CONTEXTCRAFT: ROOT CAUSE ANALYSIS WITH TOOL-AUGMENTED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Root Cause Analysis (RCA) for modern distributed systems requires synthesizing gigabytes of heterogeneous observability signals during active incidents where engineers must restore service quickly despite noisy, delayed, or missing telemetry data. Recent LLM-guided RCA systems demonstrate that agents can iteratively query telemetry and propose diagnoses, yet they typically commit to a single conclusion via unstructured scratchpads with no mechanism to compare or audit competing hypotheses. We propose ContextCraft—a well designed agentic RCA system to operationalize Analysis of Competing Hypotheses (ACH) for AI agents augmented with a curated set of tools that are efficient in root cause localization. Across our evaluations, ContextCraft demonstrates that structured hypothesis–evidence reasoning can improve the accuracy–efficiency trade-off over free-form agentic RCA, achieving up to a 76.3% relative accuracy gain and 2.14× faster execution in our primary comparison, while producing fully auditable, evidence-grounded reports.

1 INTRODUCTION

Root Cause Analysis (RCA) is a foundational capability for operating modern distributed systems. During incidents, engineers must rapidly determine why the system is malfunctioning and where the fault originates. Evidence is distributed across gigabytes of heterogeneous observability data—metrics, logs, distributed traces, and service topology (Xu et al., 2025; Luo et al., 2025; Wittkopp et al., 2024; Wang & Qi, 2024)—and many explanations can fit the early symptoms. Automating RCA could therefore reduce downtime, diagnosis costs, and the cognitive load of on-call engineers.

Recent works have shown that large language models (LLMs) can act as *agents* that interleave reasoning with tool use (Yao et al., 2023) to iteratively query telemetry and propose diagnoses (Xu et al., 2025; Luo et al., 2025; Zheng et al., 2025; Ji & Luo, 2025). OpenRCA (Xu et al., 2025) provides a public benchmark in which an agent inspects metrics, logs, and traces and returns a root-cause report. Its ground-truth labels, standardized scoring, and open-source baseline permit matched comparison. Existing agentic RCA systems (Xu et al., 2025; Zheng et al., 2025; Luo et al., 2025; Ji & Luo, 2025) typically accumulate evidence in free-form text and commit to one hypothesis. This makes it difficult to compare alternatives, track support and refutation, or know when to stop gathering evidence. The problem is acute when faults propagate across dependencies and make downstream symptoms resemble root causes (Wittkopp et al., 2024; Wang & Qi, 2024). As a result, several plausible causes may exhibit partially overlapping telemetry signatures—for example, a network fault and a downstream-service slowdown can both surface as latency spikes. Without an explicit mechanism for comparing alternatives and tracking supporting or refuting evidence, an agentic diagnosis is prone to confirmation bias (committing to the first plausible hypothesis), is harder to audit in post-mortems, and tends toward excessive evidence gathering when convergence is unclear. We designed ContextCraft to address these three failure modes: ACH-guided hypothesis management enforces diverse, comparative reasoning (Problem 1); the evidence matrix provides structured, per-hypothesis verdict tracking (Problem 2); and the budget-aware sufficiency criterion stops the agent when additional tool calls are unlikely to change the ranking (Problem 3). Finally, we propose **ContextCraft**,[1] an agentic RCA framework that operationalizes ACH for tool-augmented LLM agents to address

[1]The name reflects the design goal of the system: it actively *crafts* the evidence "context" that is exposed to the LLM at each step, rather than dumping raw tool outputs into the prompt.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the three failure modes described above: implicit reasoning, lack of structured evidence tracking, and unbounded evidence gathering. ContextCraft begins by generating a diverse hypothesis set, then collects evidence using a curated toolkit of telemetry analysis tools grounded in statistical methods and prior RCA research. The gathered evidence directly drives hypothesis refinement and targeted follow-up evidence collection. Evidence is organized in a structured matrix that maps every retrieved evidence item to a verdict for each hypothesis, providing both the agent and human reviewers with full traceability of every diagnostic decision.

Finally, ContextCraft employs an evidence sufficiency gate to stop the investigation when additional tool calls are unlikely to change the ranking of hypotheses, which reduces latency, API cost, and prompt-context dilution.

2 Background

2.1 Task Definition: Observability-Based Root Cause Analysis

We focus on the RCA setting most commonly studied in recent agentic benchmarks (Xu et al., 2025; Luo et al., 2025): an incident is defined by a time window and a textual incident description (or alert), and the agent has read-only access to the system's observability data. Provided telemetry modalities include (i) time-series metrics (e.g., error rate, latency), (ii) unstructured or semi-structured logs, and (iii) distributed traces capturing request paths. In our research, we deliberately restrict attention to this incident-localization formulation—rather than, for example, change-induced regression analysis (Yu et al., 2024)—because it is the setting for which public benchmarks, ground truth, and tool-call budgets are currently well defined; the ACH-based machinery we introduce is not specific to it and could be applied wherever the analyst must adjudicate between competing explanations supported by evidence collected through the OpenTelemetry observability framework (OpenTelemetry Authors, 2026). For the purposes of this paper, we define the task objective as identifying the most plausible explanatory factor, i.e., the root cause, and producing an actionable RCA report.

2.2 Analysis of Competing Hypotheses (ACH)

ACH, as described by Heuer (1999), is a structured methodology for evaluating multiple hypotheses simultaneously in order to reach evidence-based conclusions while minimizing confirmation bias. Rather than seeking only confirming evidence for a favored explanation, ACH evaluates whether available evidence supports or disconfirms each hypothesis. ACH originated in intelligence analysis and has been successfully applied to reduce confirmation bias in national-security assessments (Heuer, 1999; Dhami et al., 2019), and collaborative sensemaking under uncertainty (Convertino et al., 2008). We adopt ACH for RCA for the same reason: production incidents typically admit several causally plausible explanations whose telemetry signatures partially overlap, and we want the agent to keep all of them in view until evidence has explicitly separated them. (Wittkopp et al., 2024; Wang & Qi, 2024) In ContextCraft, ACH motivates (i) maintaining a set of competing hypotheses, (ii) collecting evidence for candidate hypotheses via tool calls, and (iii) evaluating each item against *every* current hypothesis. Evidence collection is candidate-driven, but the shared matrix allows the same result to support one hypothesis and refute another.

3 Related Work and Literature

3.1 LLM-Based Root Cause Analysis

Recent literature has explored the use of LLMs as agents for automated RCA, where models retrieve telemetry data and generate theories on the root of the failure. Early LLM-based RCA systems such as RCACopilot (Chen et al., 2024) and RCAgent (Wang et al., 2024) demonstrated that production cloud incidents can be diagnosed by aggregating diagnostic signals and reasoning over them in natural language; both, however, depend on manually engineered incident handlers or domain workflows rather than an open, reproducible benchmark. OpenRCA (Xu et al., 2025) introduces a benchmark and evaluation framework in which LLM based agents analyze varied and vast observability data, including metrics, logs, and traces, in order to identify the root cause of software failures.

This type of solution explores the feasibility of LLM-driven RCA, but also exposes key limitations: agents typically accumulate evidence in free form reasoning steps and prematurely converge to a single hypothesis without systematically evaluating alternatives.

Several following works extend this paradigm with richer agentic workflows (Luo et al., 2025; Ji & Luo, 2025), often incorporating multi-step reasoning, dynamic tool generation, or planning-based execution; the more heavily structured and multi-agent variants are reviewed in Appendix A.3. These systems often rely on preprocessing telemetry into structured intermediate representations before reasoning, highlighting the importance of transforming raw observability data into structured context.

However, these approaches generally lack explicit mechanisms for maintaining competing hypotheses, tracking evidence relationships, and encouraging disconfirmation. As a result, they remain prone to incomplete exploration, and limited auditability—failure modes that have recently been observed to systematically degrade agentic cloud RCA (Kim et al., 2026). In contrast, ContextCraft introduces an explicit structure over the reasoning process by operationalizing ACH (Section 2.2), forcing the agent to evaluate each piece of evidence against all candidate explanations.

3.2 Telemetry Analysis and Anomaly Detection

A core component of any RCA system is the ability to extract meaningful signals from telemetry data that is otherwise large and not always homogeneous or straightforward to interpret (Zhang et al., 2025a). Prior research in time-series anomaly detection provides a wide variety of statistical and ML based methods to identify deviations in system behavior. Comprehensive surveys (Schmidl et al., 2022) evaluate these methods across diverse settings and show that classical techniques, such as wavelet based decomposition, density-based outlier detection, and causality analysis remain highly competitive, particularly in noisy and heterogeneous environments. A large body of pre-LLM RCA work builds directly on these signals: graph- and correlation-based localization for microservices (Wu et al., 2020), log-anomaly-driven metric localization (Wang et al., 2020), deep-learning log anomaly detection (Du et al., 2017; van Ede et al., 2022), symbolic execution-trace analysis (Yagemann et al., 2021), and reusable metric-based RCA libraries (Liu et al., 2023).

In this work, we adopt a fixed toolkit spanning wavelet and local-outlier detection, STL-based cross-component comparison, KPI-family change analysis, log bursts, trace latency, service topology, metric-tail retrieval, and topology-aware Granger causality (Thill et al., 2017; Han et al., 2025). These methods are selected based on their demonstrated effectiveness in scenarios characterized by multivariate dependencies, temporal locality, and noisy observations, conditions that are common in production observability data. Importantly, these tools produce structured, LLM interpretable outputs that can be directly incorporated into the evidence matrix, aligning with our goal of maintaining auditable and structured reasoning— an important property for industrial adoption, since SRE and on-call engineers must be able to validate, trust, and act on an automated diagnosis during a live incident.

Unlike prior agentic approaches that dynamically generate analysis code at runtime (Xu et al., 2025; Luo et al., 2025), ContextCraft relies on a fixed, validated toolkit with well-defined interfaces, improving reliability and consistency across runs. This design choice reflects our broader emphasis on context engineering and tool reliability over dynamic generation, reducing both variance in outcomes and computational overhead. Appendix A.1 lists every tool used by the pipeline and its role.

4 Method: ContextCraft

4.1 Core definitions

As discussed in Section 2.1, a task is an incident instance consisting of a free-text description, a time window, and read-only access to the available telemetry. The agent must return a root-cause hypothesis identifying the faulty component, occurrence time, and/or failure reason (depending on which dimensions are missing from the query).

We define the following core objects:

- An *incident time window* $w = [t_{\text{start}}, t_{\text{end}}]$ bounding the period under investigation.

- A *dimension set* $D \subseteq \{\text{TIME}, \text{COMPONENT}, \text{REASON}\}$ specifying which root-cause fields are missing from the incident query and must be inferred by the agent.
- A set of *hypotheses* $H^{(t)} = \{h_1, \ldots, h_m\}$ maintained at iteration t of the evidence loop, where each $h_j \in H^{(t)}$ is a candidate root cause that instantiates exactly the fields in D (e.g., if $D = \{\text{COMPONENT}, \text{REASON}\}$, every hypothesis names a suspected component and a failure reason). The set evolves across iterations by appending planner-proposed and deterministically extracted candidates (Section 4.2).
- A set of *evidences* $E^{(t)} = \{e_1, \ldots, e_n\}$ accumulated through iteration t from tool calls, where each $e_i \in E^{(t)}$ records the tool name, its parameters, and a normalized summary of its output.
- A curated set of *evidence tools* $\mathcal{T} = \{T_k\}_{k=1}^{K}$, where $T_k : w \times \mathcal{P} \to E$ maps a window and tool parameters to structured evidence. The subset $\mathcal{T}_{\text{init}}$ runs before hypothesis generation; Appendix A.1 lists the implementation.
- A *verdict* alphabet $V = \{\text{SUPPORTS}, \text{WEAK_SUPPORT}, \text{NOT_RELEVANT}, \text{REFUTES}\}$ used to label how a single evidence item relates to a hypothesis.
- A *round cap* $N_{\max} = 3$ that bounds evidence collection independently of the LLM-based sufficiency check.

Evidence-verdict prompt and matrix. After each evidence round, one evaluator prompt receives the new tool outputs, the current hypotheses, and the initial telemetry summary. It labels each relevant hypothesis–result pair as SUPPORTS, WEAK_SUPPORT, REFUTES, or NOT_RELEVANT, using dimension-specific rules for time proximity, component anomaly strength, and KPI-family agreement (Listing 4). ContextCraft stores these labels in an accumulated evidence matrix EM, deduplicated by hypothesis and tool-call identifier. Hypotheses added later are evaluated from that round onward rather than retroactively against earlier outputs.

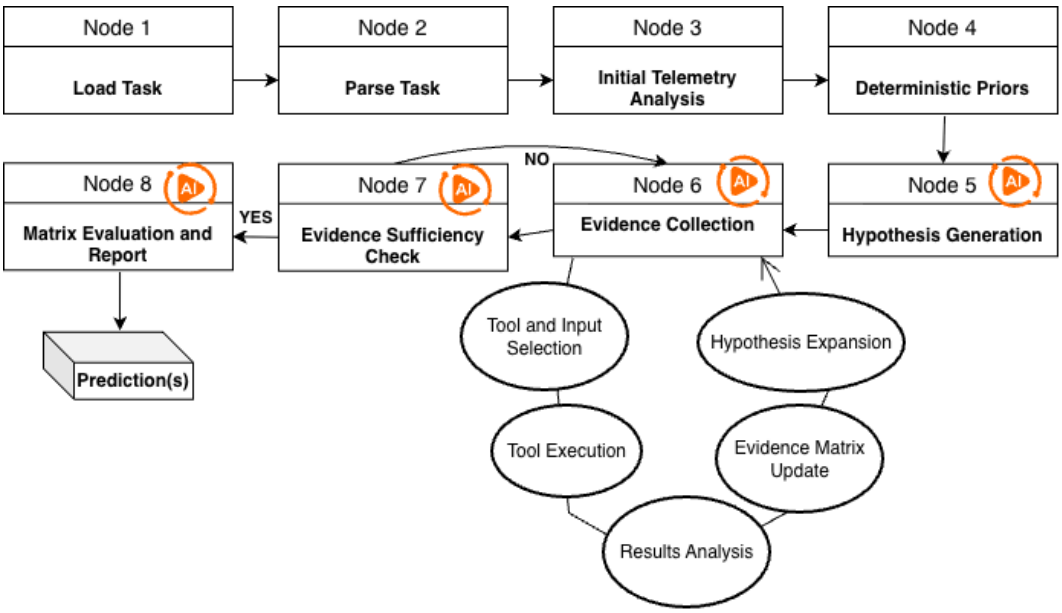


Figure 1: ContextCraft's state machine

4.2 AGENTIC WORKFLOW OVERVIEW

ContextCraft is implemented as a LangGraph state machine (see Figure 1); the nodes mirror the steps enumerated below.

Step 1–2: Load and Parse Task. Given a free-text human query q (the incident description provided by an engineer or alert system), the parser returns

$$\pi(q) = (w, D),$$

In the implementation, D is encoded as an integer `task_type` (1–7), whose mapping to missing dimensions is given in Table 4 of Appendix A.5. The code is produced by the parser LLM and then corrected by a deterministic validator. The window w is treated as context, and TIME is marked missing only when the query explicitly asks to identify the exact occurrence minute. Thus an unknown exact time gives TIME $\in D$ while w remains the nonempty search interval; it does not produce $\pi(q) = (\emptyset, D)$. Parsing D early prevents wasted tool calls and avoids biasing $H^{(t)}$ toward already-known fields.

Step 3–5: Initial Telemetry Analysis, Deterministic Priors, and Hypothesis Generation. Before consulting the LLM we run every tool in the initial subset $\mathcal{T}_{\text{init}}$ over the incident window w (Appendix A.1), collecting the initial evidence $E^{(0)} = \bigcup_{T_k \in \mathcal{T}_{\text{init}}} T_k(w, \emptyset)$. The hypothesis-generation prompt combines their summaries with the parsed task $\pi(q) = (w, D)$ to produce the initial candidate set $H^{(0)}$ over the missing dimensions D (Listing 2). Running this analysis *before* hypothesis generation is what makes the initial $H^{(0)}$ data-grounded rather than a purely prior-driven brainstorm.

Step 6a: Tool Planning. At iteration t the LLM-based planner selects a batch of tools to run

$$\textsc{Plan}\Big(H^{(t)}, EM^{(t)}\Big) = \big\{T_r\big\}_{r=1}^{R^{(t)}} \subseteq \mathcal{T}.$$

The planner is shown the current hypotheses and the existing evidence matrix $EM^{(t)}$ and is instructed, via qualitative rubrics (tool-to-hypothesis matching and convergence rules; see Listing 3), to prefer tool calls that discriminate between competing candidates rather than piling on support for an already-favored one. The batch size $R^{(t)}$ is not fixed or bounded by the system—it is chosen by the planner.

Step 6b: Tool Execution. Each selected tool $T_t r$ receives the incident window w and hypothesis-specific parameters $\mathcal{P}_r$ (e.g., a candidate component name) and returns new evidence items. Bounding the number of *rounds* (via $N_{\max}$) plus planning conditioned on $EM^{(t)}$ is what keeps successive calls targeted rather than exploratory.

Step 6c: Hypothesis Expansion. The planning call returns new candidates alongside the tool-call plan. After de-duplication, a deterministic fallback adds up to three candidates for any required D for expansion when the planner returned none; denote the resulting batch by $H^{(t)}_{\text{plan}}$. This batch is appended before tool execution. After execution, a second deterministic extractor $X(\Delta E^{(t+1)})$ may add candidates surfaced by strong raw signals. Before verdict assignment, all candidates are therefore merged as

$$H^{(t+1)} = H^{(t)} \cup H^{(t)}_{\text{plan}} \cup X\Big(\Delta E^{(t+1)}\Big).$$

The planning prompt requests at most three new hypotheses per round (see Listing 3); the post-execution extractor may add at most one component and one reason candidate per round. All additions are gated by the missing dimensions D. In code, X reads the structured tool outputs and adds the strongest unseen component or reason only when fixed thresholds are met (at least 30% KPI-family anomaly share or a $2\times$ trace/log burst). The update does not delete existing hypotheses: the set grows monotonically, and discrimination among competing candidates is deferred to the aggregate scoring in Step 8.

Step 6d: Batch Verdict Evaluation. The evaluator prompt scores every current hypothesis–tool-result pair (including with NOT_RELEVANT), which is the ACH mechanism that encourages comparative, disconfirmation-aware reasoning. The resulting rows are merged into $EM^{(t)}$ to form $EM^{(t+1)}$. For scoring, let $I(h)$ index the entries for hypothesis h:

$$\text{count}_v(h) = \sum_{i \in I(h)} \mathbf{1}\big[EM_{i,h} = v\big], \qquad v \in V.$$

Step 7: Evidence Sufficiency Check. Termination is governed by two independent conditions. First, a hard round cap: if the round index t reaches $N_{\max}$ (Section 4.1), the loop exits regardless of any other signal. Second, an LLM-based sufficiency check

$$\mathrm{Stop}\Big(H^{(t+1)}, EM^{(t+1)}\Big) \in \{\textsc{yes}, \textsc{no}\},$$

realized by a dedicated evaluator prompt (see Appendix A.6) that is shown the evidence matrix and hypotheses and asked whether, for every missing dimension $d \in D$, a single candidate "clearly stands out" (more supports than its alternatives and clear separation from competitors). This is a qualitative LLM judgment returning a yes/no verdict. If the verdict is NO and the round cap has not been reached, the agent repeats from Step 6; otherwise it proceeds to the report. Because each additional round adds latency, monetary cost, and prompt-context bloat that crowds out relevant evidence and tends to degrade LLM reasoning quality (Kim et al., 2026), this early-stopping check is the main lever by which ContextCraft reduces tool-call count relative to the OpenRCA baselines introduced by Xu et al. (2025); Luo et al. (2025) (see Section 5).

Step 8: Matrix Evaluation and Report. From EM we compute, for each missing dimension $d \in D$, a *scorecard* that aggregates the verdict counts $\mathrm{count}_v(h)$ for each candidate hypothesis h. Let S_h, W_h, and R_h denote its numbers of SUPPORTS, WEAK_SUPPORT, and REFUTES verdicts, respectively. The scorecard uses a separate score for each dimension:

$$\phi_{\textsc{time}}(h; EM) = 3S_h + W_h - 3R_h.$$

This direct evidence balance gives weak support one-third the weight of strong support and lets one refutation cancel one strong support. To compute $\phi_{\textsc{component}}$, we introduce a verdict-weight function $s : V \to \mathbb{R}$, defined by $s(\textsc{supports}) = 3$, $s(\textsc{weak_support}) = 1$, $s(\textsc{not_relevant}) = 0$, and $s(\textsc{refutes}) = -3$. Let $\lambda_i = 0.3$ only for `get_top_changing_kpis_for_component`, which examines one component in isolation, and $\lambda_i = 1$ for comparative tools such as cross-component ranking, trace latency, and log bursts. Let p_h be the component's initial multi-source ranking score, computed before the evidence loop from DWT, SubLOF, STL comparison, log, time-cluster, and topology signals. Then

$$\begin{aligned}\phi_{\textsc{component}}(h; EM) = &\sum_{i \in I(h)} \mathrm{round}(\lambda_i s(v_i)) \\ &+ \mathrm{round}\Big(\min\Big\{4\log\Big(1 + \frac{p_h}{100}\Big), 12\Big\}\Big).\end{aligned}$$

The 0.3 factor specifically down-weights isolated KPI evidence when ranking components; the separate logarithmic prior bonus is capped at 12 points so that initial cross-source agreement helps break ties without dominating later evidence. For $\phi_{\textsc{reason}}$, we use the Bayesian-smoothed support rate

$$\phi_{\textsc{reason}}(h; EM) = \mathrm{round}\left(100\,\frac{3S_h + W_h}{S_h + W_h + R_h + 5}\right).$$

The five pseudo-observations prevent a reason tested on only a few components from outranking one supported across many components. These scoring rules are ContextCraft design choices rather than part of ACH itself.

The final root-cause values are then chosen by an LLM (see Listing 6) that treats the scorecard as the authoritative signal and uses consensus summaries and deterministic priors only as tie-breakers. Here, *consensus summaries* are the agreement statistics produced by the initial-analysis tools (how many independent tools point to the same component, minute, or reason family), and *deterministic priors* are the ranked anchors emitted by those tools; both are precomputed in Steps 3–5 and reused here only to break ties. The decision LLM is restricted to the set of admissible candidates, i.e., the values present in the current hypothesis set $H^{(t)}$ (with reasons – the REASON element of D – restricted to the controlled vocabulary).

A deterministic post-processor then validates each requested field. A predicted time is accepted if it lies within the incident window w; otherwise invalid values are replaced by the highest-ranked admissible candidate for that dimension. Among its repair rules, a support-based swap may overrule a weak choice (net-refuted or zero-support) when an admissible alternative has strictly more supports.

We note that the pipeline does *not* impose a hard $\mathrm{count}_{\text{SUPPORTS}}(h) \geq 1$ admissibility gate: if no candidate has positive support in a dimension, the highest-ranked admissible candidate on the scorecard is selected as a fallback. The report cites the specific rows of EM that support the chosen candidate and the rows that refute close alternatives, making each decision auditable: any reviewer can trace a claim back to a tool call and verdict.

5 EVALUATION

5.1 EXPERIMENTAL SETUP

We evaluate ContextCraft on the *Bank* system of the OpenRCA benchmark. Each query asks for one or more fields of the root-cause tuple: the faulty component, the occurrence time, and the root-cause reason. We compare ContextCraft with the complete OpenRCA agent using the same model for both systems. We run this comparison with Gemini 2.5 Pro, Gemini 3.5 Flash, and Qwen3. We also repeat the matched comparison with Qwen3 backing both agents to test whether the gains are model-family specific.

For the no-ACH ablation,[2] `USE_ACH=false` routes the same initial telemetry summaries and deterministic priors directly to the final-decision prompt. It bypasses hypothesis generation, evidence collection, the evidence matrix, and the sufficiency check, isolating the contribution of the ACH-guided loop.

Throughout this paper, accuracy refers to the mean partial-credit score: for each query, we compute the fraction of required root-cause fields predicted correctly and then average these scores across all 136 queries; strict exact-match accuracy is identified separately. Accordingly, all cross-system accuracy comparisons use the same mean partial-credit metric; the reported exact-match rates are separate ContextCraft diagnostics and are not compared with OpenRCA's mean scores.

5.2 RESULTS

ContextCraft's measured Gemini 2.5 Pro run achieves a mean partial-credit score of 0.2506 (95% bootstrap CI: 19.31%–31.11%). Its strict exact-match rate is 13.3%, and 41.5% of queries recover at least one correct field. Table 1 summarizes the matched comparisons.

Table 1: Results for the matched model comparisons. Runtime, calls, and tokens are per-task means; tokens include input and output.

	Gemini 2.5 Pro			Gemini 3.5 Flash		Qwen3	
Metric	OpenRCA	CC no-ACH	CC	OpenRCA	CC	OpenRCA	CC
Mean score	14.21%	16.23%	25.06%	38.73%	24.57%	18.63%	20.28%
Wall time/task (s)	581.13	45.00	271.61	607.01	215.59	332.77	429.78
LLM calls/task	49.83	2.00	5.00	48.52	7.82	51.84	7.85
Tokens/task	783,177	28,012	52,539	1,757,603	57,955.51	898,304	99,327

5.3 EFFICIENCY: RUNTIME, CALLS, AND TOKENS

Across all three matched models, ContextCraft uses 83.9–90.0% fewer LLM calls and 88.9–96.7% fewer tokens than OpenRCA. It is also 2.14 times faster with Gemini 2.5 Pro and 2.82 times faster with Gemini 3.5 Flash, although Qwen3 is slower. Thus, the call and token savings are consistent across models, while wall-clock gains depend on the serving stack. In the primary comparison, these savings accompany a higher mean score and a reduction in zero-score queries from 78.7% to 58.5%.

Flash exhibits a different quality–efficiency trade-off: OpenRCA scores 38.73% versus ContextCraft's 24.57%, despite ContextCraft using 96.7% fewer tokens. Our architecture deliberately restricts model autonomy through fixed candidates, tools, schemas, and evidence checks, an inductive bias designed for bounded RCA. The remaining accuracy gap under Flash suggests a model–workflow interaction rather than a universal gain from the structured workflow. ContextCraft av-

[2]The no-ACH ablation was run with Gemini 2.5 Pro.

erages 215.59 seconds (3.59 minutes) per completed Flash query, with 25th, 75th, and 90th percentile wall times of 177.58, 241.31, and 281.64 seconds, respectively. Its mean 54,396.74 input and 3,558.77 output tokens sum to 57,955.51 tokens per completed query. This is only 10.3% more than the 52,539 tokens used with Pro, despite the higher 7.82 versus 5.00 calls per task. The large resource reduction relative to OpenRCA remains useful when latency or inference budget is the binding constraint.

5.4 ACH ABLATION

ContextCraft no-ACH scores 16.23% (95% bootstrap CI: 11.51%–21.31%), whereas full ContextCraft scores 25.06% (95% bootstrap CI: 19.31%–31.11%). This is an observed ACH gain of 8.83 percentage points, or 54.4% relative. The exact-match rate rises from 6.6% to 13.3%, while the zero-score rate falls from 72.1% to 58.5%. Since both configurations receive the same initial telemetry summaries and deterministic priors, the observed difference is associated with the ACH-guided hypothesis, evidence, and sufficiency loop rather than the initial analysis alone.

This gain comes at a cost. Full ContextCraft averages 271.61 seconds, 5.00 LLM calls, and 52,539 tokens per task, compared with 45.00 seconds, 2.00 calls, and 28,012 tokens for no-ACH. The confidence intervals also overlap, so we treat the result as evidence of an accuracy gain in this evaluation rather than a conclusive model-independent effect.

5.5 TOOL USAGE AND RELIABILITY

ContextCraft made 1,024 evidence-tool calls across the full task suite (7.53 per query). KPI-change analysis dominates (479 calls), followed by trace latency analysis (266) and log burst analysis (249). KPI tools appear particularly valuable for component and reason discrimination, while trace-latency tools support timestamp localization. Log-based evidence contributes less frequently, likely reflecting limitations of OpenRCA's log data: no severity annotations, many components with no logs at all, and narrow scope.

5.6 DISCUSSION

The ablation suggests that the ACH loop improves accuracy over the same initial analysis, although it increases runtime and model usage.

The matched comparisons show that the accuracy–efficiency balance depends on the model. Candidate recall for component and reason is high while final accuracy is not, so the next version should focus less on generating candidates and more on ranking them.

Overall, ContextCraft offers a structured and auditable approach to tool augmented RCA. The results also point to clear areas for improvement: better time localization, stronger treatment of contradictory evidence, more precise token cost measurement, and more selective follow up tool calls.

6 LIMITATIONS AND FUTURE WORK

First, our evaluation covers three models from two families and provides an initial view of how accuracy and efficiency vary across them. We therefore do not claim model-independent improvements. More models, decoding settings, and repeated runs are needed to identify which capabilities support effective hypothesis ranking and whether the observed ordering is stable. The overlapping confidence intervals in the ACH comparison also limit the strength of causal conclusions that can be drawn from the current ablation.

Second, results depend on tool definitions, heuristics, and partly empirical hyperparameters. Several detectors share source telemetry and their evidence is not statistically independent, while verdict counts and hand-designed score functions are not calibrated probabilities. Systematic tuning, component-level tool ablations, and sensitivity analyses are needed before production use. The fixed toolkit improves reproducibility but may also miss incident-specific analyses that dynamically generated code could discover.

Third, OpenRCA was the only public dataset we found at a suitable scale and quality, which limits external validity. Its logs are sparse and lack severity annotations, and some benchmark labels may admit multiple operationally reasonable explanations. Evaluation on additional datasets and expert adjudication of ambiguous cases remain necessary.

Fourth, OpenRCA simplifies operational RCA to a root-cause tuple within a bounded window. Production incidents may be multi-causal, involve changes and history, and require remediation (Yu et al., 2024; Wang & Qi, 2024). We do not claim that the absolute scores transfer directly to this broader setting. The ACH machinery is not tied to OpenRCA, but still requires validation on real incidents with SRE engineers.

Finally, our evaluation measures diagnostic accuracy and resource use rather than human outcomes. An auditable matrix can expose the basis of a recommendation, but it does not guarantee that operators will interpret the evidence correctly or trust the agent appropriately. Future work should measure investigation time, calibration, usefulness of cited evidence, and failure recovery in controlled user studies. It should also examine privacy, access control, and safe handling of sensitive telemetry before deployment in production environments.

AI USE STATEMENT

Generative AI tools were used in a supporting role during research ideation and execution. This included literature retrieval, drafting and editing some parts of the paper, assistance with LaTeX, and writing and editing some parts of the agent implementation. The tools did not independently generate experimental data, make methodological decisions, or determine the interpretation of the results. The authors reviewed all AI-assisted text and code, verified the code through testing and inspection, and take full responsibility for the content, claims, and artifacts presented in this work.

REFERENCES

Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys)*, 2024. doi: 10.1145/3627703.3629553. URL https://arxiv.org/abs/2305.15778.

Gregorio Convertino, Dorrit Billman, Peter Pirolli, Jean Pierre Massar, and Jeff Shrager. The cache study: Group effects in computer-supported collaborative analysis. *Computer Supported Cooperative Work*, 17(4):353–393, 2008. doi: 10.1007/s10606-008-9078-2.

Shengkun Cui, Rahul Krishna, Saurabh Jha, and Ravishankar K. Iyer. Agentic structured graph traversal for root cause analysis of code-related incidents in cloud applications, 2025. URL https://arxiv.org/abs/2512.22113.

Mandeep K. Dhami, Ian K. Belton, and David R. Mandel. The analysis of competing hypotheses in intelligence analysis. *Applied Cognitive Psychology*, 2019. doi: 10.1002/acp.3550.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285–1298, 2017.

Y. Gao, Z. Cai, and B. Yang. Rcaflow: A workflow-informed hierarchical planning multi-agent system for root cause analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 300–308, 2026. doi: 10.1609/aaai.v40i1.36991. URL https://doi.org/10.1609/aaai.v40i1.36991.

Xiao Han, Saima Absar, Lu Zhang, and Shuhan Yuan. Root cause analysis of anomalies in multivariate time series through granger causal discovery. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=k38Th3x4d9.

Jr. Heuer, Richards J. Analysis of competing hypotheses. In *Psychology of Intelligence Analysis*, chapter 8, pp. 95–110. Center for the Study of Intelligence, Central Intelligence Agency, Washington, DC, 1999. URL https://www.cia.gov/resources/csi/static/Pyschology-of-Intelligence-Analysis.pdf.

Jin Huang, Pengfei Chen, Guangba Yu, Yilun Wang, Haiyu Huang, and Zilong He. Faasrca: Full lifecycle root cause analysis for serverless applications, 2024. URL https://arxiv.org/abs/2412.02239.

Cheng Ji and Huaiying Luo. Leveraging large language model for intelligent log processing and autonomous debugging in cloud ai platforms, 2025. URL https://arxiv.org/abs/2506.17900.

Taeyoon Kim, Woohyeok Park, Hoyeong Yun, and Kyungyong Lee. Why do ai agents systematically fail at cloud root cause analysis?, 2026. URL https://arxiv.org/abs/2602.09937.

Yichen Li, Yulun Wu, Jinyang Liu, Zhihan Jiang, Zhuangbin Chen, Guangba Yu, and Michael R. Lyu. Coca: Generative root cause analysis for distributed systems with code knowledge, 2025. URL https://arxiv.org/abs/2503.23051.

Chenghao Liu, Wenzhuo Yang, Himanshu Mittal, Manpreet Singh, Doyen Sahoo, and Steven C. H. Hoi. Pyrca: A library for metric-based root cause analysis, 2023. URL https://arxiv.org/abs/2306.11417.

Yu Luo, Jiamin Jiang, Jingfei Feng, Lei Tao, Qingliang Zhang, Xidao Wen, Yongqian Sun, Shenglin Zhang, and Dan Pei. From observability data to diagnosis: An evolving multi-agent system for incident management in cloud systems, 2025. URL https://arxiv.org/abs/2510.24145.

OpenTelemetry Authors. Opentelemetry documentation. https://opentelemetry.io/, 2026. Accessed: 2026-06-05.

Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jing Li, Gaogang Xie, and Dan Pei. Flow-of-action: SOP enhanced LLM-based multi-agent system for root cause analysis. In *Companion Proceedings of the ACM Web Conference 2025 (WWW Companion), Industry Track*, 2025. doi: 10.1145/3701716.3715225. URL https://arxiv.org/abs/2502.08224.

Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.*, 15(9):1779–1797, May 2022. ISSN 2150-8097. doi: 10.14778/3538598.3538602. URL https://doi.org/10.14778/3538598.3538602.

Markus Thill, Wolfgang Konen, and Thomas Bäck. Time series anomaly detection with discrete wavelet transforms and maximum likelihood estimation. In *Proceedings of the International Conference on Time Series and Forecasting (ITISE)*, Granada, Spain, 2017.

Thijs van Ede, Hojjat Aghakhani, Noah Spahn, Riccardo Bortolameotti, Marco Cova, Andrea Continella, Maarten van Steen, Andreas Peter, Christopher Kruegel, and Giovanni Vigna. DeepCASE: Semi-Supervised Contextual Analysis of Security Events. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022.

Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. Root-cause metric location for microservice systems via log anomaly detection. In *2020 IEEE International Conference on Web Services (ICWS)*, pp. 142–150, 10 2020. doi: 10.1109/ICWS49710.2020.00026.

Tingting Wang and Guilin Qi. A comprehensive survey on root cause analysis in (micro) services: Methodologies, challenges, and trends. *arXiv preprint arXiv:2408.00803*, 2024.

Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. RCAgent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*, 2024. doi: 10.1145/3627673.3680016. URL https://arxiv.org/abs/2310.16340.

Thorsten Wittkopp, Philipp Wiesner, and Odej Kao. Logrca: Log-based root cause analysis for distributed services. *arXiv preprint arXiv:2405.13599*, 2024.

Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. Microrca: Root cause localization of performance issues in microservices. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9. IEEE Press, 2020. doi: 10.1109/NOMS47738.2020.9110353. URL https://doi.org/10.1109/NOMS47738.2020.9110353.

Yong Xiang, Charley Peter Chen, Liyi Zeng, Wei Yin, Xin Liu, Hu Li, and Wei Xu. Simplifying root cause analysis in kubernetes with stategraph and llm, 2025. URL https://arxiv.org/abs/2506.02490.

Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. OpenRCA: Can large language models locate the root cause of software failures? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=M4qNIzQYpd.

Carter Yagemann, Simon P. Chung, Brendan Saltaformaggio, and Wenke Lee. Automated bug hunting with data-driven symbolic root cause analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pp. 320–336, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384544. doi: 10.1145/3460120.3485363. URL https://doi.org/10.1145/3460120.3485363.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.03629.

Guangba Yu, Pengfei Chen, Zilong He, Qiuyu Yan, Yu Luo, Fangyuan Li, and Zibin Zheng. Changerca: Finding root causes from software changes in large online systems. *Proc. ACM Softw. Eng.*, 1(FSE), July 2024. doi: 10.1145/3643728. URL https://doi.org/10.1145/3643728.

Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip Yu, and Ying Li. A survey of aiops in the era of large language models. *ACM Comput. Surv.*, 58(2), September 2025a. ISSN 0360-0300. doi: 10.1145/3746635. URL https://doi.org/10.1145/3746635.

Lingzhe Zhang, Tong Jia, Kangjin Wang, Weijie Hong, Chiming Duan, Minghua He, and Ying Li. Adaptive root cause localization for microservice systems with multi-agent recursion-of-thought, 2025b. URL https://arxiv.org/abs/2508.20370.

Lecheng Zheng, Zhengzhang Chen, Dongjie Wang, Chengyuan Deng, Reon Matsuoka, and Haifeng Chen. Lemma-rca: A large multi-modal multi-domain dataset for root cause analysis, 2025. URL https://arxiv.org/abs/2406.05375.

A Supplementary Material

A.1 Tool inventory and schemas

Table 2 lists the telemetry operations used in the evaluated pipeline. Initial-scan tools run before $H^{(0)}$ is generated; planner tools are selected during the evidence loop. Some operations serve both roles.

Table 2: Telemetry tools used by ContextCraft.

Tool	Stage	Role
dwt_mle_anomaly	Initial / planner	Point and short-window KPI anomalies
sublof_anomaly	Initial / planner	Local subsequence outliers
compare_components_for_incident	Initial / planner	STL-based cross-component ranking
get_service_graph_from_timeframe	Initial	Trace-derived service topology
topology_granger_screen	Initial	Upstream causal-direction tie-breaker
get_p10_from_metric_app	Initial	Low-success metric-tail minutes
get_relevant_logs_for_timeframe	Initial	Components present near candidate minutes
get_top_changing_kpis_for_component	Planner	KPI-family and reason evidence
get_trace_latency_spike_per_component	Planner	Comparative latency and peak-minute evidence
get_log_error_burst	Planner	Component-level error-burst evidence
stl_anomaly	Planner	Sustained KPI anomaly intervals

All tools accept the incident window plus tool-specific parameters and return structured records. Metric anomaly tools take a component and KPI; comparative tools take a component set; log and trace tools may scan all components or a planner-selected subset. This uniform interface lets the evaluator convert every result into the same verdict alphabet.

A.2 Implementation thresholds and safeguards

The planning prompt requests at most three new hypotheses per round. If the planner supplies none for a required dimension, a deterministic fallback can add up to three candidates. After tool execution, the extractor can add at most one component and one reason per round when a fixed threshold is met: at least 30% KPI-family anomaly share or a $2\times$ trace or log burst.

The final-decision LLM selects only values in the current hypothesis set, with reasons restricted to the controlled vocabulary. A deterministic post-processor replaces times outside the incident window and may replace a net-refuted or unsupported choice when another candidate has strictly more support. The pipeline does not require positive support in every dimension; when none exists, it falls back to the highest-ranked admissible candidate.

A.3 Structured and multi-agent RCA systems

RCAFlow, introduced by Gao et al. (2026), proposes a workflow-informed hierarchical planning pipeline that decomposes RCA into planning, evidence collection, and reasoning stages. Flow-of-Action (Pei et al., 2025) similarly injects Standard Operating Procedure knowledge into a multi-agent system. Specialized sub-agents can improve modularity and permit parallel collection, but add inter-agent messages, orchestration latency, and failure modes such as non-deterministic scheduling or dependency-driven retries (Luo et al., 2025).

Other work constrains agents through code or infrastructure structure. Praxis (Cui et al., 2025) traverses service and program dependence graphs; COCA (Li et al., 2025) grounds generation in code knowledge; and platform-specific systems use Kubernetes state graphs (Xiang et al., 2025) or serverless lifecycles (Huang et al., 2024). RCLAgent (Zhang et al., 2025b) emulates SRE drill-down through a multi-agent recursion-of-thought process. These systems show the value of structure, but may require source code, platform-specific graphs, or multiple coordinating agents. ContextCraft instead targets telemetry-only diagnosis with one agent, deterministic tools, and a budget-aware stopping rule. A complete RCAFlow runtime artifact bundle was unavailable for matched reproduction, so we do not report a direct numerical comparison.

A.4 Hypothesis recall and candidate sources

The hypothesis-generation stage is stronger than the final answer accuracy alone suggests. Table 3 reports whether the ground-truth field was present anywhere in the generated hypothesis list. The correct component is present in 70.4% of applicable Gemini 2.5 Pro cases and the correct reason in 82.1%. Time recall is lower at 49.4%, as time localization is sensitive to small shifts in telemetry and timestamp normalization.

Table 3: Ground-truth recall in the hypothesis set before final selection.

Field	Applicable	Gemini 2.5 Pro	Qwen3
Time	77 / 78	0.4935	0.5256
Component	71 / 71	0.7042	0.8873
Reason	78 / 78	0.8205	0.9487

For component and reason, the correct answer is often available to the agent, but the final ranking does not consistently select it. A large part of the remaining error is therefore a decision problem rather than a retrieval problem. Log-derived components contain the true component in 23.0% of queries, deterministic component priors in 25.2%, and metric-tail timestamps contain the true time in 24.4%. The early top-3 component and reason rankings reach 8.2% and 5.9%, respectively. The merged set $H^{(0)}$ is therefore more useful than any single early ranking.

A.5 Detailed task-type and root-cause results

OpenRCA assigns one integer task type to each nonempty subset of the three output dimensions. Table 4 makes this encoding explicit: its middle column gives the dimension set D that the agent must infer, alongside ContextCraft's accuracy for that task type under both evaluated models.

Table 4: Task-type encoding and ContextCraft accuracy under both models. The set D contains the root-cause dimensions missing from the query and therefore inferred by the agent. n is the Qwen3 query count per type.

`task_type`	Missing dimension set D	n	Gemini 2.5 Pro	Qwen3
1	{TIME}	25	0.2917	0.2600
2	{REASON}	22	0.1364	0.0455
3	{COMPONENT}	15	0.2667	0.1333
4	{TIME, REASON}	18	0.2639	0.2639
5	{TIME, COMPONENT}	18	0.3333	0.2500
6	{COMPONENT, REASON}	21	0.2976	0.3095
7	{TIME, COMPONENT, REASON}	17	0.1665	0.1371

Multi-dimension tasks can earn partial credit when any requested field is correct, whereas single-dimension tasks are binary. Yet type 7 remains weaker than types 4–6 despite requesting all three fields, under both models: partial credit alone does not explain the pattern, and jointly localizing all dimensions introduces additional ambiguity. Under Qwen3, type 2 (reason only) collapses to 0.0455 despite 94.9% reason recall, the clearest single instance of the ranking bottleneck.

Table 5: ContextCraft accuracy grouped by root-cause reason, under both models. n is the Qwen3 query count per reason.

Root-cause reason	n	Gemini 2.5 Pro	Qwen3
High CPU usage	4	0.5486	0.4175
Network packet loss	1	0.3300	0.0000
High memory usage	5	0.2917	0.3000
Network latency	16	0.2670	0.1250
High disk space usage	7	—	0.2386
High disk I/O read usage	34	0.1559	0.1688
JVM Out of Memory (OOM) Heap	6	0.1154	0.1667
High JVM CPU load	5	0.0000	0.2000

Both models are strongest on high CPU usage and weaker on disk I/O and JVM-specific failures, which likely require more targeted JVM evidence than the current general KPI-family and log tools provide. Qwen3 trades some classes against others, recovering non-zero accuracy on high JVM

CPU load while losing ground on network latency. Several classes have very few queries, so per-class differences between models are indicative rather than significant.

A.6 PROMPT TEMPLATES

ContextCraft uses a staged prompting pipeline aligned with the agent workflow in Section 4.2. The prompt templates are responsible for:

(i) parsing the natural-language incident description into a structured RCA task, (ii) generating ranked candidate hypotheses for each missing dimension, (iii) planning discriminative evidence-gathering tool calls, (iv) scoring evidence against hypotheses to populate the evidence matrix, (v) deciding whether evidence is sufficient to stop, and (vi) synthesizing the final RCA decision.

All prompts constrain the model to emit structured JSON to improve reliability and simplify downstream parsing.

A.6.1 QUERY PARSING

This prompt converts a natural-language RCA query into a structured task with time window, failure count, and task type. A key design choice is that the query window is treated as context, whereas the exact occurrence minute is considered missing only if the query explicitly asks for it.

Listing 1: Query parsing prompt

```
SYSTEM:
You parse natural-language root-cause queries into structured tasks.

A task may be missing one or more of these three dimensions:
  - time      - the exact occurrence minute is unknown
  - component - the failing component is unknown
  - reason    - the failure category is unknown

task_type encodes which combination of dimensions is missing:
  1: time only
  2: reason only
  3: component only
  4: time + reason
  5: time + component
  6: component + reason
  7: time + component + reason

Rules for determining task_type:
  - The query ALWAYS provides a time WINDOW (e.g. "between 18:00 and
    18:30").
    The window is context, not the answer.
  - "time" is missing ONLY if the query explicitly asks to identify, find
    ,
    pinpoint, or determine the exact occurrence time.
  - "component" is missing if the query says the component is unknown and
    asks you to identify it.
  - "reason" is missing if the query says the reason is unknown and asks
    you to identify it.
  - Do NOT add "time" to task_type just because a time range is mentioned
    .

Output only valid JSON.

USER:
Extract from the query:
  - raw_timeframe_start_str (ISO UTC: "YYYY-MM-DDTHH:MM:SSZ")
  - raw_timeframe_end_str   (ISO UTC: "YYYY-MM-DDTHH:MM:SSZ")
  - failure_count           (integer, default 1)
  - task_type               (integer 1-7, see system message)
```

```
Query:
{query}

Return JSON only.
```

A.6.2 HYPOTHESIS GENERATION

This prompt proposes exactly five ranked hypotheses for each missing RCA dimension. It uses anomaly-tool outputs and telemetry-derived priors, while restricting root-cause reasons to a fixed controlled vocabulary.

Listing 2: Hypothesis generation prompt

```
SYSTEM:
You are an expert SRE proposing root-cause hypotheses for a single
    incident.

Hypothesis dimensions:
  1) time      - UTC datetime, minute resolution.
  2) component - name of one component.
  3) reason    - one of the allowed root-cause labels:
                 high memory usage; network latency; network packet loss;
                 high JVM CPU load; JVM Out of Memory (OOM) Heap;
                 high disk I/O read usage; high disk space usage; high
    CPU usage.

Produce exactly 5 ranked hypotheses per dimension that is marked missing
    in
`missing`. Skip the other dimensions entirely.

Ranking rules:
  - Use the initial-analysis tool outputs (DWT, SubLOF, component
    comparison, KPI-family rollups, consensus, time clusters) as data-
    driven
    priors. The deterministic Stage A/B/C anchors in `
    deterministic_priors`
    are an additional signal, NOT the decisive one.
  - For TIME: prefer minutes that appear in multiple anomaly tools and
    inside
    the query window.
  - For COMPONENT: prefer components ranked high in `component_comparison
    `,
    DWT, and SubLOF. The trace P95 signal is unreliable for non-network
    failures (it elevates downstream victims).
  - For REASON: rank by which KPI family showed the strongest anomalies.
    Avoid defaulting to "high memory usage" unless memory KPIs clearly
    dominate the family rollup.

Output format:
  - JSON only, no markdown.
  - Times are UTC, minute-floored, with trailing "Z" and seconds ":00".
  - Use the exact key names below.
  - Omit any dimension that is not in `missing`.
USER:
Task context:
  timeframe_start:    {timeframe_start}
  timeframe_end:      {timeframe_end}
  missing_dims:       {missing_dims}
  missing:            {missing}

Available components:
  metric_component_names: {metric_component_names}
  trace_component_names:  {trace_component_names}
```

```
Initial-analysis tool outputs:
  dwt_results:                              {dwt_results}
  sublof_results:                           {sublof_results}
  component_comparison:                     {component_comparison}
  top_service_graph_edges:                  {top_edges}
  metric_app_p10_rr:                        {metric_app_p10_rr}
  relevant_components_from_logs:            {relevant_components}
  initial_time_clusters:                    {initial_time_clusters}
  initial_kpi_family_rollups:               {initial_kpi_family_rollups}
  initial_consensus_summary:                {initial_consensus_summary}
  initial_candidate_service_graph_summary: {
    initial_candidate_service_graph_summary}

Deterministic Stage A/B/C anchors (one input among many):
  {deterministic_priors}

Output schema (omit dimensions not in `missing`):

{{
  "time_hypotheses":      [{{"root cause occurrence datetime": "YYYY-MM-
    DDTHH:MM:00Z"}}],
  "component_hypotheses": [{{"root cause component": "ComponentName"}}],
  "reason_hypotheses":    [{{"root cause reason": "one of {root_causes
    }"}}]
}}

Return JSON only.
```

A.6.3 EVIDENCE-GATHERING PLAN

This prompt selects tool calls that are maximally discriminative for the current hypotheses and may add a small number of new hypotheses when the current set is incomplete.

Listing 3: Evidence planning prompt

```
SYSTEM:
You design tool-call plans to test root-cause hypotheses on a distributed
     system.

For each hypothesis you select tools that will either support or refute
    it.
You also propose new hypotheses when the existing list is missing a
    likely
candidate that the data clearly surfaces.

Tool-to-hypothesis matching:
  - COMPONENT hypothesis (which service is the cause?)
    * Primary: `compare_components_for_incident` - ranks ALL components.
    * Secondary: `get_trace_latency_spike_per_component`, `
    get_log_error_burst`.
    * Do not use `get_top_changing_kpis_for_component` for component
      discrimination; it examines one component in isolation.

  - REASON hypothesis (which failure category?)
    * Primary: `get_top_changing_kpis_for_component` on the top 2-3
    component
      candidates. `family_dominance` directly maps to reason hypotheses.

  - TIME hypothesis (which minute?)
    * `dwt_mle_anomaly` for short spikes / point anomalies.
    * `stl_anomaly` for sustained intervals.
    * `sublof_anomaly` to discriminate between components on the same KPI
     family.
    * `get_trace_latency_spike_per_component` for peak_minute.
```

```
Convergence rules:
  - Once a hypothesis has supporting evidence, prefer discriminative tool
    calls that test competing candidates instead of piling on more
    support.
  - Do not call all three anomaly tools on the same hypothesis unless
    each
    call answers a different uncertainty.

New-hypothesis rules:
  - You may add at most 3 new hypotheses per round.
  - Do not repeat hypotheses already present in `hypotheses`.
  - Do not re-propose candidates already refuted in `
    existing_evidence_matrix`
    unless new evidence contradicts the refutation.
  - If `must_add_*_candidate` is true for a dimension, you MUST include
    at
    least one new candidate in that dimension's `new_hypotheses` list.

Output format:
  - STRICT JSON, double quotes throughout.
  - No markdown, no commentary.
  - No placeholder values like "path" or "/path/to/dataset" - omit
    dataset_root.
  - Every tool call has a `justification` field ($\leq 1$ sentence)
    saying what
    uncertainty it reduces.

USER:
Task context:
  timeframe_start: {timeframe_start}
  timeframe_end:   {timeframe_end}
  missing_dims:    {missing_dims}

Available tools and their parameters:
  {tool_list}

Initial-analysis context:
  dwt_results:                {dwt_results}
  sublof_results:             {sublof_results}
  component_comparison:       {component_comparison}
  top_service_graph_edges:    {top_edges}
  metric_app_p10_rr:          {metric_app_p10_rr}
  relevant_components_from_logs: {relevant_components}
  metric_component_names:     {metric_comps}
  trace_component_names:      {trace_comps}

Hypotheses to test:
  {hypotheses}

Prior-round state (may be empty):
  existing_evidence:          {existing_evidence}
  existing_evidence_matrix:   {existing_evidence_matrix}

Expansion requirements:
  must_add_time_candidate:      {must_add_time_candidate}
  must_add_component_candidate: {must_add_component_candidate}
  must_add_reason_candidate:    {must_add_reason_candidate}

Return JSON in this exact schema:

{{
  "plan": {{
    "evidence_gathering_plan": [
      {{
```

```
        "hypothesis": {{"<dimension_key>": "<value>"}},
        "tool_calls": [
          {{
            "tool_name": "<tool>",
            "justification": "<short>",
            "parameters": {{"param1": "value1"}}
          }}
        ]
      }}
    ]
  }},
  "new_hypotheses": {{
    "time_hypotheses":      [{{"root cause occurrence datetime": "YYYY-MM
    -DDTHH:MM:00Z"}}],
    "component_hypotheses": [{{"root cause component": "ComponentName
    "}}],
    "reason_hypotheses":    [{{"root cause reason": "one of {root_causes
    }"}}]
  }}
}}

Return JSON only.
```

A.6.4 EVIDENCE-MATRIX EVALUATION

This prompt scores each (hypothesis, tool-call) pair with one of four verdicts: $V = \{supports,\ weak_support,\ not_relevant,\ refutes\}$. It operationalizes the evidence matrix discussed in Section 4.1.

Listing 4: Evidence evaluation prompt

```
SYSTEM:
You evaluate tool outputs and label each (hypothesis, tool-call) pair
    with a verdict.

Verdict set:
  - supports     - strong, direct evidence for the hypothesis.
  - weak_support - partial / indirect evidence.
  - refutes      - direct contradiction.
  - not_relevant - tool output cannot test this hypothesis.

CRITICAL SCORING PRINCIPLE:
  When a tool examines ONE hypothesis in isolation (e.g.,
  get_top_changing_kpis for a single component), judge purely by SIGNAL
  STRENGTH on that hypothesis. Strong anomaly = supports, moderate =
  weak_support, no anomaly = refutes. Do not refute a hypothesis merely
  because that tool call cannot compare it with other hypotheses.

  "supports" is the default when the tool finds a clear anomaly. Reserve
  "refutes" for an explicit absence of anomaly or a contradiction.

Scoring rules:

TIME hypotheses (root cause occurrence datetime), tolerance 1 minute:
  - Within $\pm$1 min of strongest anomaly $\rightarrow$ supports.
  - Within $\pm$3 min $\rightarrow$ weak_support.
  - Within $\pm$10 min $\rightarrow$ weak_support only if no closer
    anomaly exists.
  - Anomalies all > 10 min away $\rightarrow$ refutes.
  - No time-localized anomaly info $\rightarrow$ not_relevant.

COMPONENT hypotheses:
  - Strong anomaly (z>4 or ratio>3x) for this component $\rightarrow$
    supports.
```

```
  - Moderate anomaly (z=2-4 or ratio 1.5-3x) $\rightarrow$ weak_support.
  - Ranked #1 or #2 in a multi-component tool $\rightarrow$ supports.
  - Ranked #3-#5 $\rightarrow$ weak_support.
  - No anomaly detected or burst_ratio $\leq 1$ $\rightarrow$ refutes.

REASON hypotheses (KPI-family match):
  - Anomalous KPIs belong to the hypothesis's family $\rightarrow$
    supports.
  - Mixed families with some match $\rightarrow$ weak_support.
  - Anomalous KPIs all belong to a different family $\rightarrow$ refutes
    .
  - Tool not family-discriminating $\rightarrow$ not_relevant.

Output rules:
  - JSON only, no markdown.
  - Compact: do NOT include `parameters` or `justification` in the output
    .
  - `explanation` must be $\leq$ 15 words.
  - `hypothesis` field must match exactly one input hypothesis.

USER:
Task context:
  timeframe_start: {timeframe_start}
  timeframe_end:   {timeframe_end}
  missing_dims:    {missing_dims}

Inputs:
  tool_results:     {tool_results}
  hypotheses:       {hypotheses}
  initial_analysis: {initial_analysis}

Return JSON in this exact schema:

{{
  "evidence_matrix": [
    {{
      "hypothesis": {{"<dimension_key>": "<value>"}},
      "evidence": [
        {{
          "tool_name": "<tool>",
          "tool_call_id": "<id>",
          "verdict": "supports" | "weak_support" | "refutes" | "
    not_relevant",
          "explanation": "<$\leq$15 words>"
        }}
      ]
    }}
  ]
}}
```

A.6.5 EVIDENCE SUFFICIENCY

This prompt decides whether the collected evidence is already sufficient to identify a single best candidate in each unresolved dimension.

Listing 5: Evidence sufficiency prompt

```
SYSTEM:
You decide whether the evidence matrix is sufficient to converge on a
    single best candidate per missing dimension.

Return "yes" only if, for EVERY required missing dimension, a single
candidate clearly stands out (more supports than alternatives, and clear
separation from competitors).
```

```
Return "no" when:
  - Multiple candidates in the same dimension have comparable support.
  - Some candidates have only `weak_support` and no `supports`.
  - The strongest candidate is also refuted by another tool.

Output JSON only:
  {{ "verdict": "yes" | "no",
     "explanation": "<one short sentence>" }}

USER:
Inputs:
  evidence_matrix: {evidence_matrix}
  hypotheses:      {hypotheses}
  missing_dims:    {missing_dims}
  root_causes:     {root_causes}

Evaluate only the dimensions listed in `missing_dims`. If any one of them
lacks a clear single best candidate, return "no" and explain what
discriminative evidence is still needed.

Return JSON only.
```

A.6.6 FINAL DECISION

This final prompt selects the output root cause. The evidence matrix is treated as the authoritative signal; consensus summaries and deterministic priors are used only as tie-breakers.

Listing 6: Final decision prompt

```
SYSTEM:
You are an expert SRE producing the final root-cause prediction for a
    single incident.

You will receive:
  1.  hypotheses             - candidates per dimension (time, component,
     reason).
  2.  evidence_matrix        - verdicts per (hypothesis, tool) pair.
  3.  enough_evidence_result - the prior step's converge-yes/no verdict.
  4.  matrix_scorecard       - aggregated supports/refutes per dimension.
  5.  admissible_candidates  - the only values you are allowed to output.
  6.  compressed_negative_evidence - strongest refutations.
  7.  consensus_summary      - cross-tool agreement.
  8.  kpi_family_rollups     - KPI-family anomaly aggregates.
  9.  candidate_service_graph_summary - graph context for component
    candidates.
 10.  deterministic_priors   - Stage A/B/C anchors derived from telemetry
    .
 11.  topology_granger_screen - pairs of (caller, callee) where the
    caller's
        KPI Granger-causes the callee's KPI. Components appearing as
        `source` in causal edges are upstream of the failure on that path
    .

Decision policy (in priority order):
  1. The evidence matrix is authoritative. Prefer candidates with more
     `supports` than `refutes`. `weak_support` counts as half a support.
     `not_relevant` rows do not contribute.
  2. For each missing dimension, pick the top-ranked admissible candidate
     in `matrix_scorecard`.
  3. If the matrix is tied, use `consensus_summary` (cross-tool agreement
    ).
  4. If still tied, use `deterministic_priors` as a tiebreaker.
```

```
  5. Never select a candidate that appears in `
    compressed_negative_evidence`
     with strong refutation when an admissible non-refuted alternative
     exists.
  6. If no candidate has positive net support in a dimension, pick the
     highest-ranked admissible candidate from `matrix_scorecard`.

Time-dimension guard (default to the prior unless the matrix says
    otherwise):
  - `deterministic_priors.predicted_time_iso_utc` is the median of three
    independent telemetry signals and is on a 1-minute grid. The OpenRCA
    scoring tolerance is also $\pm$1 minute.
  - Default to that prior minute. Only deviate when the evidence_matrix
    scorecard ranks a different admissible candidate STRICTLY higher
    (strictly more `supports` AND a higher overall score).
  - If `time_candidates` shows 2+ sources agreeing within 2 minutes, the
    prior is doubly trustworthy - do not deviate.
  - Do NOT pick a neighbor minute ($\pm$1, $\pm$2) just because proximity
    -injection
    rated it as "supports". Those neighbors exist as a safety net for
    cases
    where the LLM had a better candidate; they are not better than the
    median prior.

Reason-aware component selection (when both component and reason are
    missing):
  - If the strongest-supported reason family is metric-local (memory, CPU
    ,
    disk, JVM), prefer the candidate with the highest container-metric
    anomaly score. Trace-propagation evidence often elevates downstream
    victims for these reasons.
  - If the strongest-supported reason family is network, prefer the
    candidate marked `topology_role: cause` over `topology_role: victim`.
    A candidate with no topology role (trace-invisible) is neutral.

Topology Granger tiebreaker (component dimension):
  - `topology_granger_screen.causal_edges` lists (source, target) pairs
    where the source's KPI statistically Granger-causes the target's KPI
    over the pre-incident window.
  - When two admissible component candidates have similar evidence_matrix
    scores, prefer the one that appears as `source` in a causal edge over
    the one that appears as `target`.

Hard constraints:
  - Output only the dimensions listed in `missing_dims`. For all other
    dimensions, output "".
  - All values must come from `admissible_candidates` (datetime values
    may be normalized for ISO-8601 UTC formatting).
  - Datetimes must be ISO-8601 UTC with trailing Z: "YYYY-MM-DDTHH:MM:SSZ
    ".
  - Reason must be one of the allowed root causes.
  - Never output placeholders ("Unknown", "ComponentName", "None", etc.).
  - Top-level JSON key is the string "1" exactly.
  - Return JSON only - no markdown, no commentary.
USER:
Task context:
  timeframe_start: {timeframe_start}
  timeframe_end:   {timeframe_end}
  missing_dims:    {missing_dims}

Inputs:
  hypotheses:                    {hypotheses}
  evidence_matrix:               {evidence_matrix}
  enough_evidence_result:        {enough_evidence_result}
  matrix_scorecard:              {matrix_scorecard}
```

```
  admissible_candidates:            {admissible_candidates}
  compressed_negative_evidence:     {compressed_negative_evidence}
  consensus_summary:                {consensus_summary}
  kpi_family_rollups:               {kpi_family_rollups}
  candidate_service_graph_summary:  {candidate_service_graph_summary}
  deterministic_priors:             {deterministic_priors}
  topology_granger_screen:          {topology_granger_screen}

Produce the final answer in exactly this schema:

{{
  "1": {{
    "root cause occurrence datetime": "YYYY-MM-DDTHH:MM:SSZ",
    "root cause component": "ComponentName",
    "root cause reason": "one of {root_causes}"
  }}
}}

Fill only the fields whose dimension is in `missing_dims`. Set the others
    to "".
Return JSON only.
```

A.7 Additional results and case studies

Tables 3–5 supplement the aggregate comparison with candidate-recall, task-type, and root-cause breakdowns. Together they show a recurring pattern: the correct component or reason is often generated but not selected, especially when evidence is mixed across telemetry families. Qualitative inspection likewise finds the clearest successes when independent metric, trace, or log signals agree. These observations motivate improving evidence calibration and final candidate ranking rather than simply expanding the hypothesis set.

Author Contributions

All authors contributed to the conceptualization, methodology, implementation, experiments, analysis, and writing. All authors reviewed and approved the final manuscript.

Acknowledgments

We thank colleagues who provided feedback on the system design and evaluation, as well as the anonymous reviewers for their time and comments. The views expressed in this paper are those of the authors.