s execution section, the model-run script and two small runner options recorded in the pre-freeze log (a clean stop at the provider’s spending limit and a NOTE-only mode for the reruns). They were committed four minutes later and the complete state was tagged `protocol-v3-frozen-2` (commit 939df23, 11:34 UTC). The first RE2 contexts were built at 12:40 UTC. Both tags are kept, and every later stage refers to the second.

Execution

All runs used one AWS EC2 instance (4 vCPU, 16 GB, Ubuntu 24.04, Python 3.12) between 10 and 13 October 2026, with the package list recorded at the freeze and after each environment change. The pre-registered RE2 model runs made 2,980 calls. Over all suites and conditions, including the pilot and the exploratory runs, the study made 6,743 successful model calls (61 of them schema retries) with 10.9M prompt and 9.0M completion tokens before the text ablation. The text ablation, the generic pipeline and the evidence-prompt study added 1,036, 2,160 and 1,366 responses and 11 schema retries, for 11,316 calls with 17.3M prompt and 11.7M completion tokens in all; three responses reached the output limit. Calls that failed (most of them while an API key was missing) were retried by the resumable runner; counts per run are in the replication package. The repository holds the protocol, every later stage and the execution log, all code, every prompt hash and raw response, and the per-case rankings of every method (Section 8).

5 Results

We report the confirmatory RE2 results (Sections 5.1–5.4), then the pre-registered replication on RE3 (Section 5.5), the exploratory analyses (Section 5.6), and the other four pre-registered follow-up studies (Sections 5.7–5.10). Unless stated otherwise, differences are in AC@1 on all cases of a suite (on the 245 flagged cases where stated), with 95% cluster-bootstrap intervals in brackets; p is the cluster permutation p -value and p_H its Holm-adjusted value within the family. The exact McNemar tests, an unclustered secondary analysis, agree with every pooled RQ1/RQ2 verdict except one: for multi-source BARO, McNemar gives $p = 0.0005$ where the clustered test gives $p_H = 0.061$. All pre-registered measures (AC@1, AC@3, AC@5, Avg@5, per system and pooled, on all and on flagged cases) and the McNemar values are in the replication package.

5.1 Coverage (RQ0)

On RE2, at least one detector fired in 245 of 270 cases: 66 of 90 in Online Boutique, 90 of 90 in Sock Shop and 89 of 90 in Train Ticket. The model was therefore not called for 24 OB cases and one TT case, which receive the chance ranking. Fifty flagged cases (OB 25, SS 14, TT 11) carry at least one natural NOTE; of their 68 NOTE lines, 42 mention the root-cause service. On RE3, 78 of 90 cases were flagged (OB 30, SS 27, TT 21), and 23 carry a NOTE, all but one in Online Boutique. On RE2 the unflagged cases are Online Boutique CPU (7), delay (10), socket (6) and packet-loss (1) faults and one Train Ticket CPU fault. Natural NOTES occur mainly with disk (18 cases), packet-loss (21) and memory (9) faults; the replication package gives coverage per system and fault for both suites.

Table 4 Accuracy on all cases (unflagged cases and cases a method did not rank count at chance). AC@1 for RE2 per system and pooled; Avg@5 pooled. *Rk*: cases with a ranking, out of 270 (RE2) or 90 (RE3). Every method ranks the same candidate set. † MicroCause receives the root service’s latency as its start metric from RCAEval’s harness (Section 4.6). n/r: not runnable on RE3 (no preprocessed log and trace files); these three remain in the RE3 Holm family, scored as chance. n/a: not run on RE3 (secondary, not part of the replication protocol).

Method	RE2						RE3		
	AC@1				Avg@5		AC@1	Avg@5	Rk
	OB	SS	TT	All	All	Rk	All	All	
gpt-oss-120b (deployed)	0.689	0.167	0.467	0.441	0.659	245	0.600	0.747	78
gpt-oss-20b	0.689	0.167	0.467	0.441	0.658	245	0.544	0.713	78
qwen3.8-27b	0.700	0.067	0.411	0.393	0.641	245	0.633	0.793	78
Pipeline: detector count	0.689	0.522	0.578	0.596	0.693	270	0.378	0.589	90
Pipeline: first listed	0.644	0.133	0.444	0.407	0.644	270	0.267	0.556	90
Pipeline: largest anomaly	0.400	0.156	0.022	0.193	0.470	270	0.233	0.447	90
Same-feature classifier (LR)	0.722	0.567	0.589	0.626	0.710	270	0.422	0.549	90
leave one system out	0.622	0.422	0.589	0.544	0.631	270	0.422	0.636	90
random forest (rf_visible)	0.633	0.611	0.622	0.622	0.681	270	0.522	0.593	90
unmasked features (lr_all)	0.833	0.689	0.622	0.715	0.818	270	0.778	0.867	90
SimpleRCA-style	0.744	0.911	0.778	0.811	0.928	270	0.833	0.931	90
N-Sigma	0.789	0.856	0.756	0.800	0.896	270	0.733	0.896	90
N-Sigma (multi-source)	0.800	0.867	0.744	0.804	0.905	269	n/r	n/r	0
BARO	0.144	0.067	0.667	0.293	0.769	270	0.256	0.820	90
BARO (multi-source)	0.167	0.067	0.689	0.307	0.773	269	n/r	n/r	0
CIRCA	0.667	0.778	0.022	0.489	0.599	181	0.444	0.622	61
CIRCA (multi-source)	0.544	0.611	0.022	0.393	0.555	185	n/r	n/r	0
ϵ -Diagnosis	0.033	0.156	0.000	0.063	0.172	270	0.022	0.158	90
MicroCause†	0.089	0.100	0.022	0.070	0.221	48	0.211	0.387	44
CausalRCA (secondary)	0.089	0.133	0.022	0.081	0.207	6	n/a	n/a	–
Chance	0.083	0.091	0.015	0.063	0.189	–	0.062	0.185	–

5.2 Accuracy against the comparators (RQ1, RQ2)

Table 4 gives AC@1 and Avg@5 for every method, and Figure 1 the paired AC@1 differences. On RE2 the deployed model reaches an AC@1 of 0.441. The two other models do not do better (**gpt-oss-20b** 0.441, **qwen3.8-27b** 0.393).

The model against its own pipeline

The pipeline’s detector count, which ranks services by the number of signals that flag them and uses no model and no label, is more accurate than the model on RE2: -0.156 $[-0.219, -0.093]$, $p_H = 0.001$. The gap lies in Sock Shop (-0.356) and Train Ticket (-0.111); in Online Boutique the two are equal (0.689). On flagged cases only (exploratory) the gap is -0.184 $[-0.250, -0.118]$. Ordering services as the **AFFECTED SERVICES** line lists them gives the same accuracy as the model ($+0.033$ $[-0.044, +0.119]$, $p_H = 1.0$).

RQ2 and P2

A logistic regression on the 13 features the prompt shows is more accurate than the model: same-feature classifier minus model = $+0.185$ $[+0.119, +0.256]$. The lower

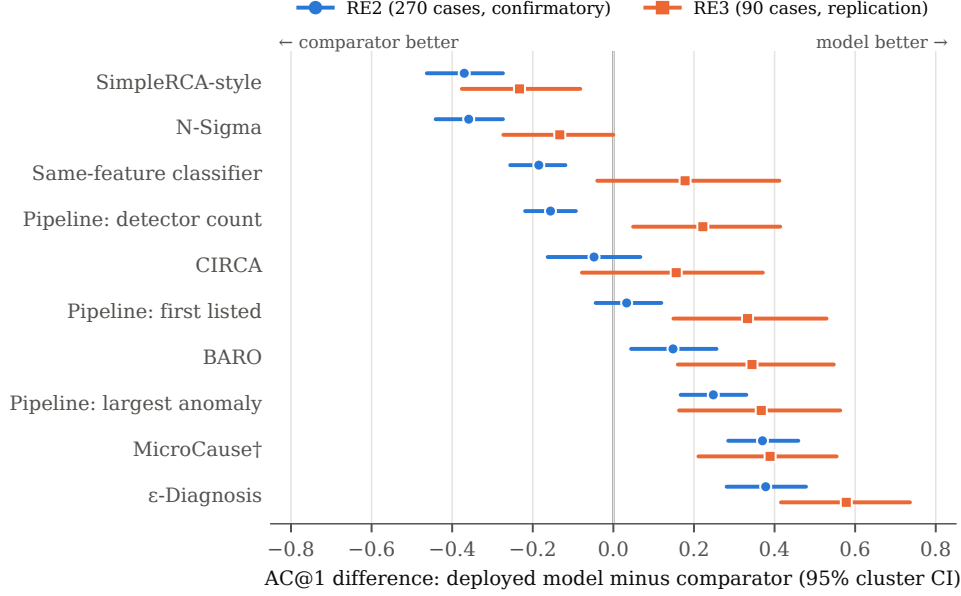


Fig. 1 AC@1 difference between the deployed model and each comparator, with 95% cluster-bootstrap intervals. RE2 is the confirmatory benchmark, RE3 the pre-registered replication. Multi-source baselines are omitted because they cannot run on RE3 (Table 4).

bound exceeds the non-inferiority margin of -0.05 and also zero, so **P2 is supported, and the classifier is superior**. A random forest on the same features gives 0.622. In an exploratory check, the strictest supervised version, trained on two systems and tested on the third, is still ahead of the model ($+0.104$ [$+0.041, +0.167$]). All these classifiers are trained on benchmark labels, which the zero-shot model never sees; the label-free comparison is the detector count above.

RQ1: established methods

Of the 13 pre-registered comparisons, 9 are significant after Holm correction. The model is less accurate than the SimpleRCA-style heuristic (-0.370), N-Sigma (-0.359) and its multi-source variant (-0.363), all with $p_H = 0.001$. It is more accurate than the largest-anomaly ranking ($+0.248$), BARO ($+0.148$, $p_H = 0.045$), ϵ -Diagnosis ($+0.378$) and MicroCause ($+0.370$). It does not differ detectably from CIRCA, multi-source CIRCA, multi-source BARO ($+0.133$, $p_H = 0.061$) or the first-listed order.

Three of these wins need qualifying. ϵ -Diagnosis is at chance pooled (0.063) and scores 0.000 at every cut-off on all 90 Train Ticket cases, below chance (0.015): there it returned only three metrics per case, never including the root (all memory metrics in the cases we inspected). We report its numbers as produced. MicroCause ranked only 48 of 270 cases within the time limit, so 82% of its cases are scored as chance, and it receives a label-derived input. The win over BARO holds only at AC@1: BARO places the root in its top three far more often than the model (AC@3 0.885 against

0.693) and has the higher Avg@5 (0.769 against 0.659), as does multi-source BARO (0.773). Our BARO top-1 on Online Boutique (0.144) equals the 14.4% that [Tian et al. \(2025\)](#) report for the same suite. CIRCA and its multi-source variant did not finish Train Ticket within the two-hour limit, so their TT accuracy is chance.

Before the Amendment 2 reruns

With the first-run baseline rankings, as committed in Amendment 2, the RQ1/RQ2 family had 11 tests, because ϵ -Diagnosis and MicroCause had no ranking at all, and multi-source N-Sigma and BARO had ranked 249 cases. In that family 8 of 11 comparisons are significant after Holm, with the same verdicts as in the confirmatory family for every method the two share but one: the model’s win over multi-source BARO (+0.174, $p_H = 0.012$ with the first run) is no longer significant with the complete rerun (+0.133, $p_H = 0.061$). The reruns thus removed one win for the model and added two, over the two methods that had not run, both near chance.

One vote

On the 27 flagged cases of the vote check, the first vote’s top-1 equalled the unique plurality of five votes in 25. Across three complete runs on the flagged RE2 cases (exploratory, E1), the deployed model’s AC@1 was 0.469, 0.482 and 0.478 (SD 0.006), and the plurality of the three runs scored 0.465.

5.3 Interpretive annotations (RQ3)

P3a: natural NOTES

Removing the NOTES changed the model’s top-1 in 14, 10 and 9 of the 50 NOTE cases (`gpt-oss-120b`, `gpt-oss-20b`, `qwen3.8-27b`). The pre-registered test compares this with the change between two runs of the same annotated text. The differences are +0.100 [−0.021, +0.234], −0.020 [−0.206, +0.163] and +0.140 [+0.055, +0.250], with $p_H = 0.47$, 1.0 and 0.053. **P3a is not supported** for any model: an effect of natural NOTES beyond run-to-run variation was not detected. For `qwen3.8-27b` the result is borderline: the exact permutation p gives $p_H = 0.047$ (Amendment 1). One reason is visible in the control cases. On the 195 flagged cases without a NOTE, the annotated and stripped prompts are byte-identical, yet the top-1 changed in 39 (`gpt-oss-120b`) and 41 (`gpt-oss-20b`) of them; `qwen3.8-27b` changed in 6.

P3b: a false claim

A decoy NOTE claiming that a seeded non-root service is the problem made each model name that service first far more often than without it (Table 5). **P3b is supported for all three models** ($p_H < 0.001$). No model followed the decoy in more than half of the cases, so by the pre-registered rule, which pools the two templates, the decoy did not outweigh the evidence. Split by template, `gpt-oss-20b` named the decoy first in 59 of the 110 cases with a dependency decoy (54%). The decoy still cost accuracy: `gpt-oss-120b`’s AC@1 fell from 0.456 (stripped) to 0.370, and among the 119 cases it got right without the decoy, it named the decoy first in 24. The dependency template

Table 5 Decoy NOTEs on RE2 (245 flagged cases): how often the decoy service is named first. *Harm*: among cases the model gets right under **stripped**, those in which it names the decoy first under **decoy**.

Model	stripped	placebo	decoy	Difference [95% CI]	Harm	Memory / dep.
gpt-oss-120b	9	17	83 (34%)	+0.302 [+0.237, +0.367]	24/119	36/135 / 47/110
gpt-oss-20b	10	–	102 (42%)	+0.376 [+0.313, +0.442]	38/112	43/135 / 59/110
qwen3.8-27b	9	–	52 (21%)	+0.176 [+0.128, +0.229]	8/96	39/135 / 13/110

was more effective than the memory-pressure template for the two **gpt-oss** models, and less effective for **qwen3.8-27b**.

Decoy against placebo (secondary)

A placebo NOTE that names the same service without a claim made **gpt-oss-120b** name it first in 17 of 245 cases, against 83 for the decoy (+0.269 [+0.210, +0.331], $p < 0.001$). The claim, not the mention, moves the top-1. The mention alone still pulls the service up the ranking: it appears among the model’s top five in 76 cases under stripped, 172 under placebo and 224 under decoy.

P3c: agreement

On the NOTE cases, all three models named the same top-1 service in 36 of 50 cases with the NOTEs and 37 of 50 without them (-0.020 , $p = 1.0$). **P3c is not supported**.

5.4 Service names (RQ4)

Replacing service names with neutral labels (**anon**) changed **gpt-oss-120b**’s AC@1 on the flagged cases by +0.045, with a 90% interval of $[-0.000, +0.090]$ ($p = 0.17$). **Equivalence within ± 0.05 is not shown**, and no change is detected (on all 270 cases the difference is +0.040). The confirmatory condition left an original name in 86 of the 245 prompts by the pre-registered count (166 by the strict detector; Section 4.4). In an exploratory split, the difference was -0.013 $[-0.075, +0.051]$ on prompts without a leaked name and +0.151 [+0.047, +0.262] on prompts with one, all of them in Sock Shop.

The exploratory **anon_full** condition left no name in any prompt. No response named a real service. AC@1 on flagged cases rose for all three models: +0.033 $[-0.008, +0.074]$ for **gpt-oss-120b**, +0.094 [+0.049, +0.139] for **gpt-oss-20b**, and +0.057 [+0.021, +0.098] for **qwen3.8-27b** (90% intervals). On RE2, hiding the names therefore did not cost accuracy for any model, and for two models it helped. On RE3 the opposite holds (Section 5.7).

5.5 Replication on RE3

Table 6 applies the pre-registered replication rule (Section 4.7). On RE3 the deployed model is more accurate than on RE2 (0.600). It is above its own detector count (0.378) rather than below it, but this difference is not significant after Holm ($p_H = 0.141$), so the RE2 result does not replicate and RE3 does not establish the reverse. The

Table 6 Replication on RE3 (90 cases). Differences are deployed model minus comparator, except P2 (classifier minus model), P3b (decoy named first under **decoy** minus the same service named first under **stripped**), and decoy vs placebo (decoy named first under **decoy** minus under **placebo**, deployed model, single test). Holm within each family. The three multi-source baselines cannot run on RE3; they stay in the 13-test family scored as chance, which does not change any other adjusted p , and are shown in one row as not interpretable.

Quantity	RE2	RE3	RE3 p_H	Verdict
vs detector count	-0.156	+0.222 [+0.049, +0.414]	0.141	opposite direction, n.s.
vs same-feature classifier	-0.185	+0.178 [-0.040, +0.412]	0.319	opposite direction, n.s.
vs SimpleRCA-style	-0.370	-0.233 [-0.376, -0.082]	0.0525	same direction, n.s.
vs N-Sigma	-0.359	-0.133 [-0.273, +0.000]	0.284	same direction, n.s.
vs largest anomaly	+0.248	+0.367 [+0.163, +0.563]	0.020	replicated
vs BARO	+0.148	+0.344 [+0.160, +0.547]	0.018	replicated
vs ϵ -Diagnosis	+0.378	+0.578 [+0.416, +0.736]	0.001	replicated
vs MicroCause†	+0.370	+0.389 [+0.211, +0.554]	0.005	replicated
vs CIRCA	-0.048	+0.156 [-0.078, +0.371]	0.319	RE2 n.s.; RE3 n.s.
vs first listed	+0.033	+0.333 [+0.149, +0.529]	0.022	RE2 n.s.; RE3 significant
vs 3 multi-source methods	-	chance	0.001	not interpretable
P2 (LR minus model)	+0.185	-0.178 [-0.412, +0.040]	-	not replicated (inconclusive)
LR trained on RE2	-	-0.133 [-0.352, +0.070]	-	-
P3b gpt-oss-120b	+0.302	+0.141 [+0.039, +0.259]	0.041	replicated (15/78 decoy first)
P3b gpt-oss-20b	+0.376	+0.295 [+0.165, +0.434]	0.001	replicated (28/78)
P3b qwen3.8-27b	+0.176	+0.115 [+0.040, +0.195]	0.029	replicated (12/78)
Decoy vs placebo	+0.269	+0.128 [+0.027, +0.241]	-	replicated ($p = 0.0495$)

decoy result replicates for all three models and against the placebo. Excluding the six pilot cases moves no point estimate by more than 0.03 and changes no verdict; only N-Sigma’s upper bound moves, from 0.000 to +0.022 (n.s. either way). The exact sign-flip tests (Amendment 1) agree with the Monte Carlo verdict for every row, so none is borderline: for the SimpleRCA-style heuristic the exact p_H is 0.055, for decoy against placebo the exact p is 0.047, and for P3b with **gpt-oss-120b** the exact p_H is 0.039.

The difference from RE2 comes from Online Boutique. There the model reaches 0.900 against 0.200 for the detector count and 0.167 for the same-feature classifier. In Train Ticket the model is correct on all 21 flagged cases. In Sock Shop it is weaker (0.200) than the detector count (0.300) and the classifier (0.467). Neither the classifier cross-validated on RE3 nor the one trained on RE2 matches the model, so **P2 does not replicate**. The model is not ahead of every metric-based method on RE3, however: N-Sigma (0.733), the SimpleRCA-style heuristic (0.833) and the classifier on unmasked detector features (0.778) all score above it. What falls short on RE3 is the pipeline’s detector count and the features the prompt shows, not metric evidence as such.

On the decoy, no model named the decoy service first in more than half of the RE3 cases (19%, 36% and 15%). Among the cases each model got right under **stripped**, it named the decoy first in 3 of 52 (**gpt-oss-120b**), 16 of 52 (**gpt-oss-20b**) and 4 of 56 (**qwen3.8-27b**). With only 23 NOTE cases, P3a and P3c are descriptive on RE3: removing the NOTEs changed the top-1 in 4, 6 and 0 of them, and the three models agreed on the top-1 in 16 of the 23 with the NOTEs and 20 without them.

5.6 Exploratory analyses

Everything in this section is post hoc (Amendments 1 and 4) and is reported as such.

An always-on alarm

In Sock Shop the log detector flags `queue-master` in all 90 cases. The alarm is chronic: in 87 of the 90 cases its error count does not exceed twice its count in the five minutes before the injection. The models name it first in 69, 70 and 82 of the 90 cases. Train Ticket has a similar alarm (`ts-inside-payment-service`, flagged in 79 of 89), which attracts the models less (18, 28 and 41). When log alarms that already fired before the injection are suppressed (E3), 203 cases remain flagged. `gpt-oss-120b`'s AC@1 rises from 0.441 to 0.600 (+0.159 [+0.096, +0.222]), from 0.167 to 0.589 in Sock Shop alone. The detector count barely moves (0.596 to 0.604). With these alarms suppressed, the model (0.600) and its own detector count (0.604) are level.

Position in the context

The model's preference for the first-listed service reflects the services it tends to choose, not their position (Appendix B).

The SimpleRCA-style heuristic

Our heuristic scores far above the values Fang et al. (2026b) publish for Online Boutique and Sock Shop (0.47 and 0.24) and close to theirs for Train Ticket (0.80). Ties do not explain the gap: in the diagnostic of Amendment 1 (X8), breaking every tie at random gives an expected AC@1 of 0.815. Table B2 (Appendix B) isolates the three implementation choices. Neither dropping the $\sigma = 0$ rule nor ignoring `-db` columns matters much. Counting at most one alert per signal type, rather than one per metric as Fang et al. describe, lowers AC@1 to 0.352 on RE2 and 0.500 on RE3. (Amendment 4 called this variant the per-modality form of Fang et al.'s heuristic; on re-reading, their description counts an alert per metric, as our pre-registered form does.) This brings Online Boutique and Sock Shop close to their values but opens a gap in Train Ticket (0.411 against 0.80), so it does not reproduce their numbers. On RE3, where Fang et al. report 0.30 for Online Boutique and 0.83 for Sock Shop and Train Ticket, our pre-registered form scores 0.833 overall. With it, the model is no longer behind (+0.089 [−0.004, +0.193] on RE2). The pre-registered form stays our comparator. RCAEval's own N-Sigma, which also scores each metric column, reaches 0.800.

When the onset is not known exactly

Every method, the model included, is told the injection time. Figure 2 shifts that time by two minutes in each direction. Pham et al. (Pham et al. 2024b) already showed that N-Sigma and BARO are sensitive to the onset time; we extend the check to the pipeline, the model and the heuristic. Placing the onset two minutes early changes no method by more than 0.04. Placing it two minutes late, so that the baseline window contains the start of the fault, collapses the strongest statistical methods: N-Sigma falls from 0.800 to 0.070 and the SimpleRCA-style heuristic from 0.811 to 0.230. The pipeline and the model degrade less (detector count 0.596 to 0.378, model

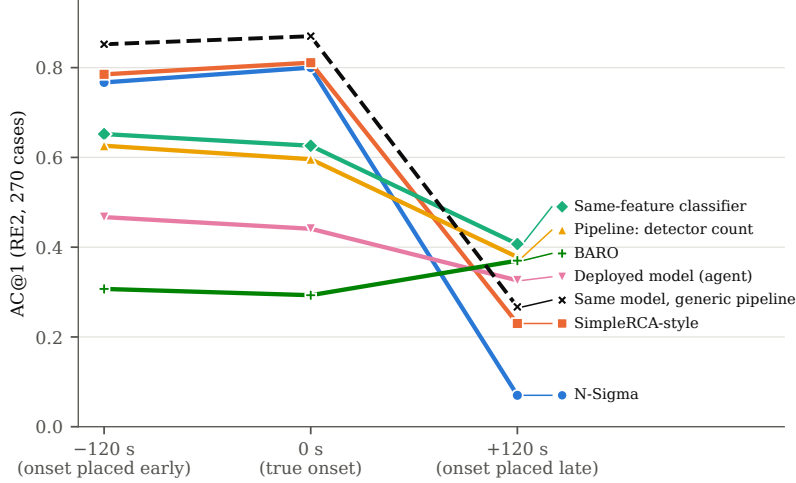


Fig. 2 AC@1 on RE2 when the injection time given to every method is shifted by -120 s and $+120$ s (exploratory). The pipeline-based methods and the model are rebuilt and re-run on each shifted window; N-Sigma and BARO use RCAEval’s `--tdelta`. Dashed: the same model in the generic pipeline under the same shifts (pre-registered follow-up, H8c; Section 5.10).

0.441 to 0.326), and BARO improves (0.293 to 0.370). At $+120$ s the model is ahead of N-Sigma ($+0.256$ [$+0.167$, $+0.348$]) and level with the SimpleRCA-style heuristic ($+0.096$ [-0.004 , $+0.196$]), level with its own detector count (-0.052 [-0.119 , $+0.019$]), and behind the same-feature classifier (-0.081 [-0.156 , -0.007]). The pipeline’s own sensitivity is partly a product of the port: offline, its baseline is the window before the given onset, whereas the live system uses a separate fault-free hour (Table 1).

5.7 Pre-registered tests of three RE2 findings on RE3

A follow-up study turned three exploratory RE2 findings into directional hypotheses (H5) and tested them on RE3 conditions that had not been run (Table 7). Excluding the six pilot cases changes no verdict (all three Holm-adjusted p -values are unchanged).

Onset (H5c)

The pre-registered test confirms, with large margins, that a late onset costs the strongest statistical baselines far more than the model. On RE3 the pipeline’s own detector count is fragile too (0.378 to 0.089). The model (0.600 to 0.389) ends ahead of N-Sigma (0.089), the SimpleRCA-style heuristic (0.267), the detector count and the same-feature classifier (0.233), and close to BARO (0.356), which again gains from the late onset. Placing the onset two minutes early changes less, except for the SimpleRCA-style heuristic (0.833 $\rightarrow$ 0.711) and N-Sigma (0.733 $\rightarrow$ 0.667).

Always-on alarms (H5a)

H5a assumed an always-on alarm like Sock Shop’s `queue-master`; we did not check whether RE3 has one. Suppressing pre-existing log alarms raised the model’s AC@1

Table 7 Pre-registered tests on RE3 of three RE2 findings (H5), all 90 cases, deployed model. H5a and H5c are differences in differences of per-case top-1 correctness; Holm over H5a, H5c-1 and H5c-2. H5b is a non-inferiority rule (lower bound above -0.05). ^a On the 245 flagged RE2 cases with a 90% interval; the RE3 test uses all 90 cases and a 95% interval.

Hypothesis	RE2 finding	RE3 result	Verdict
H5a: suppressing pre-existing log alarms helps the model more than its detector count	model +0.159, detectors +0.007	model +0.111, detectors +0.056; DiD +0.056 $[-0.036, +0.165]$, $p_H = 0.38$	not supported
H5b: hiding every name does not lower accuracy by more than 0.05	+0.033 (model) ^a	-0.211 $[-0.379, -0.065]$	not supported (reversed)
H5c-1: N-Sigma loses more than the model when the onset is 120 s late	-0.730 vs -0.115	0.733 $\rightarrow$ 0.089 vs 0.600 $\rightarrow$ 0.389; DiD +0.433 $[+0.258, +0.607]$, $p_H < 0.001$	supported
H5c-2: the same for the SimpleRCA-style heuristic	-0.581 vs -0.115	0.833 $\rightarrow$ 0.267; DiD +0.356 $[+0.253, +0.464]$, $p_H < 0.001$	supported

from 0.600 to 0.711, but the detector count also rose, and the difference in differences is not significant. H5a is not supported.

Names (H5b)

Hiding every spelling of every service name *lowered* the deployed model’s accuracy on RE3 by 0.211; descriptively, the two other models lost 0.167 each ($p = 0.068$ and 0.046, not part of the test). H5b is not supported, and the effect is in the opposite direction to RE2. The design does not say why. Because the labels are consistent, a log line that named the failing service still names its label under `anon_full`: the log samples name the root in 37 of the 78 flagged RE3 cases (OB 18, SS 19, TT none), and in all 37 its label remains; what is removed is what the name means, for example that `RouteServiceImpl` belongs to `ts-route-service`. The drop is therefore consistent with the model losing a link between code identifiers and services, which is evidence it could read, as well as with it losing prior knowledge of these systems. The follow-up study H6 tests this directly (Section 5.8).

5.8 Log text and service names on RE3 (pre-registered follow-up, H6)

This follow-up study tested, before any run, whether the model’s RE3 accuracy comes from raw log text, and what service names contribute (Section 4.4). Table 8 gives the four pre-registered tests and Table 9 the accuracy under each condition.

Log text (H6a, H6b)

Removing the raw log text, while keeping everything the log detector reports, lowered the deployed model’s RE3 accuracy from 0.600 to 0.356. **H6a is supported**. The two other models lost as much (0.244 and 0.222, descriptive). On RE2 the same removal changed nothing (+0.015 $[-0.041, +0.070]$), and the difference between the suites is significant (**H6b is supported**). On RE3 the model therefore uses the raw log text. Descriptively, without it the model is level with its own detector count

Table 8 Pre-registered tests H6 (deployed model, AC@1 on all cases; Holm over the four tests). H6b compares the per-case effect of removing log text between suites; its clusters are independent, so it uses a cluster bootstrap within each suite and a permutation of suite labels among clusters. Exact p_H : the same family with the exact sign-flip p (not defined for H6b, which keeps its Monte Carlo p).

Test	Difference [95% CI]	p_H	exact p_H	Verdict
H6a: RE3, <code>annotated</code> – <code>nolog</code>	+0.244 [+0.107, +0.409]	0.016	0.012	supported
H6b: the same, RE3 minus RE2	+0.230 [+0.069, +0.394]	0.005	0.005	supported
H6c-1: RE3, <code>permuted</code> – <code>anon_full</code>	+0.011 [−0.066, +0.080]	1.0	1.0	no difference detected
H6c-2: RE3, <code>annotated</code> – <code>permuted</code>	+0.200 [+0.056, +0.363]	0.049	0.045	correct names help

Table 9 AC@1 on all cases under the text conditions of the RE3 replication and H6. RE2 `nolog` and `permuted` were run for the deployed model only.

Suite	Model	<code>annotated</code>	<code>nolog</code>	<code>permuted</code>	<code>anon_full</code>
RE2	<code>gpt-oss-120b</code>	0.441	0.426	0.474	0.470
RE3	<code>gpt-oss-120b</code>	0.600	0.356	0.400	0.389
RE3	<code>gpt-oss-20b</code>	0.544	0.300	0.378	0.378
RE3	<code>qwen3.8-27b</code>	0.633	0.411	0.367	0.467

(−0.022 [−0.132, +0.080]), where with it the model was 0.222 ahead (not significant after Holm, Table 6): the model’s advantage over its detectors disappears without the log text. The study predicted, descriptively, that the drop would be concentrated in the cases whose log samples name the root. That was only partly borne out: the drop was larger there (37 flagged cases: 0.622 to 0.297) but clear also where they do not (41 cases: 0.756 to 0.512).

Service names (H6c)

With the names deranged, the deployed model’s RE3 accuracy fell to 0.400, below the true names by 0.200 (H6c-2, $p_H = 0.049$; exact $p_H = 0.045$), so names that denote the right service matter. No difference was detected between deranged names and neutral labels (0.389; H6c-1), so we cannot say that the look of a name helps or hurts on its own; with about 30 clusters, a difference smaller than about 14 points would go undetected. The other models agree in direction (descriptively, +0.167 and +0.267 for H6c-2); for `qwen3.8-27b`, wrong names did worse than labels (−0.100). On RE2 deranging the names made no difference (−0.033, n.s.).

Where the correspondence pulls is visible in the answers. Under `permuted`, the models named first the wrong service that now carries the root’s real name in 18, 20 and 17 of the 78 flagged RE3 cases, against about 5 expected by chance; on RE2 the deployed model did so in 10 of 245 cases, against about 15 expected. On RE3 the model follows the name when name and service are pulled apart. This fits a model that matches a service’s name to what the rest of the prompt says, such as the ordinary words of an error message that the derangement leaves in place; prior knowledge of

Table 10 Pre-registered tests H7 (generic pipeline, **gpt-oss-120b**, AC@1 on all cases; decoy tests on all cases with a decoy; Holm over the four tests). H7c is a between-suite test as H6b.

Test (re-tests)	Difference [95% CI]	p_H	exact p_H	Verdict
H7a: decoy named first, g_decoy – g_full , RE2+RE3 (P3b)	+0.586 [+0.529, +0.643]	< 0.001	< 0.001	supported
H7b: RE3, g_full – g_nolog (H6a)	+0.067 [−0.011, +0.148]	0.53	0.53	not supported
H7c: the same, RE3 minus RE2 (H6b)	+0.059 [−0.021, +0.150]	0.53	0.53	not supported
H7d: RE2, summary_count – g_full (RQ1/RQ2)	−0.044 [−0.104, +0.015]	0.53	0.53	not supported

Table 11 AC@1 on all cases of the same models in the two pipelines, with the label-free ranking of each pipeline’s own output and the strongest baselines.

	RE2		RE3	
	gpt-oss-120b	qwen3.8-27b	gpt-oss-120b	qwen3.8-27b
Agent pipeline (annotated)	0.441	0.393	0.600	0.633
Generic pipeline (g_full)	0.870	0.893	0.889	0.889
without log examples (g_nolog)	0.863	0.922	0.822	0.811
with a decoy sentence (g_decoy)	0.341	0.704	0.411	0.711
Detector count (deployed pipeline, no model)	0.596		0.378	
summary_count (generic pipeline, no model)	0.826		0.800	
N-Sigma / SimpleRCA-style	0.800 / 0.811		0.733 / 0.833	

these open-source systems would act the same way, and the design does not separate the two.

Robustness

A second annotated run on RE3 changed the top-1 in 7 of 78 flagged cases and AC@1 by -0.033 (n.s.), a fraction of the effects above. Excluding the six pilot cases changes no verdict (p_H 0.019, 0.007, 1.0 and 0.036). The permuted prompts passed the name-free comparison in every case, and no answer failed to map back to a service. The **nolog** check found one remaining log string in 3 of the 78 RE3 prompts: in the three **adservice** f5 cases, the mined template “failed to retrieve ads” stayed in the summary’s reason line because our code did not recognise that line as part of the log section. The sample lines and the anomalous-logs block were removed there as elsewhere. Any bias from these cases runs against H6a; without them every verdict holds (H6a +0.253, $p_H = 0.009$; H6b +0.238, $p_H = 0.004$; H6c-2 +0.207, $p_H = 0.041$; H6c-1 +0.011, $p_H = 1.0$).

5.9 A generic pipeline (pre-registered follow-up, H7)

Table 10 gives the four pre-registered tests in the generic pipeline and Table 11 the accuracy of the same two models in both pipelines.

The same model in two pipelines (descriptive)

With the generic summary, the same deployed model reaches 0.870 on RE2 and 0.889 on RE3, against 0.441 and 0.600 in the deployed agent (+0.430 and +0.289, both $p < 0.02$; **qwen3.8-27b** +0.500 and +0.256). These comparisons were pre-registered as descriptive. They measure whole pipelines, which differ in prompt, answer format and gate as well as in the evidence they show; the generic summary was designed after the earlier results, resembles the per-column statistics of N-Sigma, and, like them, depends on knowing the onset (Section 5.10). On RE2 it is level with or slightly above N-Sigma (+0.070 [+0.000, +0.144]) and the SimpleRCA-style heuristic (+0.059 [-0.004, +0.122]), descriptively. The model was not detectably behind the label-free ranking of its own input: `summary_count` minus model is -0.044 [-0.104, +0.015] (**H7d not supported**); for **qwen3.8-27b** the model is ahead (+0.067 [+0.015, +0.119], descriptively). By the pre-registered rule we report the RE2 finding that the model trails its own pipeline as specific to the deployed agent.

Log text

In the generic pipeline, removing the example error lines changed RE3 accuracy by only 0.067 [-0.011, +0.148] (**H7b not supported**), and the difference from RE2 is not significant either (H7c). Without the log lines the model on RE3 is level with `summary_count` (+0.022, n.s.), much as in the agent. The effect of log text was thus not detected in the generic pipeline, and by the pre-registered rule we report the H6a finding as specific to the deployed agent. The point estimate has the same sign as in the agent and is smaller, but we did not test whether the two pipelines differ. One reading is that the generic summary passes on RE3’s metric signal, which the agent’s detectors largely lose (Section 5.5).

A false claim

The decoy effect holds. One sentence naming a wrong service as the root cause made the deployed model name that service first in 168 of 270 RE2 cases (62%) and 48 of 90 RE3 cases (53%), against 3 and 2 without it (**H7a supported**, $p_H < 0.001$). Among the cases it answered correctly without the sentence, it named the decoy first in 144 of 235 (RE2) and 41 of 80 (RE3). Applying the P3b rule post hoc, in this pipeline the decoy outweighs the evidence for **gpt-oss-120b**. **qwen3.8-27b** was less susceptible (65 of 270 and 26 of 90). The generic sentence asserts the cause more directly than the deployed builder’s NOTE templates, so the size of the effect is not comparable between pipelines; its direction is the same. Excluding the six pilot cases changes no verdict.

5.10 Evidence or prompt, and the onset (pre-registered follow-up, H8)

Table 12 gives the three pre-registered tests and Table 13 the four cells of the 2×2 and the generic pipeline under a shifted onset.

Table 12 Pre-registered tests H8 (**gpt-oss-120b**, AC@1; H8a and H8b on the 323 flagged cases of RE2 and RE3, H8c on all 360 cases; Holm over the three tests).

Test	Difference [95% CI]	p_H	exact p_H	Verdict
H8a: evidence effect, generic – deployed user message	+0.296 [+0.225, +0.370]	< 0.001	< 0.001	supported
H8b: prompt effect, generic – deployed system prompt (two-sided)	+0.051 [+0.020, +0.083]	0.003	0.003	generic prompt better
H8c: g_full , given onset – onset 120 s late	+0.583 [+0.510, +0.653]	< 0.001	< 0.001	supported

Table 13 Left: AC@1 of **gpt-oss-120b** in the four cells, on the deployed agent’s flagged cases (E: evidence, the user message; P: prompt, the system message; d: deployed, g: generic). Right: **g_full** on all cases with the onset as given and shifted, beside other methods at +120 s (Figure 2 and Table 7).

	n	$E_d P_d$	$E_d P_g$	$E_g P_d$	$E_g P_g$		RE2	RE3
RE2	245	0.469	0.547	0.808	0.865	g_full , given onset	0.870	0.889
RE3	78	0.692	0.692	0.885	0.885	onset –120 s	0.852	0.711
Both	323	0.523	0.582	0.827	0.870	onset +120 s	0.267	0.367
						Deployed model, +120 s	0.326	0.389
						N-Sigma, +120 s	0.070	0.089
						SimpleRCA-style, +120 s	0.230	0.267

Evidence, not prompt

With the deployed system prompt, swapping the agent’s user message for the generic one raised accuracy on the flagged cases from 0.523 to 0.827; with the generic system prompt, from 0.582 to 0.870. The evidence effect is +0.296 (**H8a supported**). The prompt effect is also detectable but about six times smaller: +0.051 (**H8b**, generic prompt better), all of it on RE2 (+0.067; RE3 0.000). No interaction was detected (−0.015 [−0.075, +0.047]). Per suite, the evidence effect is +0.329 on RE2 and +0.192 on RE3. Under the deployed prompt and its answer schema, the generic evidence reaches 0.808 on RE2’s flagged cases; most of the difference between the two pipelines is therefore carried by what each shows the model, and a smaller part by how it is asked. The two user messages still differ in many ways at once (which signals are summarised, how, and with which annotations), so this test locates the difference in the evidence without saying which part of it.

The generic pipeline depends on the onset

With the onset two minutes late, the generic pipeline falls from 0.870 to 0.267 on RE2 and from 0.889 to 0.367 on RE3 (**H8c supported**, +0.583). It thus shares the fragility of N-Sigma and the SimpleRCA-style heuristic, whose statistics it resembles: at +120 s it is ahead of both, and descriptively close to or below the deployed agent’s model (0.326 and 0.389; not tested), whose detector baseline is less tied to the given onset, and below BARO on RE2 (0.370). With the onset two minutes early, it loses little on RE2 (0.852) and more on RE3 (0.711; pooled +0.058 [+0.014, +0.108], descriptive). Its accuracy in Section 5.9 is therefore conditional on an exact onset, like that of the

strongest baselines. Excluding the six pilot cases changes no verdict (H8a +0.294, H8b +0.052, H8c +0.582, all $p_H < 0.002$).

6 Discussion

6.1 Where the accuracy comes from

With the deployed model on RE2, the model was less accurate than its own detector count in the agent (confirmatory) and not detectably different from the ranking implied by the generic pipeline’s summary (H7d), which was itself close to the best statistical baselines. Elsewhere the model was descriptively ahead of these label-free rankings, but not by much: by 0.22 on RE3 in the agent (not significant after Holm) and by 0.04 to 0.09 in the generic pipeline (0.07 for **qwen3.8-27b** on RE2, $p = 0.03$ unadjusted). The model’s accuracy moved with what its pipeline extracted far more than it moved beyond it: the same model reached 0.44 on RE2 in one pipeline and 0.87 in the other (descriptive). A pre-registered 2×2 that crossed the two pipelines’ evidence and prompts located most of that difference in the evidence (H8a, +0.30 on the flagged cases) and a smaller part in the prompt (H8b, +0.05). Under the deployed agent’s own prompt and answer schema, the generic evidence raised the model from 0.47 to 0.81 on RE2’s flagged cases. In this study, what a pipeline extracted mattered far more than how the model was asked. Two limits apply: the generic summary was designed after we had seen how well per-column statistics do on RCAEval, and, like them, its accuracy holds only when the onset is known exactly (H8c).

Relation to LLM re-rankers

DecompRCA and GALA report that an LLM improves top-1 over the best statistical baseline when it re-ranks a statistical method’s candidates (Muhammad et al. 2026; Tian et al. 2025), whereas our model was at most modestly ahead of a label-free reading of its input. The settings differ in what the model is given. Their LLMs receive candidates already narrowed by a strong ranker, together with the evidence for each; our agent receives a detector summary in which a ranking is implicit, and our baseline is that ranking itself. One reading, which we did not test, is that an LLM adds most when it adjudicates among a short list of well-supported candidates, and least when the detectors have already compressed the evidence into a ranking. The two lines of work are complementary: theirs shows what an LLM can add on top of a statistical ranker, ours how much remains once the label-free reading of the model’s own input is the baseline.

Within the deployed agent, the picture differs between the two suites.

RE2: the model is less accurate than its own detectors

RE2’s resource and network faults show up as deviations in metrics. The pipeline’s detectors capture these deviations as a handful of numbers. Given those numbers, counting the detectors that fire localises the root cause better than the model, and so does a logistic regression trained on them (confirmatory). Cues in the text pull the model away from that evidence: a decoy NOTE does so (confirmatory), and in an

exploratory analysis so did an alarm that fires in every Sock Shop case, although the pre-registered RE3 re-test of that account (H5a) was not supported. This matches the earlier, not yet reviewed live study of the same system (Hussain and Farrukh 2026b): outside the crash scenario a classifier on the same features matched every model, and the crash verdicts depended on interpretive NOTES. The shortfall was not detected in the generic pipeline (H7d), so by the pre-registered rule it is specific to the deployed agent.

RE3: the model uses the logs

On RE3’s code-level faults the detector count reaches only 0.38 and the model 0.60. That difference is not significant after Holm, so RE3 does not replicate the RE2 result, but the pre-registered log ablation (H6) shows that the model uses the raw log text: without it, and with everything the log detector reports still in the prompt, the model falls to 0.36, and descriptively its non-significant lead over its detectors disappears; on RE2 no change was detected. This is not a better reading of the metrics: N-Sigma (0.73), the SimpleRCA-style heuristic (0.83) and a classifier on the unmasked detector features (0.78) still score higher on RE3, so the pipeline loses metric signal that the model cannot recover from the prompt. In the generic pipeline, where the metric deviations reach the model and it scores 0.89 on RE3, the effect of removing the log lines was not detected (H7b), with a smaller point estimate of the same sign. One reading is that the model leans on log text when the metric summary is weak; we did not test whether the two pipelines differ.

Names matter on RE3 too. Hiding the names lowered RE3 accuracy (H5b), and so did deranging them (H6c-2, by 0.20); no difference was detected between deranged names and neutral labels (H6c-1). The model therefore uses which service a name denotes. Two mechanisms fit this: matching a name to the ordinary words of an error message (a payment error and a service called *payment*), and prior knowledge of these open-source systems. Our design does not separate them. On RE3 the model scored lower when names carried no meaning (H5b); on RE2, hiding names did not lower accuracy for any model.

What this implies

The end-to-end accuracy of a detector-to-LLM pipeline mixes at least three things: the signal its detectors extract, the signal the prompt passes on, and the model’s response to whatever the text asserts. In our agent these pulled in different directions on the two suites: on RE2 the text mostly cost accuracy, and on RE3 it supplied all of the model’s (non-significant) advantage over its detectors. A different context for the same model changed the picture again. Evaluations that report only the end-to-end number of one pipeline cannot tell these apart, and they credit to the model what belongs to the context.

6.2 Interpretive text is a lever on the answer

The decoy result is the most robust in the study. It was pre-registered, holds for three models on two benchmark suites, and holds in a second, generic pipeline: there one sentence asserting a wrong root cause made **gpt-oss-120b** name that service first

in 62% of all RE2 cases and turned 144 of its 235 correct answers into wrong ones (H7a; `qwen3.8-27b`, 24%). The sentence differs from the agent’s NOTES, so the sizes are not comparable. A placebo that names the same service without a claim has a much smaller effect, so the model follows the claim, not merely the mention. In the deployed agent, pooled over the two templates, no model followed the decoy in a majority of cases, but `gpt-oss-20b` did follow dependency decoys in 54% of them. In the deployed agent, a false assertion turned correct answers into wrong ones in up to a third of the cases a model would otherwise have got right (`gpt-oss-20b`, 38 of 112 on RE2; 16 of 52 on RE3).

For practitioners this has a direct consequence. Pipelines that write interpretations into the model’s context are, in effect, writing part of the answer. In our system the NOTES were added during development to fix answers the model had got wrong, and their wording names the expected cause and even the expected verdict. Such text should be treated like code. It should be tested by ablation (remove it), by counterfactuals (assert a wrong cause), and against a placebo, before its effect on accuracy is credited to the model. The same applies to retrieval and tool outputs that make claims rather than report measurements. [Luo et al. \(2026\)](#) report the converse risk as well, agents that override correct diagnostic recommendations; either way, the effect of such text has to be measured rather than assumed.

Natural NOTES, by contrast, did not change answers detectably more often than repeated runs did (P3a not supported; borderline for `qwen3.8-27b`). Two things contribute. Many natural NOTES point at the true root (42 of 68 lines), so removing them changes little. And the `gpt-oss` models change their top-1 between identical calls in about one case in five, which makes small effects hard to see.

6.3 What the strongest baselines rely on

On RE2 with the exact injection time, N-Sigma and our SimpleRCA-style heuristic are far ahead of every other method. Three findings qualify that lead.

1. It depends on an implementation choice. Counting an alert per metric column, as [Fang et al. \(2026b\)](#) describe, rather than at most one per signal type, moves the heuristic from 0.35 to 0.81. The per-signal-type variant comes close to the values they publish for Online Boutique and Sock Shop but not for Train Ticket, so neither form reproduces all three.
2. It depends on knowing the onset, as [Pham et al. \(2024b\)](#) showed for N-Sigma and BARO. When the onset is placed two minutes late, so that the baseline window already contains the fault, N-Sigma and the heuristic lose most of their accuracy, while the model loses far less. We found this in an exploratory RE2 analysis and confirmed it with a pre-registered test on RE3.
3. The pipeline’s own detector count is also fragile to a late onset, especially on RE3. In our port this is partly by construction, because its baseline is the window before the given onset.

The generic pipeline matched these baselines, and it shares their dependence on the onset: two minutes late, it fell from 0.87 to 0.27 on RE2 and from 0.89 to 0.37 on RE3 (H8c, pre-registered), close to or below the deployed agent’s model under the

same shift (0.33 and 0.39, descriptive). Of the model’s contexts, the evidence that served it best at the given onset is thus the most fragile when the onset is wrong. A pipeline designer faces the same trade-off as a baseline author: windows compared around a known onset are sharp when the onset is right.

Benchmarks that hand every method the exact injection time favour methods that compare a clean pre-fault window with a post-fault window. A live system never knows the exact time, so for N-Sigma and the heuristic the standard setting is optimistic; BARO, in contrast, gained from a late onset in both suites. We recommend that RCA benchmark studies report accuracy under a misplaced onset alongside the standard setting, and over more than the single shift we tested.

6.4 Recommendations for evaluating LLM-based RCA

These recommendations come from one agent, a single-call pipeline in which detectors summarise telemetry and one LLM reads the summary. Agents that call tools or iterate may behave differently, but every one of them can be checked in the same way. From these results we draw six recommendations for studies that evaluate LLM-based RCA:

1. Compare the model with a label-free reading of its own input, and evaluate the same model with more than one way of building its context, including a plain summary of the raw evidence. In our study the two pipelines differed by 0.43 AC@1 for the same model, far more than any choice of model, and crossing their evidence and prompts showed that the evidence carried most of it.
2. Report the accuracy of the pipeline without the model and of a simple classifier on the model’s own inputs. The model’s contribution is the difference, not the end-to-end number.
3. Ablate every piece of interpretive text the pipeline adds, and test it with a decoy and a placebo.
4. Report results per system and per fault type. The agent-versus-pipeline result on one suite did not replicate on the other, in line with the warning of [Hu et al. \(2026\)](#) about pooled leaderboards.
5. Report sensitivity to the onset time for every method, including the pipeline that builds the model’s context.
6. Ablate the raw text the model reads (logs, traces, tool output) separately from the numbers, and treat anonymisation with care. Hiding names and permuting them answer different questions: only a permutation shows whether a model uses what a name denotes rather than merely that it is a name.

These recommendations complement the general guidelines for empirical studies involving LLMs ([Wagner et al. 2025](#); [Baltes et al. 2025](#)).

7 Threats to Validity

We organise the threats following empirical-standards guidance ([Ralph et al. 2020](#)) and the threats specific to studies that use LLMs ([Sallou et al. 2024](#); [Wagner et al. 2025](#); [Baltes et al. 2025](#)).

Construct validity

We measure service-level AC@1 with RCAEval’s own evaluator. It does not judge the model’s explanation, which can be wrong even when the service is right (Fang et al. 2026a), and the ground truth is the injected service. The offline port differs from the deployed system in the ways listed in Table 1. The two that matter most are that the restart and CPU-saturation NOTES cannot fire on RCAEval, so RQ3 covers memory-pressure and dependency NOTES, and that the prompt has no topology hints. We used one vote instead of five; on 27 flagged cases the first vote matched the plurality of five in 25. One RE3 metric file set contains services of a second system; every method ranks the same candidates there, but chance is lower.

Internal validity

The system and the study are ours. Against favourable choices, we pre-registered the questions, predictions, statistics and decision rules, pre-registered each follow-up study before its runs, and report every confirmatory result, including the nulls (P3a, P3c, RQ4), the result that did not replicate (P2 on RE3) and the pre-registered tests that failed (H5a, H5b, H6c-1, H7b–H7d). GitHub’s server-side event log shows that every stage’s commit reached GitHub before that stage’s first run (Table 3). Because the repository was private during the study and the run logs are written by our own machine, this establishes the order of commits and runs, not the absence of other analyses. Each follow-up study was designed with earlier results in view, as Table 3 records, and tests new conditions or a new suite. The confirmatory RCAEval rankings come from reruns specified after the first-run results were seen (Amendment 2); with the first-run rankings every shared verdict is the same except one, a win for the model that the reruns removed (Section 5.2).

RCAEval’s baselines needed small repairs to run as pinned (Amendment 2), and methods we could not run are reported as not runnable rather than reimplemented. Where a baseline did not finish within the two-hour limit, its accuracy is chance: MicroCause ranked 48 of 270 RE2 and 44 of 90 RE3 cases, and CIRCA and its multi-source variant did not finish Train Ticket. RCAEval’s harness also gives MicroCause the root service’s own latency as a starting point, and on Train Ticket ϵ -Diagnosis never returned the root in any case, so the model’s wins over these two methods say little about them. The same-feature classifiers learn from benchmark labels, which the model never sees. The decoy and placebo services are drawn at random with a fixed seed, and the decoy reuses the deployed builder’s wording; a natural false hint may act differently. Our SimpleRCA-style heuristic is a reimplementation; its pre-registered form follows Fang et al.’s per-metric description and is the strongest of the variants we tested on RE2 (Table B2).

External validity

We study one single-call agent, built by us; agents that call tools, iterate or retrieve may behave differently. The generic pipeline shows which findings survive a change of pipeline: the decoy effect does, while the RE2 shortfall and the log-text effect were not detected there (H7b and H7c on about 30 RE3 clusters, H7d on 90 RE2 clusters with interval -0.10 to $+0.02$). We designed that pipeline after seeing how per-column

statistics perform on RCAEval, so its accuracy is reported descriptively; the evidence-prompt study locates most of its advantage in the evidence it shows, which holds only at the given onset (H8c). We use two RCAEval suites from three open-source systems, and the RE2 agent-versus-pipeline result did not replicate on RE3. The models are open-weight and served by one provider, and the generic pipeline was run with two of the three families; proprietary models may behave differently, but open models let others repeat the study (Baltes et al. 2025). Every method is given the injection time; Section 5.6 and H8c show how much the results depend on it.

Contamination

The benchmark systems and RCAEval are public, and case identifiers never reach a prompt. The confirmatory anonymisation (RQ4) left names in 86 of 245 prompts by the pre-registered count and in 166 by the strict detector; `anon_full` left none. On RE2, `anon_full` did not lower accuracy for any model, which argues against a benefit from memorised names there. On RE3 it lowered the deployed model’s accuracy by 0.21 (0.17 for the other two models). The permutation (H6) shows that the correct correspondence between names and services helps: names that denote the right service helped compared with deranged names (H6c-2), and no difference was detected between deranged names and neutral labels (H6c-1). Matching names to the words of an error message and remembering these public systems would both produce that result.

Conclusion validity

Inference is clustered over (system, root service, fault) and Holm-adjusted within each pre-registered family. Appendix A lists all 53 tests. Under a Benjamini–Hochberg adjustment over all of them, every supported verdict holds except two RE3 replications, P3b for the deployed model and decoy versus placebo, whose RE2 counterparts have $p < 0.001$; the RE3 P3b verdict still holds for the other two models. RE2 can detect differences of about 7 points (10 after Holm), RE3 about 14, so a null result on RE3 is weak evidence of no effect. On byte-identical prompts `gpt-oss-120b` changed its top-1 in 20% of cases, but its AC@1 varied by only 0.006 (SD) across three full runs; every other condition was run once, and our intervals treat that run as fixed, which matters most for the small effects (RQ4, `anon_full` on RE2, E3, P3a). Three tests lie close to 0.05: RE3 decoy versus placebo ($p = 0.0495$), the RE3 SimpleRCA-style comparison ($p_H = 0.0525$) and H6c-2 ($p_H = 0.049$); the exact tests agree with all three verdicts (0.047, 0.055 and 0.045). Exploratory results are labelled as such, and the three that matter most for our conclusions were re-tested as pre-registered hypotheses (H5), as was the explanation the RE3 results suggested (H6).

8 Conclusion

We asked where the accuracy of an LLM-based root-cause analysis agent comes from, and answered the question for one detector-to-LLM agent with a pre-registered study on RCAEval, followed by a pre-registered re-test in a second, generic pipeline. With the deployed model on RE2, the model was less accurate than its agent’s own detector

count and not detectably different from a label-free ranking of the generic pipeline’s summary; elsewhere it was at most modestly ahead of such rankings. Descriptively, the same model scored 0.44 on RE2 in the agent and 0.87 in the generic pipeline, and a pre-registered 2×2 attributed most of that difference to the evidence each pipeline shows (+0.30) and a smaller part to its prompt (+0.05). The generic pipeline’s accuracy, however, held only at the given onset: two minutes late it fell to 0.27 on RE2, close to or below the agent’s model under the same shift. In the agent, on RE2, the model was less accurate than its own detector count and than a simple classifier on the features it sees. On RE3’s code-level faults it was above its detector count, though not significantly, and below the strongest metric-based methods; a pre-registered ablation showed that it used the raw log text, without which its non-significant lead disappeared, and that names denoting the right service helped. Neither the RE2 shortfall nor the RE3 log-text effect was detected in the generic pipeline, so we report both as specific to the agent. One finding held everywhere we looked: across both suites, all three models in the agent and both models in the generic pipeline, a single false sentence in the context moved the answer towards the asserted service. Separately, pre-registered tests confirmed that the strongest statistical baselines, and the generic pipeline, lose most of their accuracy when the onset is two minutes late.

We conclude that the accuracy of such a system should be attributed to its parts: compared with a label-free reading of the model’s own input and across context designs, with its interpretive text tested like code, the raw text it reads ablated separately from the numbers, and its sensitivity to onset time reported for the pipeline as well as for the baselines. Whether the name effect reflects matching names to the words of the evidence or prior knowledge of public systems remains open; a benchmark with renamed services would answer it. The replication package lets others apply the same decomposition to their own agents.

Declarations

Data and code availability

The replication package, with the protocol, every pre-registered stage and the execution logs, all code, prompts, raw model responses, per-case rankings, analysis outputs and a script that reruns every analysis without model calls, is archived on Zenodo at <https://doi.org/10.5281/zenodo.23409452> (version 1.0.1) (Hussain and Farrukh 2026a) and developed at <https://github.com/saadhussain07/llm-rca-accuracy-study>. The package is a snapshot of the pre-registration record, <https://github.com/saadhussain07/rca-benchmark-study>, whose history and tags (Table 3) show when the protocol and each later stage were fixed; a script in the package checks that its files are identical to the record’s. The system under study and the live-study data are archived as release v1.1.0 at <https://doi.org/10.5281/zenodo.23107492> (Hussain and Farrukh 2026b). The six deployed files kept in the package’s `original/` folder are identical to that release apart from line endings (the copies keep the Windows line endings of the deployment). Section 11 of the frozen protocol gives the system’s archive as 10.5281/zenodo.23107449; the release used in the study is 23107492.

Ethics approval and consent to participate

Not applicable: the study uses a public benchmark and involves no human participants.

Consent for publication

Not applicable.

Materials availability

See Data and code availability.

Funding

This research received no specific grant from any funding agency in the public, commercial or not-for-profit sectors.

Conflict of interest

The authors developed the system under study. They declare no other competing interests.

Author contributions (CRediT)

Muhammad Saad Hussain: Conceptualization, Methodology, Software, Investigation, Data curation, Formal analysis, Visualization, Writing – original draft. **Alishba Farrukh:** Supervision, Methodology, Validation, Writing – review & editing.

Use of generative AI

The authors used Anthropic’s Claude as an assistant for programming, for drafting and editing text, and for checking the analysis and the manuscript. The authors directed the study, approved every decision, and verified all code, results and text; they take full responsibility for the content.

Appendix A Ledger of hypothesis tests

Table [A1](#) lists every family of hypothesis tests in the study, the stage that fixed it, and how many of its tests are significant under the pre-registered within-family Holm adjustment and, as a sensitivity analysis, under a Benjamini–Hochberg adjustment over all 53 tests together. The per-test ledger, with each difference, raw p , Holm-adjusted p and BH-adjusted p , is in the replication package. Equivalence and non-inferiority tests (RQ4, H5b), which are judged by their confidence bounds, are not included.

Appendix B Supplementary results

Baseline reruns (Amendment 2)

Table [B1](#) gives the outcome of the Amendment 2 reruns for the methods whose first run was incomplete.

Table A1 Hypothesis tests by family. Significant: number of tests with adjusted $p < 0.05$. Under Benjamini–Hochberg over all 53 tests, every supported verdict holds except two RE3 replications, P3b for the deployed model ($p = 0.041$) and decoy versus placebo ($p = 0.0495$); their RE2 counterparts have $p < 0.001$.

Family	Fixed by	Tests	Holm (family)	BH (all 53)
RE2 RQ1/RQ2: model vs comparators	protocol	13	9	10
RE2 P3a: NOTE removal changes answers	protocol	3	0	1
RE2 P3b: decoy named first	protocol	3	3	3
RE2 decoy vs placebo	protocol	1	1	1
RE2 P3c: agreement across models	protocol	1	0	0
RE3 RQ1/RQ2: model vs comparators	RE3 replication	13	8	9
RE3 P2: classifier trained on RE2	RE3 replication	1	0	0
RE3 P3b: decoy named first	RE3 replication	3	3	2
RE3 decoy vs placebo	RE3 replication	1	1	0
H5: RE3 tests of RE2 findings	follow-up	3	2	2
H6: log text and names	follow-up	4	3	3
H7: generic pipeline	follow-up	4	1	1
H8: evidence, prompt and onset	follow-up	3	3	3
Total		53	34	35

Table B1 Outcome of the Amendment 2 reruns on RE2 for the methods whose first run was incomplete. *Not reached*: neither ranked nor failed before the two-hour limit. Every unranked case is scored as chance. The confirmatory RQ1 comparisons use the rerun rankings; the first-run results are reported as a sensitivity analysis.

Method	System	First run	Ranked	Failed	Not reached
ϵ -Diagnosis	OB, SS, TT	error, 0 ranked	90 each	0	0
MicroCause	OB	error, 0 ranked	27	0	63
MicroCause	SS	error, 0 ranked	21	0	69
MicroCause	TT	time limit, 0 ranked	0	0	90
BARO (multi-source)	TT	stopped after 69	89	1	0
N-Sigma (multi-source)	TT	stopped after 69	89	1	0

SimpleRCA-style variants

Table B2 gives the pre-registered form of the heuristic and the variants of Amendment 4 (Section 5.6).

Position in the context

The model’s top-1 is the first service of the **AFFECTED SERVICES** line in 99 of 176 multi-service cases (**gpt-oss-120b**). It is correct in 61% of cases when the root is listed first and 28% when it is not. The order of that line is the arbitrary hash order of the deployed code (Table 1). Randomly permuting it, with two seeds, left accuracy unchanged (+0.000 and +0.029, both n.s.). The permutation tests whether accuracy depends on the order, not whether the model prefers the first position. The per-case answers settle the second question. In the cases where the permutation moved a different service to the front (101 and 92), the model named the new first service in 24 and 19 of them, and kept the previously first service in 53 and 50, about as often as in the frozen run (52 and 46). The model follows the service, not the position: the

Table B2 SimpleRCA-style heuristic: pre-registered form (V0) and variants. AC@1 on all cases.

Variant	RE2				RE3
	OB	SS	TT	All	All
V0 pre-registered (one alert per metric column)	0.744	0.911	0.778	0.811	0.833
V1 without the $\sigma = 0$ rule	0.689	0.867	0.789	0.781	0.833
V2 own columns only (no -db)	0.744	0.878	0.778	0.800	0.867
V3 at most one alert per signal type	0.367	0.278	0.411	0.352	0.500
V4 V1 + V2 + V3	0.378	0.356	0.411	0.381	0.544
Fang et al., published	0.47	0.24	0.80	–	–

association arises because the services it tends to choose, such as `queue-master`, often come first in the hash order.

References

- Ahmed T, Ghosh S, Bansal C, et al (2023) Recommending root-cause and mitigation steps for cloud incidents using large language models. In: Proceedings of the 45th International Conference on Software Engineering (ICSE), pp 1737–1749, [arXiv:2301.03797](https://arxiv.org/abs/2301.03797)
- Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), pp 1–10, <https://doi.org/10.1145/1985793.1985795>
- Arp D, Quiring E, Pendlebury F, et al (2022) Dos and don’ts of machine learning in computer security. In: 31st USENIX Security Symposium (USENIX Security 22), pp 3971–3988
- Baltes S, Angermeir F, Arora C, et al (2025) Guidelines for empirical studies in software engineering involving large language models. <https://doi.org/10.48550/arXiv.2508.15503>, living version at <https://llm-guidelines.org>, [arXiv:2508.15503](https://arxiv.org/abs/2508.15503)
- Cai Y, Nie X, Yin K, et al (2026) A multi-dataset benchmark for evaluating LLM agents in microservice failure diagnosis. [arXiv:2606.29193](https://arxiv.org/abs/2606.29193)
- Cao J, Zhang W, Cheung SC (2024) Concerned with data contamination? assessing countermeasures in code language model. [arXiv:2403.16898](https://arxiv.org/abs/2403.16898)
- Chen Y, Xie H, Ma M, et al (2024) Automatic root cause analysis via large language models for cloud incidents. In: Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys), [arXiv:2305.15778](https://arxiv.org/abs/2305.15778)
- Chen Y, Shetty M, Somashekar G, et al (2025) AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. Presented at MLSys 2025, [arXiv:2501.06706](https://arxiv.org/abs/2501.06706)

- Chen Z, Kang Y, Li L, et al (2020) Towards intelligent incident management: Why we need it and how we make it. In: Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), <https://doi.org/10.1145/3368089.3417055>
- Dang Y, Lin Q, Huang P (2019) AIOps: Real-world challenges and research innovations. In: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp 4–5, <https://doi.org/10.1109/ICSE-Companion.2019.00023>
- Ernst NA, Baldassarre MT (2023) Registered reports in software engineering. Empirical Software Engineering <https://doi.org/10.1007/s10664-022-10277-5>, [arXiv:2302.03649](https://arxiv.org/abs/2302.03649)
- Fang A, Yang Y, Shang J, et al (2026a) OpenRCA 2.0: From outcome labels to causal process supervision. [arXiv:2606.27154](https://arxiv.org/abs/2606.27154)
- Fang A, Zhang S, Yang Y, et al (2026b) Rethinking the evaluation of microservice RCA with a fault propagation-aware benchmark. Proceedings of the ACM on Software Engineering 3(FSE):1605–1627. <https://doi.org/10.1145/3797100>, [arXiv:2510.04711](https://arxiv.org/abs/2510.04711)
- Fu W, Menzies T (2017) Easy over hard: A case study on deep learning. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), <https://doi.org/10.1145/3106237.3106256>, [arXiv:1703.00133](https://arxiv.org/abs/1703.00133)
- Geirhos R, Jacobsen JH, Michaelis C, et al (2020) Shortcut learning in deep neural networks. Nature Machine Intelligence 2(11):665–673. <https://doi.org/10.1038/s42256-020-00257-z>
- Gopal A, Krishnan A (2026) How far can root cause analysis go on real-world telemetry data? [arXiv:2607.13548](https://arxiv.org/abs/2607.13548)
- He P, Zhu J, Zheng Z, et al (2017) Drain: An online log parsing approach with fixed depth tree. In: Proceedings of the IEEE International Conference on Web Services (ICWS), pp 33–40, <https://doi.org/10.1109/ICWS.2017.13>
- Hellendoorn VJ, Devanbu P (2017) Are deep neural networks the best choice for modeling source code? In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), <https://doi.org/10.1145/3106237.3106290>
- Holm S (1979) A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics 6(2):65–70
- Hu L, Liu T, Fu Y (2026) Pooled leaderboards hide system-specific winners: A reporting-protocol audit of offline root-cause analysis benchmarks. [arXiv:2606.29159](https://arxiv.org/abs/2606.29159)

- Huang H, Lyu J, Jiang Z, et al (2026) From general agents to RCA experts: A self-evolving harness for root cause analysis. [arXiv:2608.25661](#)
- Hussain MS, Farrukh A (2026a) Replication package for: Where does the accuracy of LLM-based root-cause analysis come from? An empirical study on microservice benchmarks. Zenodo, version 1.0.1, <https://doi.org/10.5281/zenodo.23409452>
- Hussain MS, Farrukh A (2026b) saadhussain07/fgcs-aiops-llm-agent: v1.1.0. Zenodo, software release v1.1.0, <https://doi.org/10.5281/zenodo.23107492>
- Ikram A, Chakraborty S, Mitra S, et al (2022) Root cause analysis of failures in microservices through causal discovery. In: Advances in Neural Information Processing Systems 35 (NeurIPS)
- Jha S, Arora R, Watanabe Y, et al (2025) ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In: International Conference on Machine Learning (ICML), [arXiv:2502.05352](#)
- Kim T, Park W, Yun H, et al (2026) Why do AI agents systematically fail at cloud root cause analysis? [arXiv:2602.09937](#)
- Landauer M, Skopik F, Wurzenberger M (2024) A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques. Proceedings of the ACM on Software Engineering 1(FSE):1354–1375. <https://doi.org/10.1145/3660768>
- Le VH, Zhang H (2022) Log-based anomaly detection with deep learning: How far are we? In: Proceedings of the 44th International Conference on Software Engineering (ICSE), pp 1356–1367, <https://doi.org/10.1145/3510003.3510155>, [arXiv:2202.04301](#)
- Lee C, Yang T, Chen Z, et al (2023) Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In: Proceedings of the 45th International Conference on Software Engineering (ICSE), pp 1750–1762, [arXiv:2302.05092](#)
- Li M, Li Z, Yin K, et al (2022a) Causal inference-based root cause analysis for online service systems with intervention recognition. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), <https://doi.org/10.1145/3534678.3539041>, [arXiv:2206.05871](#)
- Li Y, Zhang T, Luo X, et al (2023) Do pre-trained language models indeed understand software engineering tasks? IEEE Transactions on Software Engineering 49(10):4639–4655. <https://doi.org/10.1109/TSE.2023.3308952>, [arXiv:2211.10623](#)
- Li Z, Zhao N, Li M, et al (2022b) Actionable and interpretable fault localization for recurring failures in online service systems. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), <https://doi.org/10.1145/3540250.3549092>,

[arXiv:2207.09021](https://arxiv.org/abs/2207.09021)

- Liu FT, Ting KM, Zhou ZH (2008) Isolation forest. In: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), pp 413–422, <https://doi.org/10.1109/ICDM.2008.17>
- Liu Z, Xia X, Hassan AE, et al (2018) Neural-machine-translation-based commit message generation: How far are we? In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), pp 373–384, <https://doi.org/10.1145/3238147.3238190>
- Luo Y, Gao R, Zhou Z, et al (2026) Rethinking tool design for agentic RCA: A controlled empirical study. [arXiv:2610.05009](https://arxiv.org/abs/2610.05009)
- Muhammad HM, Pham L, Barrière L, et al (2026) Where root cause analysis fails: A retrieval-reranking decomposition. Accepted at NeurIPS 2026, Evaluations & Datasets Track, [arXiv:2609.36686](https://arxiv.org/abs/2609.36686)
- Naakka A, Wang Y, Mäntylä MV (2026) LATs-RCA: Language agent tree search for root cause analysis in microservices. [arXiv:2605.03505](https://arxiv.org/abs/2605.03505)
- Notaro P, Cardoso J, Gerndt M (2021) A survey of AIOps methods for failure management. *ACM Transactions on Intelligent Systems and Technology* 12(6):81:1–81:45. <https://doi.org/10.1145/3483424>
- Ouyang S, Zhang JM, Harman M, et al (2025) An empirical study of the non-determinism of ChatGPT in code generation. *ACM Transactions on Software Engineering and Methodology* 34(2):1–28. <https://doi.org/10.1145/3697010>, [arXiv:2308.02828](https://arxiv.org/abs/2308.02828)
- Pedregosa F, Varoquaux G, Gramfort A, et al (2011) Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830
- Pham L (2026) Graph-free root cause analysis. [arXiv:2601.21359](https://arxiv.org/abs/2601.21359)
- Pham L, Ha H, Zhang H (2024a) BARO: Robust root cause analysis for microservices via multivariate bayesian online change point detection. *Proceedings of the ACM on Software Engineering* 1(FSE):2214–2237. <https://doi.org/10.1145/3660805>, [arXiv:2405.09330](https://arxiv.org/abs/2405.09330)
- Pham L, Ha H, Zhang H (2024b) Root cause analysis for microservices based on causal inference: How far are we? In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 706–715, <https://doi.org/10.1145/3691620.3695065>, [arXiv:2408.13729](https://arxiv.org/abs/2408.13729)
- Pham L, Zhang H, Ha H, et al (2025) RCAEval: A benchmark for root cause analysis of microservice systems with telemetry data. In: Companion Proceedings of the

- ACM Web Conference 2025 (WWW '25 Companion), pp 777–780, <https://doi.org/10.1145/3701716.3715290>, [arXiv:2412.17015](https://arxiv.org/abs/2412.17015)
- Pham L, Ha H, Zhang X, et al (2026) TORAI: Multi-source root cause analysis for blind spots in microservice service call graph. *Proceedings of the ACM on Software Engineering* 3(FSE):2928–2951. <https://doi.org/10.1145/3808137>, [arXiv:2604.13522](https://arxiv.org/abs/2604.13522)
- Ralph P, bin Ali N, Baltes S, et al (2020) Empirical standards for software engineering research. [arXiv:2010.03525](https://arxiv.org/abs/2010.03525)
- Remil Y, Bendimerad A, Mathonat R, et al (2024) AIOps solutions for incident management: Technical guidelines and a comprehensive literature review. [arXiv:2404.01363](https://arxiv.org/abs/2404.01363)
- Riddell E, Riddell J, Sun G, et al (2026) Stalled, biased, and confused: Uncovering reasoning failures in LLMs for cloud-based root cause analysis. In: *Proceedings of the IEEE/ACM 3rd International Conference on AI Foundation Models and Software Engineering (FORGE)*, <https://doi.org/10.1145/3793655.3793732>, [arXiv:2601.22208](https://arxiv.org/abs/2601.22208)
- Roy D, Zhang X, Bhawe R, et al (2024) Exploring LLM-based agents for root cause analysis. In: *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion), Industry Track*, <https://doi.org/10.1145/3663529.3663841>, [arXiv:2403.04123](https://arxiv.org/abs/2403.04123)
- Sallou J, Durieux T, Panichella A (2024) Breaking the silence: the threats of using LLMs in software engineering. In: *Proceedings of the 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pp 102–106, <https://doi.org/10.1145/3639476.3639764>, [arXiv:2312.08055](https://arxiv.org/abs/2312.08055)
- Sclar M, Choi Y, Tsvetkov Y, et al (2024) Quantifying language models’ sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In: *International Conference on Learning Representations (ICLR)*, [arXiv:2310.11324](https://arxiv.org/abs/2310.11324)
- Shi F, Chen X, Misra K, et al (2023) Large language models can be easily distracted by irrelevant context. In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pp 31210–31227, [arXiv:2302.00093](https://arxiv.org/abs/2302.00093)
- Soldani J, Brogi A (2022) Anomaly detection and failure root cause analysis in (micro)service-based cloud applications: A survey. *ACM Computing Surveys* [arXiv:2105.12378](https://arxiv.org/abs/2105.12378)
- Tian Y, Liu Y, Chong Z, et al (2025) GALA: Can graph-augmented large language model agentic workflows elevate root cause analysis? [arXiv:2508.12472](https://arxiv.org/abs/2508.12472)

- Turpin M, Michael J, Perez E, et al (2023) Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In: Advances in Neural Information Processing Systems 36 (NeurIPS), [arXiv:2305.04388](#)
- Wagner S, Muñoz Barón M, Falessi D, et al (2025) Towards evaluation guidelines for empirical studies involving LLMs. In: 2nd International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE @ ICSE 2025), [arXiv:2411.07668](#)
- Wang Q, Zhang X, Li M, et al (2025) TAMO: Fine-grained root cause analysis via tool-assisted LLM agent with multi-modality observation data. [arXiv:2504.20462](#)
- Wang Y, Yu G, Huang H, et al (2026) Cloud-OpsBench: A reproducible benchmark for agentic root cause analysis in cloud systems. [arXiv:2603.00468](#)
- Wang Z, Liu Z, Zhang Y, et al (2024) 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), <https://doi.org/10.1145/3627673.3680016>, [arXiv:2310.16340](#)
- Wu R, Keogh EJ (2021) Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. IEEE Transactions on Knowledge and Data Engineering <https://doi.org/10.1109/TKDE.2021.3112126>, [arXiv:2009.13807](#)
- Xu J, Zhang Q, Zhong Z, et al (2025) OpenRCA: Can large language models locate the root cause of software failures? In: International Conference on Learning Representations (ICLR), URL <https://openreview.net/forum?id=M4qNIzQYpd>
- Yu B, Yao J, Fu Q, et al (2024) Deep learning or classical machine learning? an empirical study on log-based anomaly detection. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE), <https://doi.org/10.1145/3597503.3623308>
- Yu G, Chen P, Li Y, et al (2023) Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), <https://doi.org/10.1145/3611643.3616249>
- Zhang L, Jia T, Jia M, et al (2025a) A survey of AIOps in the era of large language models. ACM Computing Surveys <https://doi.org/10.1145/3746635>, [arXiv:2507.12472](#)
- Zhang L, Jia T, Wang K, et al (2025b) Adaptive root cause localization for microservice systems with multi-agent recursion-of-thought. [arXiv:2508.20370](#)

- Zhang S, Xia S, Fan W, et al (2024a) Failure diagnosis in microservice systems: A comprehensive survey and analysis. [arXiv:2407.01710](#)
- Zhang W, Guo H, Yang J, et al (2024b) mABC: Multi-agent blockchain-inspired collaboration for root cause analysis in micro-services architecture. In: Findings of the Association for Computational Linguistics: EMNLP 2024, pp 4017–4033, <https://doi.org/10.18653/v1/2024.findings-emnlp.232>
- Zhang X, Ghosh S, Bansal C, et al (2024c) Automated root causing of cloud incidents using in-context learning with GPT-4. In: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion), [arXiv:2401.13810](#)
- Zhou X, Peng X, Xie T, et al (2021) Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* 47(2):243–260. <https://doi.org/10.1109/TSE.2018.2887384>
- Zhou X, Weyssow M, Widayarsi R, et al (2025) LessLeak-Bench: A first investigation of data leakage in LLMs across 83 software engineering benchmarks. [arXiv:2502.06215](#)