

Grounded or Guessing? An Empirical Evaluation of LLM Reasoning in Agentic Workflows for Root Cause Analysis in Cloud-based Systems

by

Evelien Riddell

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Evelien Riddell 2026

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Root cause analysis (RCA) is essential for diagnosing failures within complex software systems to ensure system reliability. The highly distributed and interdependent nature of modern cloud-based systems often complicates RCA efforts, particularly for multi-hop fault propagation, where symptoms appear far from their true causes. Recent advancements in Large Language Models (LLMs) present new opportunities to enhance automated RCA. In particular, LLM-based agents offer autonomous execution and dynamic adaptability with minimal human intervention. However, their practical value for RCA depends on the fidelity of reasoning and decision-making. Existing work relies on historical incident corpora, operates directly on high-volume telemetry beyond current LLM capacity, or embeds reasoning inside complex multi-agent pipelines—conditions that obscure whether failures arise from reasoning itself or from peripheral design choices.

In this thesis, we present a focused empirical evaluation that isolates an LLM’s reasoning behaviour. We design a controlled experimental framework that foregrounds the LLM by using a simplified experimental setting. We evaluate six LLMs under two agentic workflows (REACT and *Plan-and-Execute*) and a non-agentic baseline on two real-world case studies (GAIA and OpenRCA). In total, we executed 48,000 simulated failure scenarios, totalling 228 days of execution time. We measure both root-cause accuracy and the quality of intermediate reasoning traces. We produce a labelled taxonomy of 16 common RCA reasoning failures and use an LLM-as-a-Judge for annotation. Our results clarify where current open-source LLMs succeed and fail in multi-hop RCA, quantify sensitivity to input data modalities, and identify reasoning failures that predict final correctness. Together, these contributions provide transparent and reproducible empirical results and a failure taxonomy to guide future work on reasoning-driven system diagnosis.

Acknowledgements

I would like to thank my supervisor, Krzysztof Czarnecki, and Michał Antkiewicz for their guidance on this cutting-edge research topic.

I am deeply indebted to James Riddell for his extensive collaboration throughout this project. His critical insights during our brainstorming sessions and rigorous feedback were instrumental in shaping the direction and quality of this work. My gratitude also extends to Gengyi Sun for her encouragement and invaluable feedback, as well as my lab members, Yimu Wang and Adrian Chow, for their support.

On a personal note, I have profound gratitude for my husband, James, as well as my parents and sister; their unwavering love and support have been my pillars of strength during challenging times. I also wish to thank my friends for their enduring support throughout the entirety of my master's program.

Dedication

To James

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Motivations	2
1.2 Research Questions	3
1.3 Contributions	4
1.4 Thesis Structure	4
2 Background and Definitions	6
2.1 Cloud-based Systems	6
2.2 Root Cause Analysis (RCA)	7
2.3 Knowledge Graphs (KGs)	8
2.4 Large Language Models (LLMs) and Agentic Workflows	9
2.5 Problem Definition	10

3	Study Methodology	12
3.1	Data Preparation	13
3.1.1	Alert Extraction	13
3.1.2	Alert Unification	15
3.1.3	System Knowledge Graph (KG) Construction	15
3.2	RCA Inference	17
3.2.1	Workflows	17
3.2.2	Agent Tools	18
3.2.3	Input Prompts	19
3.3	Evaluation	19
3.3.1	Output Quality	19
3.3.2	Reasoning Quality	20
4	Experimental Setup	23
4.1	Experimental Data	23
4.2	LLMs	24
4.3	Implementation and Settings	25
5	Results	26
5.1	RQ_1 : How effectively can an LLM agent perform RCA tasks?	26
5.2	RQ_2 : How sensitive are final RCA outcomes to alert modalities (logs, metrics, traces)?	30
5.3	RQ_3 : What reasoning failures appear in agent inference traces?	32
5.4	RQ_4 : How does the presence of reasoning failures affect the likelihood of generating correct RCA hypotheses?	36
6	Discussion	38
6.1	The “Agentic Tax” and Competency Threshold	38
6.2	Domain-Grounding vs. General Reasoning	39
6.3	More Data is Not Always Better	40
6.4	The Danger of “Right for the Wrong Reasons”	40

7	Limitations and Threats to Validity	42
7.1	Limitations	42
7.2	Internal Validity	43
7.3	External Validity	43
8	Related Work	44
8.1	Data-driven RCA	44
8.2	LLM-based RCA	45
8.3	Evaluation of LLM Reasoning	46
8.3.1	General Evaluation Strategies for LLM Reasoning	46
8.3.2	Evaluation Approaches in LLM-based RCA	47
9	Conclusion	48
	References	50
	APPENDICES	64
A	Additional Data	65
A.1	Dataset Details	65
A.2	System Knowledge Graph Entity Specifications	66
A.2.1	Reasoning Failure Taxonomy	67
A.3	Results	67
A.3.1	Accuracy	67
A.3.2	LLM Inference Times	68
A.3.3	Sensitivity to Input Representation Strategies	68
A.4	Prompts	72

List of Figures

2.1	<i>Straight-shot</i> (direct answer) and REACT agent workflows.	9
3.1	Overview of the study method.	12
3.2	The alert extraction and unification process using traces, logs, and metrics.	13
3.3	A region of the system KG for MicroSS [11]. Blue, orange, green, and yellow nodes represent <i>Service</i> , <i>Service Instance</i> , <i>Host</i> , and <i>Cache</i> type entities, respectively. Only <i>instance-of</i> , <i>has-instance</i> , <i>hosted-on</i> , <i>hosts</i> , <i>control-flow</i> , and <i>data-flow</i> relationships are shown for simplicity.	16
5.1	Accuracy results aggregated across \mathcal{A} and \mathcal{B} (lower bound is A@1, upper bound is A@3, and dotted line is Avg@3).	28
5.2	Change in accuracy (ΔA -Avg@3) for withheld alert modalities aggregated across datasets \mathcal{A} and \mathcal{B} . Solid bars denote statistically significant differences ($p < 0.05$) according to the Wilcoxon signed-rank test, while faded bars indicate changes that are not statistically significant.	31
5.3	Prevalence of reasoning failures (RF) in RCA outputs.	32
5.4	Risk difference (RD, Wilson 95% CI) and relative risk (RR, log scale) for the top-12 RFs.	35
A.1	Reasoning failures taxonomy. Columns partition failure <i>categories</i> and horizontal swimlanes indicate the <i>scope</i> in which the failure manifests.	68
A.2	Accuracy results for datasets (a) \mathcal{A} and (b) \mathcal{B} (lower bound is A@1, upper bound is A@3, and dotted line is Avg@3).	69

A.3	Change in accuracy (ΔA -Avg@3) for (a) <i>alert unification</i> and (b) <i>KG representation</i> strategies. In both figures, solid bars denote statistically significant differences ($p < 0.05$) according to the Wilcoxon signed-rank test, while faded bars indicate changes that are not statistically significant. . .	71
A.4	Input prompt template for REACT workflow.	73
A.5	LLM-as-a-Judge prompt template with the reasoning failure taxonomy and step-by-step annotation workflow.	80

List of Tables

3.1	Description of Tools in Agentic Workflows	18
3.2	Reasoning Failure Taxonomy	21
4.1	Experimental Data Details	24
5.1	Accuracy Results	27
5.2	Proportion of Reasoning Failures by Workflow	33
A.1	Distribution of Injected Faults for Dataset \mathcal{A}	65
A.2	Distribution of Injected Faults for Dataset \mathcal{B}	66
A.3	High-level System Entity Types	67
A.4	Average LLM Inference Time Per Scenario	70

Chapter 1

Introduction

Modern software systems are increasingly built on cloud-native, microservice-based architectures. These systems promise scalability, flexibility, and resilience, but introduce new challenges in reliability and observability. In such distributed environments, a single failure can cascade across multiple components, making *root cause analysis* (RCA)—the task of identifying the originating fault behind a system failure—both essential and complex.

RCA in cloud-based systems typically involves analyzing telemetry data (e.g., logs, metrics, traces) [63, 62] to trace failure propagation and isolate the faulty component. However, the sheer scale, heterogeneity, and volume of telemetry in production environments can overwhelm automated tools and human operators alike [9]. In response, recent work has explored data-driven approaches to efficiently process vast amounts of telemetry data and generate informative clues toward the true root cause for further manual analysis performed by reliability engineers or software operators [62].

The recent emergence of Large Language Models (LLMs) introduces a promising new paradigm for automated or human-in-the-loop RCA. Thanks to their general reasoning and tool-use capabilities, LLMs offer the potential to navigate system knowledge and extract insights from both structured (e.g., incident reports) and unstructured data (e.g., logs) [71]. This makes them attractive candidates for autonomous RCA agents capable of integrating system information with observed system behaviour. However, important methodological and evaluative gaps remain that prevent us from understanding how adept LLMs *actually* are for RCA.

1.1 Motivations

The capabilities of LLMs are often obscured in existing LLM-based RCA approaches. LLM-based RCA frameworks frequently combine several sequential tasks (e.g., anomaly detection and fault analysis) [25, 79] into a single pipeline without intermediate evaluation. For example, OpenRCA [79] presents an evaluation framework for LLM-based RCA tasks, but frames each task as a sequence of sub-tasks, including anomaly detection, failure identification, and fault analysis. This conflates the core RCA ability with unrelated skills such as anomaly detection, code generation, and error handling.

Others distribute sub-tasks across specialized agents [25, 73, 92, 48, 79]. With the promise of agentic workflows and multi-agent frameworks, however, comes added complexity: RCA outputs reflect the combined contributions of multiple components (e.g., coordinator, analyzer, code-executor) [73, 92, 79]. This entanglement makes it difficult to assess individual agent contributions, particularly while discerning between the primary reasoning thread and auxiliary tasks or inter-agent chatter. Some also incorporate static protocols with incident-specific handlers and data collection mechanisms [9].

While such setups show promise, they entangle reasoning with opaque task boundaries, inter-agent dependencies, and system-specific heuristics. This makes it difficult to attribute performance bottlenecks or errors to the LLM and to isolate and evaluate the actual reasoning capabilities, particularly in multi-hop failure scenarios that require careful propagation analysis across interacting system entities. Therefore, to assess LLMs as effective RCA agents, we require evaluation settings where reasoning behaviour can be meaningfully surfaced and analyzed in isolation. Yet even if reasoning is surfaced, current evaluation practices provide limited visibility into its quality.

Existing evaluation metrics overlook reasoning quality and propagation correctness. Most current evaluations of LLM-based RCA approaches report accuracy and efficiency of the final output, while reasoning evaluation (if any) is often limited to brief, aggregated human assessments, culminating in a single “usefulness” score [92], “correctness” score [19], or post-hoc “reasoning error” count [58], offering little transparency and interpretability. Moreover, many rely on metrics capturing lexical or semantic similarity between predicted and reference root-cause descriptions [1, 73, 96, 58, 19], which can misrepresent partially incorrect diagnoses and again offer little interpretability. Furthermore, these approaches also struggle in multi-hop scenarios, where understanding the inferred propagation path and reasoning steps is crucial.

No prior work systematically evaluates reasoning quality for the RCA task. This lack of systematic, reproducible, and automated evaluation for reasoning quality constrains

researchers from conducting large-scale evaluation and comparison across studies. To understand and improve LLM performance as RCA agents, we need evaluation protocols that expose not only what was predicted, but how and why it was inferred.

In short, despite the growing use of LLMs for RCA, **we lack a clear, empirical understanding of how well singular models can reason** about fault propagation, which factors most influence their performance, and what kinds of reasoning failures they exhibit. To address this gap, we conduct a systematic empirical evaluation in a controlled RCA setting that isolates the LLM’s reasoning behaviour from confounding factors (e.g., multiple sequential/overlapping tasks, inter-agent dependencies, system-specific heuristics) and analyzes both its final RCA outputs and reasoning traces.

To enable such an analysis, we design an RCA environment that deliberately minimizes external sources of complexity. Unlike prior multi-agent or heavily engineered frameworks, we design a controlled experimental framework that isolates an LLM’s reasoning by using simple agent architectures and deterministic, semantically meaningful tools. Specifically, we focus solely on the post-anomaly-detection RCA task: the model receives pre-identified anomalous alerts (rather than raw telemetry or historical incident descriptions) and is provided structured system context via a typed knowledge graph (KG), while avoiding incident-specific heuristics or open-ended code execution. We also systematically vary the representation of this graph and alert information to assess robustness against prompt formulation. These design choices—simple agents, deterministic tools, alert-level inputs, and an explicit KG—reduce confounding factors (e.g., noisy telemetry, multi-agent contribution, lack of clear task separation) and enable more interpretable reasoning evaluation.

Importantly, we emphasize that a simplified RCA setting does not trivialize the RCA task: multi-hop failure propagation across distributed services still presents significant reasoning challenges. Rather, the controlled environment shifts the focus away from data wrangling or auxiliary analysis tasks toward a focused evaluation of output accuracy, propagation inference, and intermediate reasoning traces in a reproducible and interpretable way. To assess these traces beyond surface-level output matching, we construct a taxonomy of reasoning failures and employ a calibrated LLM-as-a-Judge evaluator to systematically identify these failures at scale.

1.2 Research Questions

We focus our study with the following research questions:

RQ₁: How effectively can an LLM agent perform RCA tasks?

- RQ₂**: How sensitive are final RCA outcomes to alert modalities (logs, metrics, traces)?
- RQ₃**: What reasoning failures appear in agent inference traces?
- RQ₄**: How does the presence of reasoning failures affect the likelihood of generating correct RCA hypotheses?

1.3 Contributions

The main contributions of this thesis are:

1. An evaluation framework that isolates LLM-based RCA reasoning performance from confounding factors.
2. A human-rated LLM-as-a-Judge evaluator capable of assessing RCA reasoning quality automatically and reproducibly.
3. An extensive empirical study (48,000 samples, 228 days of execution time) on the capabilities of LLMs for RCA under several agentic and one non-agentic workflows.
4. A complete replication package [56] with our implementation, evaluation setup, and results.

To the best of our knowledge, this study is the first empirical investigation into the isolated ability of LLM-based RCA agents and their reasoning failures.

1.4 Thesis Structure

The remainder of this thesis is structured as follows.

- Chapter 2 outlines the necessary background on RCA, agentic workflows, and contains our problem definition;
- Chapter 3 describes the methodology used to answer our research questions;
- Chapter 4 lists the experimental setup for executing the methodology;
- Chapter 5 presents results and answers to the research questions;

- Chapter 6 discusses our main findings, implications, and future work avenues;
- Chapter 7 lists the limitations and threats to validity in conducting our study;
- Chapter 8 outlines the related work in RCA and evaluating LLM outputs;
- Chapter 9 summarizes the thesis and outlines its main conclusions.

Chapter 2

Background and Definitions

In this chapter, we introduce the background and definitions of this thesis. We first define the core characteristics of cloud-native systems and the observability artifacts essential for diagnosing their failures (Section 2.1). We then introduce the RCA task and its inherent challenges for such systems (Section 2.2). Subsequently, we define KGs as a structured representation method (Section 2.3) and introduce LLMs and agentic workflows (Section 2.4). We conclude this chapter with a formal problem definition (Section 2.5).

2.1 Cloud-based Systems

Cloud-based systems are applications and services deployed on elastic, shared infrastructure provided by cloud platforms [43]. They are characterized by on-demand resource allocation, multi-tenancy, and programmatic management of compute, storage, and network resources. These platforms emphasize scalability, elasticity, and rapid iteration, which in turn shape how systems are designed, observed, and diagnosed.

Microservices are a common architectural style in cloud systems [15]. Microservice architectures decompose applications into small, loosely coupled, and independently deployable services, each encapsulating a distinct business capability. For example, Online Boutique [49], a web-based e-commerce application, is composed of 11 microservices, such as checkout, payment, and shipping.

In cloud-native systems, **system entities** are distinct components that can be individually managed and monitored [22]. Examples include (micro)services (self-contained functional units), (micro)service instances (individual deployments of services), hosts (physical

or virtual machines), data stores (data storage systems, e.g., databases, caches), coordination managers (components responsible for distributed process orchestration and state synchronization), load balancers, and network elements (components that support inter-service communication, e.g., proxies, service meshes).

Observability is the ability to infer a system’s internal state from its external outputs [63, 62]. It typically relies on three core pillars of monitoring data (a.k.a. **telemetry data**): logs, metrics, and traces. **Logs** consist of structured or unstructured records of discrete time-stamped events within the system. **Metrics** are structured, numeric measurements collected at regular intervals that reflect system performance over time, such as resource utilization (CPU, memory, disk). **Traces** are records that capture the end-to-end flow of a request through the system’s components, helping identify system call relationships and timing information. Figure 3.2 (under “Telemetry Data”) shows an example of each modality. Collectively, these form the critical observability artifacts that support monitoring, post-hoc analysis, and fault diagnosis—a necessary precursor to identifying the causes of system failures.

2.2 Root Cause Analysis (RCA)

A **failure** is an observable deviation from expected service behaviour, such as degraded performance, emitted anomaly events or alerts, or incorrect outputs. A **fault** is the underlying cause that precipitates a failure [5, 45]. **Root cause analysis (RCA)** is the a posteriori process of analyzing telemetry data to identify underlying causes of observed system failures. Formally, given a set of observable symptoms (e.g., anomalous telemetry data, alerts, or a user report) and the telemetry collected during the incident, the RCA task is to identify a (usually small, ranked) set of root-cause *locations* (e.g., entities) and/or root-cause *fault types* whose occurrence explains the observed symptoms.

RCA is not to be confused with debugging techniques, which often involve re-running applications in a testing environment to reproduce observed anomalies or faults. RCA techniques instead operate on production-level telemetry data, without re-running production workloads [62]. In practice, RCA may operate at different granularities (e.g., system entity, code block, configuration) and different time horizons (e.g., instantaneous faults, slow degradations, or intermittent flakiness). The granularity choice affects how precisely an RCA method can localize a fault.

Despite significant efforts in service maintenance, faults are inevitable due to the sheer scale of cloud-based systems and their complex dependencies [42]. Failure to promptly

locate and recover from these faults can result in user dissatisfaction or significant financial losses [89]. Therefore, RCA is crucial for maintaining cloud-based online services. However, efficiently and accurately performing RCA in these environments faces several challenges [62, 71, 89, 17]:

- (1) Due to the modularity of service-oriented systems, frequent updates and deployments result in rapidly evolving systems, thereby changing system behaviour (and entity dependencies) over time.
- (2) High dimensionality, cardinality, and volume of telemetry data make exhaustive analysis expensive. Moreover, telemetry can be noisy [30].
- (3) Telemetry data management overhead means practitioners must make explicit trade-offs (e.g., sampling, aggregation, and retention policies) that directly influence the quality and fidelity of the available telemetry for RCA. As a result, partial observability (due to sampling frequency, data retention limits, and instrumentation gaps) means that not all events are captured, and failures may be invisible to the telemetry pipeline.
- (4) Downstream failures can hide or amplify upstream causes, making localization difficult. A single user request may traverse dozens of loosely coupled services, and a fault in one service can propagate unpredictably, causing a cascade of failures across the system.

Traditionally, RCA is performed by reliability engineers who must inspect heterogeneous telemetry data, apply domain expertise, and iterate on various potential origins until the true cause is found. However, the aforementioned practical constraints motivate automated and semi-automated RCA methods that reduce time to resolution and help engineers prioritize likely causes.

2.3 Knowledge Graphs (KGs)

Knowledge graphs (KGs) are structured representations of a set of entities and their relationships, typically modelled as directed, labelled graphs. Formally, a KG is a collection of facts structured as subject-predicate-object triplets (s, p, o) , where s and o are entities (nodes) and p is a directed, labelled relationship (edge) between them [28].

In software systems, KGs can encode architectural structure, network topology, and operational dependencies, thereby revealing underlying connections between system entities and offering a comprehensive view of system structure and behaviour [89]. For RCA, KGs can be used to model relationships between different system entities, providing a critical structural context for localizing faults and understanding propagation paths. The semantic richness of KGs makes them well-suited for integration with LLMs for relational reasoning and fault propagation analysis [76].

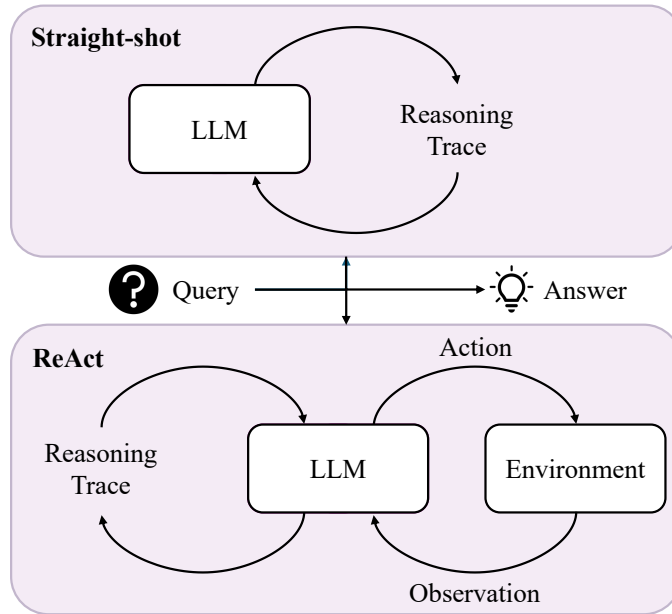


Figure 2.1: *Straight-shot* (direct answer) and REACT agent workflows.

2.4 Large Language Models (LLMs) and Agentic Workflows

Large Language Models (LLMs) are large neural networks trained on extensive textual corpora to perform natural language processing tasks, such as text comprehension, reasoning, and generation. Recent models like GPT-5 [46] and Llama 4 [4] exhibit strong proficiency in generalizing across diverse tasks and domains without task-specific training. Additionally, reasoning models like DeepSeek-R1 [14] and o3 [47] are LLMs trained with reinforcement learning specifically to perform reasoning and excel in complex problem-solving and tasks that benefit from step-by-step thinking.

Agentic workflows are a series of automated steps or sub-tasks where AI agents, often powered by LLMs, make decisions, take actions, and coordinate tasks to achieve a specific goal. Workflows help decompose complex tasks, enable intermediate reasoning, and improve transparency, making agent behaviour easier to interpret and debug [52]. The simplest form is a *straight-shot* or direct answering workflow, where the LLM receives all inputs upfront and outputs its results in a single pass, optionally including its reasoning traces (e.g., chain-of-thought [74]). In more interactive settings, agents must engage with

an external *environment* to access tools, retrieve data, or perform other actions. These augmented LLMs adopt workflows like REACT (Reason + Act) [81], which interleaves reasoning and action through a *thought-action-observation* loop. Each action is informed by prior observations from the environment, enabling dynamic decision-making based on intermediate feedback. Figure 2.1 illustrates both paradigms.

2.5 Problem Definition

We focus on (1) root cause localization, (2) fault type classification, and (3) propagation path identification based on multi-modal telemetry data. Given a set of failures (i.e., multi-modal alert events) caused by a single fault, the agent’s objective is to jointly perform the three specified tasks.

We model a heterogeneous cloud-based software system as a typed knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T}_{\mathcal{E}}, \mathcal{T}_{\mathcal{R}})$ which encodes explicit domain knowledge and operational dependencies within the system, where \mathcal{E} is the set of system entities composed of entity types $t \in \mathcal{T}_{\mathcal{E}}$ (e.g., services, hosts, data stores) and $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{T}_{\mathcal{R}} \times \mathcal{E}$ is the set of typed edges with types drawn from the relationships in $\mathcal{T}_{\mathcal{R}}$ (e.g., *control-flow*, *hosted-on*, *instance-of*). Each entity type $t \in \mathcal{T}_{\mathcal{E}}$ is associated with a set of fault classes \mathcal{F}_t .

For a given fault s and its associated multi-modal monitoring data (metrics, traces, and logs), we assume a set of alerts \mathcal{A}_s , each mapped to an element in the knowledge graph, as determined by an external alert extraction procedure (Section 3.1.1). Specifically, each alert $a \in \mathcal{A}_s$ is associated with a graph element, i.e., either an entity or relationship, denoted as $c(a) \in \mathcal{E} \cup \mathcal{R}$. Given a set of alerts for fault s , the LLM agent is prompted to produce a ranked list of k fault hypotheses:

$$\mathcal{H}_s[k] = \{h_s^{(i)}\}_{i=1}^k.$$

Each hypothesis $h_s^{(i)}$, is a structured tuple

$$h^{(i)} = (e^{(i)}, f^{(i)}, p^{(i)}, j^{(i)}),$$

where $e^{(i)} \in \mathcal{E}$ is the predicted root-cause entity, $f^{(i)} \in \mathcal{F}_t$ is the predicted fault type for $e^{(i)}$ with t being the type of $e^{(i)}$, $p^{(i)}$ is a path in the knowledge graph, and $j^{(i)}$ is the natural language justification. Each propagation path $p^{(i)}$, defined as

$$p^{(i)} = [r_{e^{(i)}, \xi}, r_{\xi, \xi-1}, \dots, r_{1, c(a)}]$$

must correspond to a valid walk in the graph, where each step $r_{m,n} = (e_m, \tau_{m,n}, e_n)$ corresponds to an edge $r \in \mathcal{R}$ with type $\tau \in \mathcal{T}_{\mathcal{R}}$, and must terminate at an alerted entity or relationship $c(a)$. The agent provides a ranked list of human-interpretable hypotheses, each explaining a likely root cause, its fault type, and its propagation path through the system graph.

Chapter 3

Study Methodology

Figure 3.1 summarizes the end-to-end overview of our study methodology. We first extract modality-specific alerts from raw telemetry data (logs, metrics, and traces)—which are canonicalized into a unified alert format—and a KG that encodes the entities, control- and data-flow dependencies, and deployment configurations of the software system (Section 3.1). These form the input into the RCA inference setup (Section 3.2), where an LLM is run under three distinct workflows (*Straight-Shot*, REACT, *Plan-and-Execute*), producing final root-cause hypotheses (location, fault type, propagation path) and step-level inference traces. Finally, we evaluate both the final outputs and the inference traces (Section 3.3). We expand on each stage in the following sections.

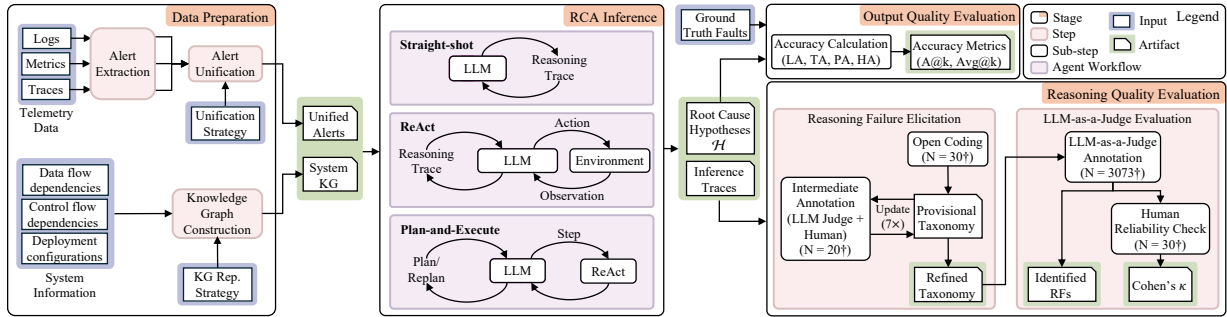


Figure 3.1: Overview of the study method.

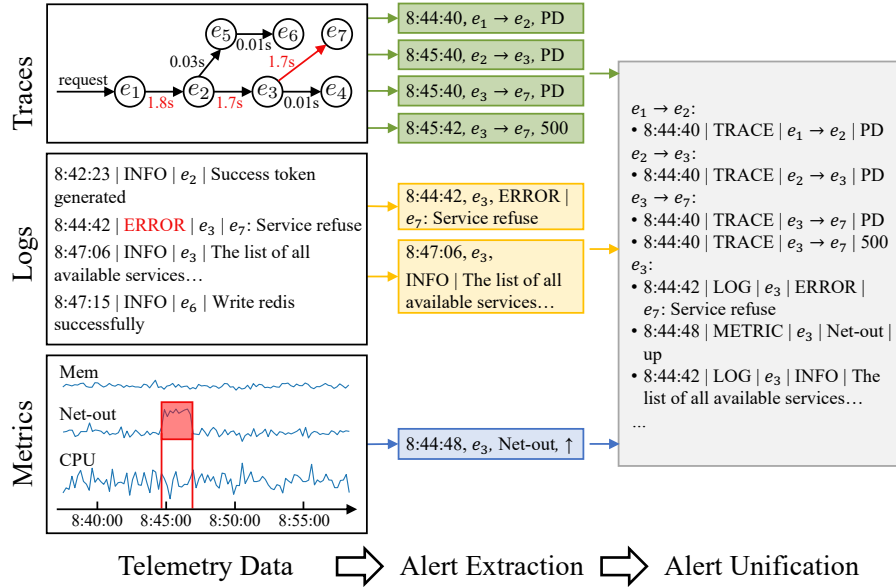


Figure 3.2: The alert extraction and unification process using traces, logs, and metrics.

3.1 Data Preparation

Data preparation consists of alert compilation and KG construction. The former is decomposed into two sequential steps: 1) alert extraction and 2) alert unification. Figure 3.2 shows an example of alert compilation for a single fault.

3.1.1 Alert Extraction

We perform multi-modal alert extraction as a data pre-processing step. This step is designed to emulate realistic production settings, where monitor alerts or event triggers serve as the primary entry point to failure triage and diagnosis [19]. Since most open-source RCA monitoring datasets consist of large quantities of raw telemetry without clear natural-language alert labels, we extract and process the alert events suitable for LLM use. Specifically, we adopt a *feature fusion* approach [89], where we process multi-modal monitoring data and extract a unified representation of the fault, which serves as input to the LLM agent. Our approach follows prior work on fusion-based approaches [88, 86, 77], applying anomaly detection techniques independently to each modality (metrics, traces, and logs) to generate alerts.

Log Alerts

We use a log parsing approach Drain [26] to extract static log templates and dynamic log parameters from raw log messages, similar to [86, 88, 77]. It utilizes a fixed-depth parse tree to group similar log messages and extract common patterns. To ensure that we select valuable logs for failure diagnosis and filter out unnecessary noise, we use a two-part log alert sampling technique. First, we preserve occurrences of ERROR-level and low-frequency log templates as candidate alerts, following the approach in [77]. Although these templates are globally rare, they can manifest as high-volume bursts during a fault (e.g., retry loops or cascading errors). Second, to ensure that the alerts fit in the LLM context window, we perform representative sampling for high-volume templates within a fault window. For each system entity, we retain only the first occurrence of a frequent template. The extracted log alerts contain a *timestamp*, *system entity*, and *message*.

Trace Alerts

We use the Isolation Forest (IForest) [38] to detect anomalies in traces between services and service instances. IForest is an unsupervised anomaly detection method that identifies points that are intrinsically easy to separate from the rest of the data. It constructs an ensemble of binary trees by repeatedly selecting a random feature and a random split value, then partitioning the dataset. Each data point follows a path down a tree and the algorithm records the number of splits (i.e., the path length) required to isolate it. Intuitively, an anomalous point will be isolated after a few partitions, whereas a typical point blends with many others and needs more splits to be separated. Because the method averages path lengths across many randomly built trees, it is robust to noisy features and scales well. Shorter average path length thus signals that an invocation pair lies in a sparse or extreme region of the trace feature space and is therefore more likely to be anomalous.

We apply IForest to the response time and status codes of each invocation pair. High response time indicates potential performance degradation (PD), while abnormal status codes suggest errors (ERROR). The extracted alerts consist of a *timestamp*, *callee*, *caller*, *operation name*, and *abnormality type* (i.e., PD, 400-ERROR, or 500-ERROR).

Metric Alerts

We opt for the 3-sigma rule [51] to detect anomalous metrics. For a given metric, we collect the numerical fluctuations over a time period and compute the mean μ and standard

deviation σ . When the value of a metric exceeds the upper bound of 3-sigma ($\mu + 3\sigma$), it is deemed an alert with an *up* anomaly direction. Similarly, the anomaly direction is *down* for an alert if the metric value falls below the lower bound ($\mu - 3\sigma$). The extracted metric alerts therefore consist of a *timestamp*, *system entity*, *metric name*, and *anomaly direction*.

3.1.2 Alert Unification

To obtain a consistent input representation for the LLM, we group alerts according to a *unification strategy*. For input robustness, we consider two strategies: (1) *time-based unification*, which merges alerts into a single chronological sequence to emphasize temporal ordering and potential cause-effect relations between events; and (2) *element-based unification* (shown in Figure 3.2), which aggregates alerts by their reporting element (i.e., the entity or relationship in the KG), making it easier to attribute symptoms to specific system entities or interactions.

3.1.3 System Knowledge Graph (KG) Construction

Most existing research relies on incident metadata (e.g., title and summary) for fine-tuning LLMs [1] or directly querying LLMs with in-context examples [96, 19, 58, 9]. However, failures stemming from faults in dependent components require additional information for holistic reasoning. Both [19] and [92] show that upstream dependency information can assist LLMs in better reasoning and improve the quality of recommendations. To this end, we model a software system as an explicit, typed KG that captures its structural and operational aspects. KGs’ semantic richness is well-suited for integration with LLMs for relational reasoning and fault-propagation analysis [76].

Concretely, we represent the system as the typed knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T}_{\mathcal{E}}, \mathcal{T}_{\mathcal{R}})$ introduced in Section 2.5. Nodes $e \in \mathcal{E}$ (with types $t \in \mathcal{T}_{\mathcal{E}}$) represent system entities consisting of both software (e.g., *Service*, *Service-Instance*, *Database*) and hardware components (e.g., *Host*). Typed edges $r \in \mathcal{R}$ (with types in $\mathcal{T}_{\mathcal{R}}$) capture dependencies and interactions commonly observed in cloud-native architectures, such as control and data flow dependencies, instantiations of software components to capture redundancy and scaling (e.g., *instance-of*), and deployment relations (e.g., *hosted-on*). This formalization allows us to (i) associate each entity type t with its fault class set \mathcal{F}_t ¹ and (ii) ground alerts $a \in \mathcal{A}_s$ to graph elements via the mapping $c(a) \in \mathcal{E} \cup \mathcal{R}$.

¹Some entity types in some datasets have an empty \mathcal{F}_t . For example, Dataset \mathcal{A} contains faults only for *Service-Instance*-type entities, while the system is comprised of additional types of entities.

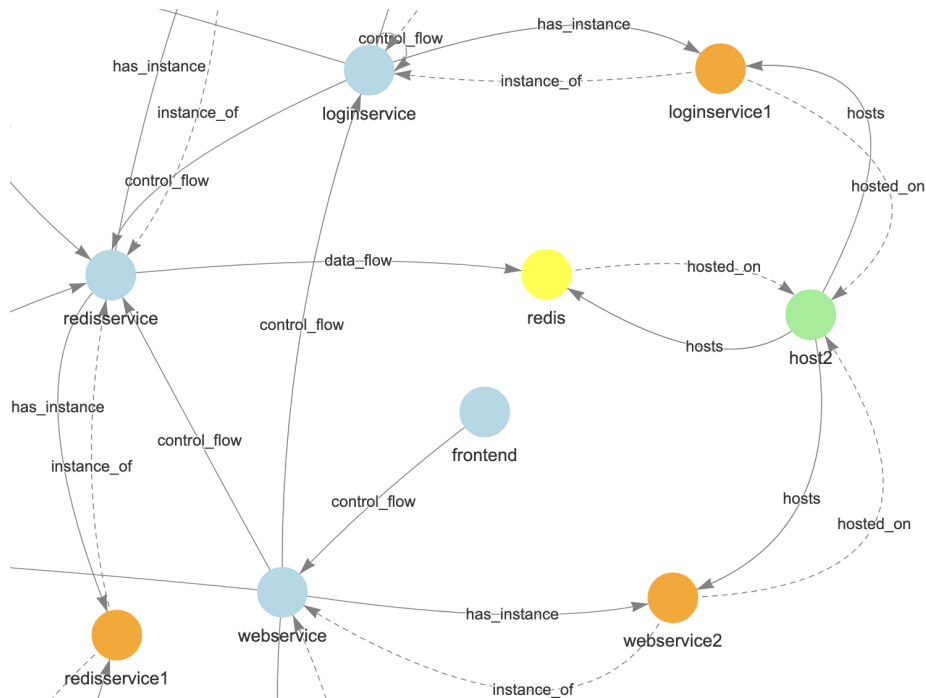


Figure 3.3: A region of the system KG for MicroSS [11]. Blue, orange, green, and yellow nodes represent *Service*, *Service Instance*, *Host*, and *Cache* type entities, respectively. Only *instance-of*, *has-instance*, *hosted-on*, *hosts*, *control-flow*, and *data-flow* relationships are shown for simplicity.

To build the KG, we manually parse architectural documentation, deployment manifests, and configuration files, and normalize the results into a typed entity-and-relationship schema. During normalization, we ensure extracted nodes and edges are assigned types in $\mathcal{T}_{\mathcal{E}}$ and $\mathcal{T}_{\mathcal{R}}$, respectively. Figure 3.3 provides an example of a region of the KG constructed for MicroSS. In this case, entity types include *Service*, *Service-Instance*, *Orchestration-Manager*, *Host*, *Database* and *Cache*. For simplicity, we show a subset of relationships.

We represent the KG according to a *KG representation strategy*. Similar to the alert unification strategy, the purpose of this is to increase the robustness of the input prompts to model-specific biases. We use two strategies: (1) *list representation*, where nodes and edges are represented as bullet-form lists, and (2) *JSON representation*, where nodes and edges are provided as a JSON object.

3.2 RCA Inference

We evaluate LLM-based agents responsible for diagnosing multi-hop failures under three paradigms: reactive reasoning with tool-based feedback (**REACT**), plan-then-execute reasoning (*Plan-and-Execute*), and a non-agentic baseline. These paradigms differ primarily in how the agent organizes decision-making and interacts with an external environment. In this section, we further describe the (1) agent workflows, (2) available tools, and (3) input prompts.

3.2.1 Workflows

All workflows involve a single LLM-based agent operating autonomously.

ReAct. The agent follows a classical REACT (Reasoning + Acting) [81] pattern, interleaving tool-based observations with incremental reasoning. Concretely, the agent implements a *thought-action-observation* loop: it reasons, issues a tool call (action), receives an observation, and uses that observation to inform the next reasoning step. This fine-grained, reactive workflow closely mirrors real-world RCA, i.e., sequential troubleshooting decisions and knowledge-intensive analysis of newly acquired information. The interleaved structure allows the agent to quickly adapt to new information without committing to a fixed long-horizon plan upfront.

Plan-and-Execute. The *Plan-and-Execute* workflow follows the Plan-and-Solve [69] prompting paradigm, where planning and execution are explicitly separated to encourage more deliberate exploration, clearer intermediate outputs, and reduced redundant reasoning loops. The agent first generates a high-level investigative plan from the initial scenario context, then executes each step sequentially using available tools. Based on the results of each step, the plan is adjusted accordingly by the LLM. Effective up-front planning for RCA remains a challenge for LLMs [48], motivating our evaluation of this separation between planning and executing.

Straight-Shot (Non-agentic Baseline). The non-agentic baseline implements a straight-shot workflow: the LLM receives all alerts and system knowledge upfront and returns a diagnosis in a single pass, without intermediate tool use. This straight-shot setup mirrors direct-answering workflows where the model must reason in one pass, optionally exposing

Table 3.1: Description of Tools in Agentic Workflows

Category	Tool	Description
Data Characteristics	check node existence	Check if a named entity exists in the system.
	get node attributes	Retrieve attributes and alert data of a given entity.
	get all instances of entity type	Enumerate all instances of a specified entity type.
	get edge attributes	Inspect properties of edges between two entities.
Graph Traversal	get node neighborhood	Retrieve the r-hop neighborhood of a given entity.
	get all simple paths	Enumerate all simple paths between two entities.

chain-of-thought [74] or “think” traces. In practice, this approach would require careful context management to fit the relevant information into the context window.

3.2.2 Agent Tools

We design the tools provided to the LLM agent to be semantically minimalist (i.e., each tool has narrow, clearly defined semantics) and deterministic. We refrain from allowing direct access to the KG or code-writing capabilities for exploration or analysis, inspired by observations from [73, 79]. Although this reduces the action space, it ensures a more reliable and interpretable evaluation of reasoning behaviour. Specifically, it confines agent decisions to a well-defined, reproducible set of tool calls, making it possible to attribute performance outcomes to reasoning quality rather than tool ambiguity, stochasticity, or code execution artifacts. To this end, the tools serve to expose entity relationships and the system topology. They are organized into two categories, *Data Characteristics* and *Graph Traversal*, and summarized in Table 3.1.

Data Characteristics Tools allow the agent to validate the presence of specific entities, inspect their attributes and alerts, list all instances of a given type, and examine attributes of a given relationship. These tools enable structured inspection of relevant entities and alert data.

Graph Traversal Tools allow the agent to explore the structural topology of the system. The agent can retrieve the r -hop neighbourhood of a given entity, or query for all simple paths between two entities. These operations support inference over causal chains and fault propagation paths.

Each tool output provides information about the KG formatted according to the KG representation strategy. For every tool call, we require the LLM to provide a justification for its invocation. We do this by adding an additional required parameter “reasoning” to each

tool definition. This provides us with some insight into the reasoning behind the specified actions, particularly useful for non-reasoning or -thinking LLMs. These justifications are included in the inference trace and provided as input to the LLM-as-a-Judge for evaluation (see Section 3.3.2).

3.2.3 Input Prompts

Each agent workflow is guided by an initial prompt that defines the RCA objective (generating a ranked list of k fault hypotheses), KG schema (entity and relationship types), and available tools. It also contains the unified alert representation (Section 3.1.2) for a particular fault scenario. We provide a condensed example of the REACT workflow in the Appendix (Figure A.4) to illustrate its general structure. While the core components of the prompts—task objectives, schema, output requirements, alert data—remain consistent across workflows, each prompt is tailored to the respective decision-making strategy (e.g., incremental reasoning in REACT, plan specification in *Plan-and-Execute*). Additionally, we provide the full KG in text format (according to a KG representation strategy) for the *Straight-Shot* approach. Full prompt templates for all workflows are provided in our online repository [56].

3.3 Evaluation

To assess LLM capabilities, we assess both the final root-cause hypotheses and the inference traces produced during the RCA inference stage. We define an *inference trace* to consist of all reasoning traces (i.e., intermediate LLM outputs, content of think tags), actions (i.e., tool calls and their justification), observations (i.e., tool outputs), and plan steps (for *Plan-and-Execute* case). This forms a linear history of thought (and action) that led to the final root-cause hypotheses produced by the LLM agent.

3.3.1 Output Quality

To answer RQ_1 and RQ_2 , we assess the alignment of each root-cause hypothesis with the ground-truth to evaluate the ability of the LLM to *correctly* identify the root cause. As such, we measure the root-cause Location Accuracy (LA), Type Accuracy (TA), and Hypothesis Accuracy (HA), where HA requires both the fault location and type to match the ground truth. Additionally, we measure Propagation Path Validity (PA), which checks

whether the predicted propagation path is a valid walk in the KG. For each accuracy measure, we employ the top- k accuracy (A@ k), which checks the presence of the ground truth in the top- k hypotheses. Formally, let \mathcal{S} be the set of system fault scenarios, $s \in \mathcal{S}$ denote one fault case, V_s denote the ground-truth value of s , and $H_s[k] = \{h_s^{(i)}\}_{i=1}^k$ denote the top- k predicted hypotheses for s . Then,

$$A@k = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \begin{cases} 1, & \text{if } V_s \in \mathcal{H}_s[k] \\ 0, & \text{otherwise} \end{cases} \quad (3.1)$$

We report results for $k = 1, 3$. We additionally measure the Average Accuracy@ K (Avg@ K) for $k = 3$, defined as

$$\text{Avg}@K = \frac{1}{K} \sum_{1 \leq k \leq K} A@k. \quad (3.2)$$

3.3.2 Reasoning Quality

Evaluating the quality of an LLM’s reasoning requires methods that can capture both the depth and nuance of its decision process. In our setting, RCA inference traces are long, unstructured, and interleaved with tool interactions, making them unsuitable for structured parsing approaches (e.g., first-order logic, causal graphs) and conventional reasoning metrics (e.g., ROSCOE [20], ReCEval [50] BERTScore [91]). Instead, we adopt a rationale-based evaluation scheme [44] that combines qualitative analysis with structured LLM-based judging (i.e., *LLM-as-a-Judge* [99]). This hybrid approach offers both interpretability and scalability: qualitative inspection grounds the evaluation in observed reasoning patterns, while LLM-based judging enables greater systematic coverage. Figure 3.1 summarizes the procedure consisting of two steps: *Reasoning Failure Elicitation* (Section 3.3.2) and *LLM-as-a-Judge Evaluation* (Section 3.3.2). Nodes labelled with N^\dagger denote independent random samples of size N .

Reasoning Failure Elicitation

We developed a taxonomy of *reasoning failures* (RFs) through a structured, mixed-methods elicitation procedure. Starting from a random sample of 30 RCA outputs (drawn from diverse agent and input configurations to maximize behavioural coverage), we performed *Open Coding* [13] to identify recurrent reasoning issues (e.g., related to evidence handling, temporal inference, provenance, causal attribution, tool misuse). We subsequently drafted

Table 3.2: Reasoning Failure Taxonomy

ID	Name	Description	Scope	Category
RF-01	Fabricated evidence	Asserts existence of alerts, metrics, logs, or traces not found in the provided evidence.	Per-hypothesis	General
RF-02	Metric interpretation error	Misreads metric semantics (e.g., inverts directionality or confuses counters and gauges).	Per-hypothesis	RCA-specific
RF-03	Confused provenance	Attributes causation to the component observing a symptom rather than its true source.	Per-hypothesis	RCA-specific
RF-04	Temporal misordering	Infers causal direction that violates chronological order of observed events.	Per-hypothesis	General
RF-05	Spurious causal attribution	Claims causal relationships unsupported by alerts or knowledge graph structure.	Per-hypothesis	General
RF-06	Unjustified instance specificity	Asserts instance-level fault without discriminating instance-specific evidence.	Per-hypothesis	RCA-specific
RF-07	Arbitrary evidence selection	Chooses evidence subsets inconsistent with systematic triage heuristics.	Per-hypothesis	RCA-specific
RF-08	Evidential insufficiency	Relies on weak or non-specific evidence insufficient to support the diagnostic claim.	Per-hypothesis	General
RF-09	Failure to update belief	Does not revise or retract claims contradicted by later evidence or tool outputs.	Full trace	Procedural
RF-10	Simulation or role confusion	Treats simulated or assumed tool outputs as factual evidence.	Full trace	Procedural
RF-11	Excessive speculation	Engages in prolonged speculative or circular reasoning that obstructs analysis.	Full trace	Procedural
RF-12	Repetition or failure to resume	Repeats planning or reasoning across turns without substantive progress.	Full trace	Procedural
RF-13	Anchoring bias	Fixates prematurely on one hypothesis and neglects exploration of alternatives.	Cross-cutting	General
RF-14	Invalid inference pattern	Applies formal or informal logical fallacies in diagnostic reasoning.	Cross-cutting	General
RF-15	Internal contradiction	Produces mutually inconsistent statements within the inference history.	Cross-cutting	General
RF-16	Arithmetic or aggregation error	Performs numeric miscalculations or aggregations affecting interpretation.	Cross-cutting	Procedural

definitions and exemplars for each provisional code. These initial codes formed the *Provisional Taxonomy* used in subsequent rounds. Taxonomy refinement proceeded through seven iterative rounds. In each round, we sampled a new random batch of 20 RCA outputs. Each sample underwent *Intermediate Annotation* in parallel by an LLM-based judge and a human annotator according to the working taxonomy. In each round, we collected novel failure cases, updated failure definitions and examples, and refined the set of evaluation steps provided to the judge. This paired-annotation approach (LLM judge + human) was used to align interpretations and, in turn, update the taxonomy and input prompt. Only minor revisions were required in the last round, thereby forming the *Refined Taxonomy* (Table 3.2), and we used this taxonomy, evaluation steps, and prompt for the final *LLM-as-a-Judge Annotation*. The full prompt is provided in Section A.4.

We organize the failures into three interpretable buckets. *Procedural* failures arise in the

agent’s decision-to-act loop (planning, tool invocation/interpretation, multi-turn control), causing procedural breakdowns in how the LLM functions as a reasoning agent. *RCA-specific* failures are domain-related errors tied to diagnostic knowledge (metric semantics, provenance, instance discrimination, triage heuristics). *General* reasoning failures are defects that undermine the credibility of hypotheses or whole traces (unsupported causal links, temporal ordering errors, anchoring, logical fallacies, internal contradictions).

LLM-as-a-Judge Evaluation

LLM-as-a-Judge merges the scalability of automatic methods with the detailed, context-sensitive reasoning found in expert judgments [21]. We employ GPT-5 [46], a state-of-the-art reasoning model, as an independent evaluator to annotate reasoning failure across RCA traces. To mitigate self-enhancement bias—a phenomenon that LLM evaluators may prefer responses generated by themselves [99, 83]—the judge model differs from those under evaluation. Following best practices to ensure reliable and accurate evaluations, it is guided by prompts that incorporate: (1) the reasoning failure taxonomy with definitions and exemplars, (2) decomposed evaluation steps, and (3) a fixed output schema requiring categorical judgments and textual explanations with evidence from the RCA inference trace. We apply this judge to a stratified subset of 3,073 (of 19,200; 16%) RCA inference traces (~ 100 per dataset-model-workflow combination) to quantify failure prevalence.

To provide a *Reliability Check* of the judge outputs, the author and a collaborator independently reviewed a random sample of 30 judge-annotated traces drawn from the judged set. Each reviewer applied the taxonomy and we report Cohen’s $\kappa = 0.92$ (95% CI) as a reliability metric. These human annotations were used only to assess judge reliability and were not used to alter final judge labels or for downstream analysis.

Chapter 4

Experimental Setup

4.1 Experimental Data

We conducted our experiments on two open-source microservice datasets (\mathcal{A} , \mathcal{B}) with multi-modal monitoring data (i.e., logs, metrics, and traces), listed in Table 4.1. Dataset \mathcal{A} is derived from the Generic AIOps Atlas (GAIA) dataset [11], which is collected on the MicroSS microservice application. Dataset \mathcal{B} is derived from the *Market* dataset of OpenRCA [79], which is collected from an open-source cloud-native application Online Boutique [49].

MicroSS contains 19 software entities: a frontend service, five services containing core business logic with two service instances each, a coordination manager (Zookeeper), and two data stores (MySQL and Redis). These are supported by five host machines, totalling 24 entities. The GAIA dataset encompasses five distinct faults: system stalls, process crashes, login issues, missing files, and access denials. These are injected into *service instances*, of which there are 10. Similarly, the Online Boutique contains 10 services, each with four instances, and a data store (Redis) with two instances, all distributed across six hosts. The *Market* dataset contains 15 fault categories across three fault levels: host, service, and service instance (e.g., service network latency, host CPU spike), resulting in 56 possible fault locations. These systems have been widely used in many previous RCA studies [64, 86, 32, 88, 98, 25, 24, 72, 77, 79, 48] and have become important experimental platforms for researchers to study microservice architectures and performance.

Dataset Balancing. The two datasets differed substantially in the number of injected faults and data volumes (differences up to 100-fold), which can bias the evaluation. To

Table 4.1: Experimental Data Details

Dataset	System	Samples	Entities	Fault Locations	Fault Categories	Fault Granularity
\mathcal{A}	MicroSS	152	24	10	5	Service instance
\mathcal{B}	Online Boutique	148	60	56	15	Host, service, service instance

address this, we downsampled the larger dataset (\mathcal{A}) to satisfy two goals: (1) match the overall number of fault records to \mathcal{B} , and (2) reduce extreme class imbalance so no single fault class dominates. We first removed records with temporally overlapping faults (23.7%) and removed records with missing telemetry, yielding 4,343 (26.8%) records. We then downsampled \mathcal{A} to the same order of magnitude as \mathcal{B} and applied class-aware undersampling that caps overly frequent classes (\mathcal{A} originally contained 16,205 faults, 95.5% of which were Session Timeout) to improve per-class balance. The final counts are given in Table 4.1. No additional balancing was required for \mathcal{B} , since the dataset was already balanced by the authors of [79]. The final per-fault-type distributions for \mathcal{A} and \mathcal{B} are summarized in Tables A.1 and A.2.

4.2 LLMs

To support the RCA task requiring processing long contexts and action formulation, we selected a set of open-source instruction-tuned models with at least 128K token context windows and support for tool calling and structured output from a variety of model families and parameters, including Llama 3.2 (3B) [2], Qwen 3 (4B and 32B) [80], Llama 3.3 Instruct (70B) [3], and Command R+ (104B) [12]. We additionally select a distilled DeepSeek-R1 (70B) [14] reasoning model. However, we evaluate Deepseek-R1 only on the *Straight-Shot* workflow as it does not support tool calling. The model checkpoints are included in our online repository [56]. We restrict our selection to open-source models as these are often readily deployable in-house and reduce privacy and security concerns [73].

4.3 Implementation and Settings

We implement all agent workflows using LangGraph¹ with a graph recursion limit of 50. As structured generation (i.e., the process of producing content in standardized formats like JSON) negatively impacts LLMs’ abilities [65], including for reasoning and domain comprehension, we allow agents to generate unstructured final answers and apply a post-hoc LLM call to convert them into a structured format (JSON) for systematic evaluation.

Additionally, following prior work [60, 73], we use pre-defined criteria to mark problematic actions or states, as LLM tool invocation is prone to error propagation [53]. These include: (i) empty responses, (ii) invocation of non-existent tools, (iii) tool calls with invalid or missing parameters, (iv) trivial or repeated plans in *Plan-and-Execute*, and (v) premature termination without providing a final answer. When such conditions are met, we provide the agent with error messages and suggestions to ensure continued and focused execution. All experiments were run locally on a Linux machine with Python 3.12, across ten 48 GB NVIDIA RTX A6000 GPUs and eight 32 GB NVIDIA Tesla V100 GPUs. We executed 48,000 fault scenarios, totalling 228 days of execution time.

¹<https://langchain-ai.github.io/langgraph/>

Chapter 5

Results

In this chapter, we present the results for each RQ: RQ_1 (Section 5.1), RQ_2 (Section 5.2), RQ_3 (Section 5.3), and RQ_4 (Section 5.4).

5.1 RQ_1 : How effectively can an LLM agent perform RCA tasks?

We report the accuracy of the root-cause locations (LA), types (TA), propagation paths (PA), and overall hypotheses (HA) in Table 5.1 and Figure 5.1. We include a random-guessing baseline for each dataset (computed using the number of fault locations and fault type categories for LA and TA, respectively). Dataset \mathcal{B} represents a substantially harder localization and type-identification problem than \mathcal{A} : it contains over five times as many fault locations (56 vs. 10) and 3 times as many fault categories (15 vs. 5), resulting in a much lower random-guessing baseline (see Table 4.1). As a result, accuracies are consistently lower on \mathcal{B} , even when relative trends across models and workflows remain comparable.

Observation 1: No one-size-fits-all—model strengths are task-specific. Table 5.1 shows Qwen 3 (32B) consistently attains the highest HA across all workflows and datasets, with HA@3 up to 0.36 on \mathcal{A} (*Straight-Shot*) and 0.09 on \mathcal{B} (REACT). Qwen 3 (32B) and Llama 3.3 exhibit similar LA and TA performance (2.6 ± 4.9 difference). However, Qwen 3 (32B) leads on *Plan-and-Execute* in \mathcal{B} , while Llama 3.3 achieves markedly higher PA across all workflows (evident in Figure 5.1). Surprisingly, Qwen 3 (4B) performs competitively with Qwen 3 (32B) for *Straight-Shot* and REACT, averaging only 1.2 ± 3.5

Table 5.1: Accuracy Results

Dataset	Workflow	Model	LA@1	LA@3	LA-Avg@3	TA@1	TA@3	TA-Avg@3	PA@1	PA@3	PA-Avg@3	HA@1	HA@3	HA-Avg@3
<i>A</i>	<i>Straight-Shot</i> (Baseline)	Llama 3.2	0.13	0.31	0.23	0.24	0.64	0.44	0.03	0.39	0.25	0.05	0.10	0.07
		Qwen 3 (4B)	0.31	0.49	0.40	<u>0.42</u>	0.60	0.52	0.13	0.24	0.19	0.27	0.34	0.31
		Qwen 3 (32B)	0.36	0.52	0.45	0.42	0.51	0.47	0.15	0.29	0.23	0.31	0.36	0.34
		Llama 3.3	0.35	0.46	0.41	0.40	0.68	0.54	0.45	0.61	0.54	0.25	0.29	0.28
		DeepSeek-R1	0.34	0.49	0.42	0.42	0.65	0.54	0.43	0.67	0.57	0.26	0.33	0.30
		Command R+	0.29	0.46	0.38	0.38	0.64	0.51	0.32	0.44	0.39	0.21	0.27	0.25
	REACT	Llama 3.2	0.11	0.23	0.17	0.17	0.38	0.27	0.15	0.36	0.27	0.04	0.09	0.06
		Qwen 3 (4B)	0.33	0.43	0.38	0.40	0.66	0.53	0.01	0.02	0.02	0.27	0.29	0.28
		Qwen 3 (32B)	0.35	0.47	0.42	0.41	0.63	0.52	0.11	0.22	0.17	0.29	0.35	0.33
		Llama 3.3	0.36	0.48	0.42	0.42	0.73	0.56	0.62	0.73	0.68	0.27	0.31	0.29
		Command R+	0.30	0.44	0.37	0.35	0.50	0.42	0.17	0.23	0.20	0.19	0.26	0.23
	<i>Plan-and-Execute</i>	Llama 3.2	0.02	0.05	0.04	0.03	0.11	0.07	0.04	0.12	0.09	0.01	0.01	0.01
		Qwen 3 (4B)	0.04	0.07	0.05	0.06	0.11	0.08	0.01	0.02	0.02	0.03	0.03	0.03
		Qwen 3 (32B)	0.29	0.46	0.38	0.38	0.58	0.48	0.11	0.37	0.27	0.24	0.3	0.27
		Llama 3.3	0.31	0.46	0.39	0.37	0.60	0.49	0.72	0.78	0.75	0.23	0.29	0.26
		Command R+	0.23	0.36	0.30	0.29	0.52	0.41	0.18	0.27	0.23	0.14	0.20	0.17
	Random Guessing		0.10	0.30	0.20	0.20	0.60	0.40	-	-	-	0.02	0.06	0.04
	<i>B</i>	<i>Straight-Shot</i> (Baseline)	Llama 3.2	0.03	0.10	0.06	0.06	0.17	0.12	0.01	0.04	0.02	0.0	0.007
Qwen 3 (4B)			0.07	0.19	0.13	0.07	0.19	0.13	0.14	0.37	0.28	0.015	0.041	0.029
Qwen 3 (32B)			0.07	0.22	0.14	0.08	0.24	0.16	0.18	0.34	0.28	0.019	0.074	0.044
Llama 3.3			0.03	0.16	0.10	0.09	0.24	0.16	0.10	0.45	0.30	0.005	0.020	0.013
DeepSeek-R1			0.03	0.16	0.10	0.06	0.19	0.12	0.19	0.59	0.41	0.007	0.025	0.016
Command R+			0.03	0.08	0.06	0.07	0.19	0.13	0.28	0.47	0.38	0.005	0.019	0.012
REACT		Llama 3.2	0.02	0.07	0.05	0.03	0.07	0.05	0.21	0.25	0.23	0.0	0.003	0.002
		Qwen 3 (4B)	0.11	0.33	0.23	0.08	0.22	0.15	0.01	0.08	0.05	0.024	0.083	0.059
		Qwen 3 (32B)	0.10	0.24	0.17	0.10	0.25	0.18	0.20	0.30	0.26	0.035	0.088	0.062
		Llama 3.3	0.04	0.14	0.10	0.05	0.17	0.11	0.54	0.73	0.65	0.003	0.017	0.010
		Command R+	0.06	0.10	0.08	0.06	0.12	0.09	0.26	0.32	0.29	0.008	0.015	0.012
<i>Plan-and-Execute</i>		Llama 3.2	0.01	0.03	0.02	0.01	0.03	0.02	0.03	0.04	0.03	0.002	0.003	0.003
		Qwen 3 (4B)	0.01	0.03	0.02	0.01	0.02	0.01	0.0	0.01	0.01	0.0	0.003	0.002
		Qwen 3 (32B)	0.09	0.24	0.17	0.08	0.21	0.15	0.23	0.35	0.30	0.024	0.061	0.042
		Llama 3.3	0.02	0.11	0.06	0.05	0.13	0.09	0.54	0.59	0.57	0.002	0.014	0.007
		Command R+	0.03	0.07	0.06	0.07	0.13	0.10	0.22	0.27	0.24	0.008	0.014	0.011
Random Guessing			0.02	0.05	0.04	0.07	0.20	0.13	-	-	-	0.001	0.004	0.002

Per metric column: the top value within each dataset–workflow is shown in **bold**; the top value across workflows (per dataset) is underlined; values below the random-guessing baseline are shown in color.

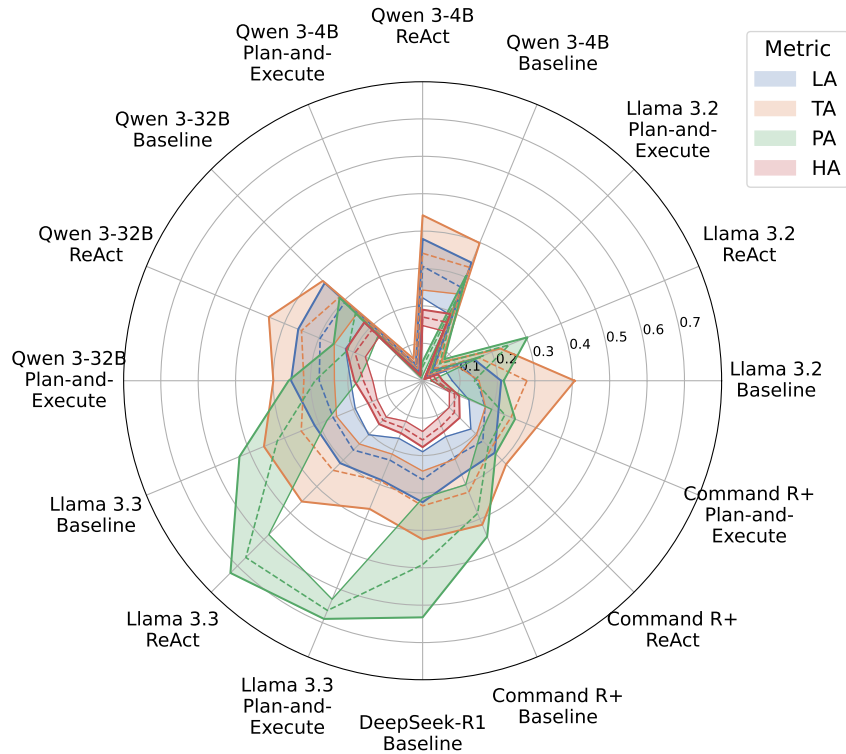


Figure 5.1: Accuracy results aggregated across \mathcal{A} and \mathcal{B} (lower bound is A@1, upper bound is A@3, and dotted line is Avg@3).

accuracy points lower (max 8.8), excluding PA for REACT which does not follow this trend. Taken together, these results show a task-specific specialization: Qwen 3 (32B) is the best single-model choice for overall hypothesis synthesis—its 4B counterpart is a suitable choice for smaller systems or less complex agentic workflows—while Llama 3.3 is preferable when path reconstruction and grounding in the system context is the objective. No single model dominates across all RCA subtasks. Model selection should therefore reflect the target RCA objective (e.g., ranking hypotheses, path reconstruction) rather than model size alone.

We note a pronounced gap between partial (LA, TA) and full-hypothesis accuracy (HA) on \mathcal{B} . Relatively high LA and TA but low HA indicate that models often identify plausible locations *or* fault types, but fail to align them correctly within the same hypothesis. In \mathcal{B} , the larger hypothesis space amplifies this effect, showing that partial correctness does not reliably compose into a correct hypothesis.

Observation 2: Agentic workflows show diminishing or negative returns on LA, TA, and HA, and increase below random-guessing rates in specific cases. Across many configurations, *Straight-Shot* often equals or outperforms REACT and *Plan-and-Execute* for these metrics. For the strongest models (Qwen 3 (32B), Llama 3.3), differences between their respective REACT and *Plan-and-Execute* results are typically small (within a few percentage points), indicating limited marginal gains from the added agentic structure. By contrast, smaller models degrade substantially with increased workflow complexity, e.g., Llama 3.2 LA@3 falls from 0.31 (*Straight-Shot*) to 0.23 (REACT) and to 0.05 (*Plan-and-Execute*). Crucially, rates of below random-guessing performance rise with workflow complexity: TA is below its random baseline for many model/workflow pairs, while LA and HA fall below random primarily for smaller models (Llama 3.2, Qwen 3 (4B)) as workflows become more agentic. Agentic workflows increase the cognitive and procedural demand on the model. When model capacity is insufficient, added structure does not guide reasoning; rather, it amplifies compounding missteps, producing systematic errors.

Observation 3: Intentional exploration via agentic workflows improves PA. For sufficiently capable models, intentional exploration via agentic workflows improves propagation path validity: e.g., Llama 3.3 shows PA@3 increases from 0.61 to 0.73 (*Straight-Shot* to REACT) and up to 0.78 (in *Plan-and-Execute*) on \mathcal{A} (Figure 5.1). Agentic workflows can better ground their responses in the system knowledge (i.e., KG) and trace plausible propagation paths. However, we note that this benefit is conditional on the model’s ability to reliably plan and execute tool-based exploration.

Observation 4: Execution errors and limited tool coverage severely constrain end-to-end utility for small models. Practical gains are curtailed by execution errors (graph recursion limits, replanning errors) concentrated in small models: Qwen 3 (4B) and Llama 3.2 suffer high failure rates under *Plan-and-Execute* (87.3% and 77.3%, respectively). Qwen 3 (4B) also shows poor PA under REACT and *Plan-and-Execute*: it explored the KG with tools in only 23.4% of REACT samples (19.8% of non-error samples) and 49.5% of *Plan-and-Execute* samples (67.8% of non-error samples), versus $\sim 94.5\%$ tool coverage on average for other models. Omitted tool calls force the model to guess or hallucinate paths. Llama 3.2 is similarly impacted under *Plan-and-Execute* (though less so under REACT). Execution errors reflect a capacity mismatch—the iterative planning depth and attention to previous steps required by *Plan-and-Execute* exceed what these smaller models can sustain, producing systematic execution breakdowns rather than benign uncertainty. This means end-to-end performance and real-world utility remain severely limited in agentic workflows.

Answer to RQ_1 : We find that the RCA task still proves to be challenging for competitive open-source LLMs, with sub-guessing performance arising as workflow complexity scales, particularly for smaller models. Although agentic workflows can improve PA, suggesting better groundedness in the system context, it is highly model-specific whether LLMs can adequately exploit agentic workflows. We observe that the utility of smaller models in more complex agent workflows like Plan-and-Execute is critically constrained by high recursion limit failures, which reflect a capacity mismatch: the multi-step planning these workflows demand exceeds what smaller models can sustain.

5.2 RQ_2 : How sensitive are final RCA outcomes to alert modalities (logs, metrics, traces)?

To answer RQ_2 , we conduct holdout experiments wherein we withhold each data modality during the alert unification procedure, thereby separately excluding traces, logs, and metrics during RCA inference. Figure 5.2 shows the change in accuracy (ΔA -Avg@3) for each withheld modality aggregated across both datasets. We apply the Wilcoxon signed-rank test [75] for statistical significance on the per-sample correctness scores.¹ We observe that different modalities affect RCA performance in distinct ways.

Observation 5: Withholding metrics strongly degrades localization. Across nearly all models and workflows, excluding metrics produces the largest and most consistent drop in localization (ΔLA typically between -0.07 and -0.15). For example, Qwen 3 (4B) shows $\Delta LA = -0.155$. This indicates metric anomalies are the modality most effectively leveraged by the LLM reasoners for identifying root-cause locations.

Observation 6: Withholding logs primarily harms fault-type attribution. Excluding logs produces the largest degradation in TA (ΔTA down to -0.13). E.g., Qwen 3 (4B) under *Plan-and-Execute* has $\Delta TA = -0.128$. This suggests that textual alerts contain semantic cues that models use to infer error type.

Observation 7: Traces often distract; removing them improves accuracy. Surprisingly, excluding traces typically increases PA (up to $+0.28$) and HA (up to $+0.06$). Qualitatively, we found that LLMs often focus closely on call relationships from trace alerts and ignore other possible entity relationships (e.g., deployment). Additionally, trace

¹The per-sample score is the mean of A@1, A@2, and A@3. It assigns a value of 1.00, 0.67, 0.33, or 0.00 depending on whether the correct root-cause value was found at rank 1, 2, 3, or not found, respectively.

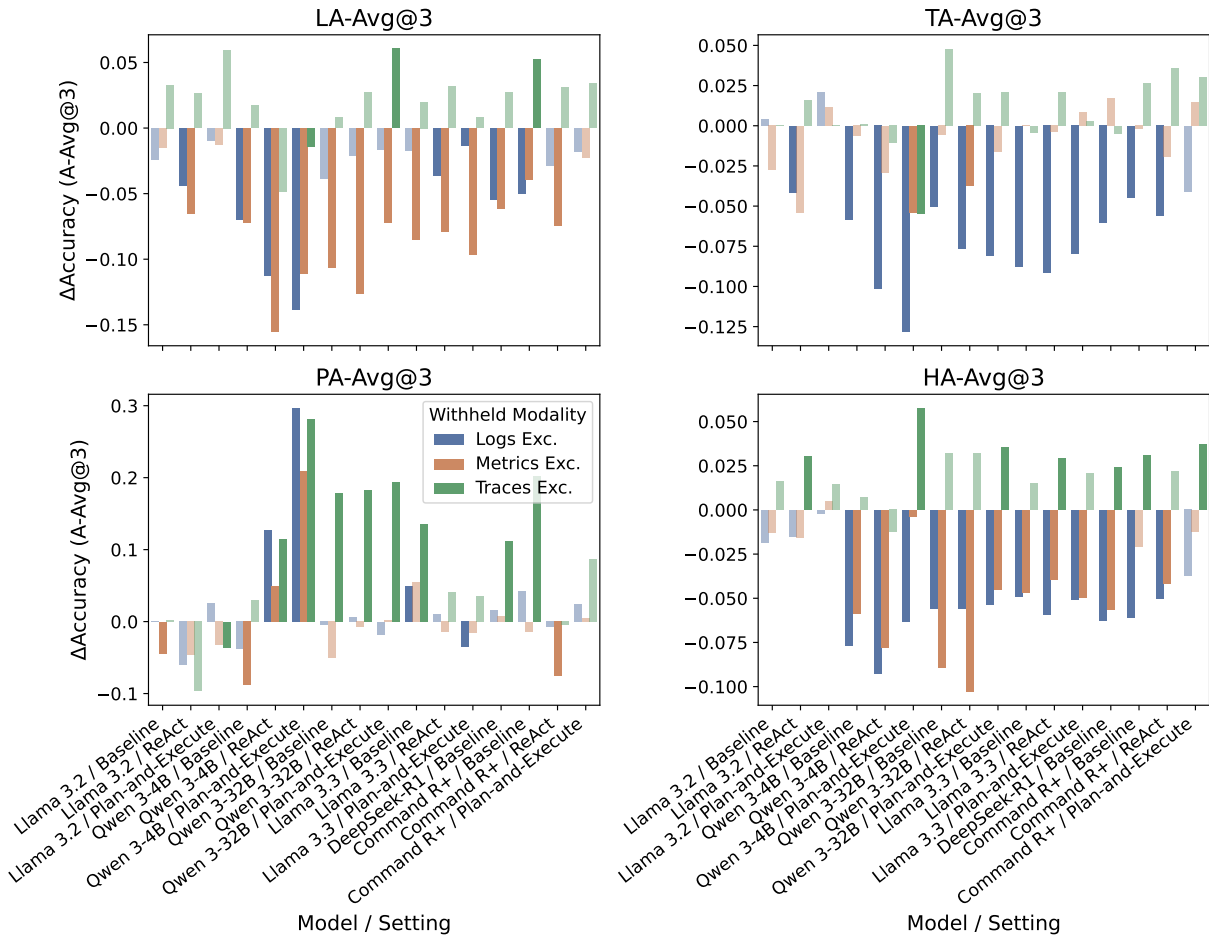


Figure 5.2: Change in accuracy ($\Delta A\text{-Avg}@3$) for withheld alert modalities aggregated across datasets \mathcal{A} and \mathcal{B} . **Solid bars** denote statistically significant differences ($p < 0.05$) according to the Wilcoxon signed-rank test, while **faded bars** indicate changes that are not statistically significant.

alerts were often noisy and voluminous, making alert triage more difficult and drowning out important yet infrequent signals. In their absence, we noticed that the analysis and exploration of the LLM were more focused and better grounded in the system KG.

These effects compound in the overall hypothesis accuracy: trace exclusion produces the strongest positive shift (ΔHA up to $+0.07$), while log and metric exclusions yield degradation (ΔHA down to -0.10).

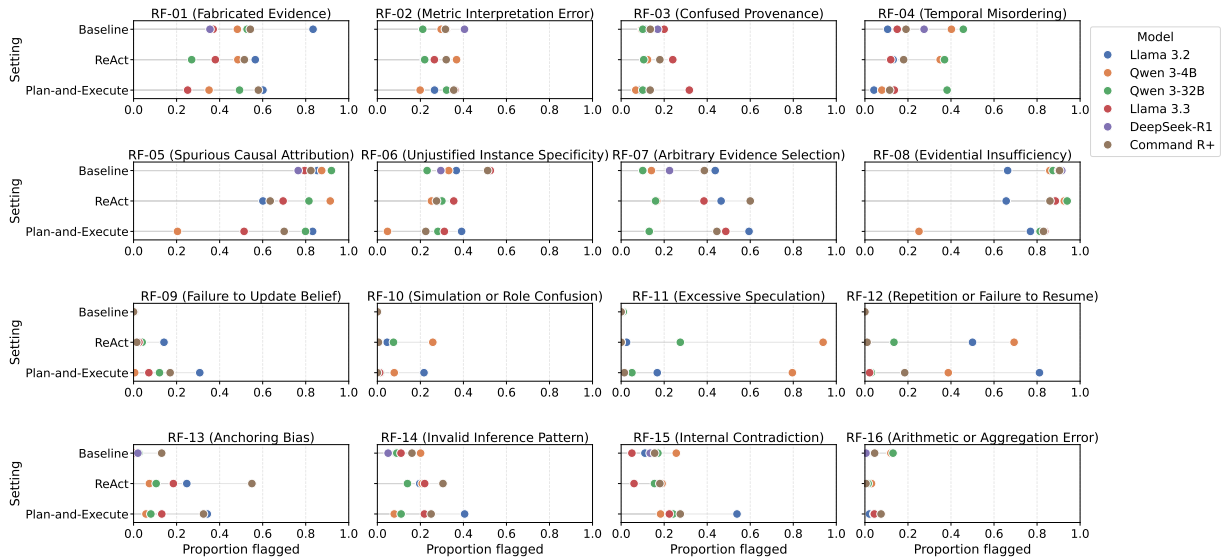


Figure 5.3: Prevalence of reasoning failures (RF) in RCA outputs.

Answer to RQ_2 : *Excluding metrics or logs markedly reduces accuracy, indicating that these modalities are most effectively leveraged by LLMs—metrics for fault localization and logs for interpreting fault type. Conversely, excluding traces often improves accuracy, suggesting that current models struggle to integrate raw trace alerts and that their inclusion can distract reasoning. Overall, these results identify which modalities LLMs can reason with effectively and which require better integration strategies for LLM-based RCA.*

5.3 RQ_3 : What reasoning failures appear in agent inference traces?

To answer RQ_3 , we identify common reasoning failures observed in RCA inference traces (see Table 3.2) and quantify their prevalence, following the procedure described in Section 3.3.2. Table 5.2 and Figure 5.3 show the relative proportions of RFs identified by the LLM-as-a-Judge.

Observation 8: *Procedural reasoning failures vary substantially across models and workflows, indicating model-specific instability under agentic con-*

Table 5.2: Proportion of Reasoning Failures by Workflow

ID	Category	Reasoning Failure	<i>Straight-Shot</i> (Baseline)	REACT	<i>Plan-and-Execute</i>
RF-01	General	Fabricated evidence	0.518	0.441	0.451
RF-02	RCA-specific	Metric interpretation error	0.311	0.275	0.302
RF-03	RCA-specific	Confused provenance	0.135	0.154	0.142
RF-04	General	Temporal misordering	0.263	0.226	0.158
RF-05	General	Spurious causal attribution	0.839	0.725	0.602
RF-06	RCA-specific	Unjustified instance specificity	0.377	0.294	0.243
RF-07	RCA-specific	Arbitrary evidence selection	0.279	0.362	0.343
RF-08	General	Evidential insufficiency	0.853	0.851	0.698
RF-09	Procedural	Failure to update belief	0.001	0.047	0.127
RF-10	Procedural	Simulation or role confusion	0.000	0.071	0.056
RF-11	Procedural	Excessive speculation	0.003	0.222	0.209
RF-12	Procedural	Repetition or failure to resume	0.000	0.253	0.259
RF-13	General	Anchoring bias	0.035	0.238	0.180
RF-14	General	Invalid inference pattern	0.116	0.214	0.202
RF-15	General	Internal contradiction	0.146	0.154	0.279
RF-16	Procedural	Arithmetic or aggregation error	0.052	0.009	0.047

trol. Procedural RFs—including failure to update belief (RF-09), simulation/role confusion (RF-10), excessive speculation (RF-11), and repetition/stalling (RF-12)—show the widest spread of RF prevalence across models and workflows. Numeric/aggregation errors (RF-16) are rare across all conditions. For agentic workflows, Llama 3.2 and Qwen 3 (4B, 32B) commit procedural errors at a far higher frequency. Prevalence rises from REACT to *Plan-and-Execute* for Llama 3.2, while decreasing for Qwen 3 (4B, 32B), indicating differing sensitivity to iterative planning. This suggests that agentic workflows can amplify latent procedural weaknesses in some models while stabilizing reasoning in others, underscoring the need for model-specific workflow tuning rather than treating planning as universally beneficial. Prevalence for other (larger) models is comparatively low. We hypothesize that small models struggle to exploit agentic procedures.

Observation 9: RCA-specific failures are consistent across models, reflecting domain-grounding rather than workflow sensitivity. Metric interpretation errors (RF-02), provenance confusion (RF-03), incorrect instance-specificity (RF-06), and arbitrary triage (RF-07) show few clear trends across models—Qwen models (4B, 32B) are at the lower end for most prevalence counts—and minimal differences between REACT and *Plan-and-Execute*. The comparatively narrow spread across models suggests that these failures stem from the inability to perform RCA-specific reasoning, indicating

insufficient domain specification. We suggest that improving RCA performance will likely require explicit domain guidance to properly interpret, process, and prioritize the input alert information, and reasoning patterns (e.g., triaging heuristics, common propagation mechanisms) that an LLM should follow. Fine-tuning these behaviours could also be beneficial.

Observation 10: General per-hypothesis reasoning failures are widespread, indicating that hypothesis claims are often weakly supported. This subset (fabricated evidence (RF-01), evidential insufficiency (RF-08), temporal misordering (RF-04), spurious causality (RF-05)) has the highest average prevalence across RFs, meaning that single-hypothesis statements are frequently under-justified or logically inconsistent. Notably, RF-04 appears more often in Qwen models. The high prevalence suggests that hypotheses should not be trusted at face value; effective RCA systems must apply evidence sufficiency checks, temporal ordering validators, and KG-consistency filters before ranking or selecting hypotheses.

Observation 11: General cross-cutting reasoning failures vary by model family, with “thinking”-oriented models showing lower prevalence under agentic workflows. Llama 3.2 and Command R+ show consistently higher prevalence of cross-cutting RFs (e.g., anchoring, internal contradiction, invalid inference patterns), particularly in multi-step traces. In contrast, Qwen models exhibit lower rates of these failures under both REACT and *Plan-and-Execute*. This suggests that global reasoning coherence may depend more on model training style and inductive biases than workflow structure alone.

Answer to RQ₃: RFs are pervasive across models and workflows and group into procedural, RCA-specific, and general failure categories. Prevalence varies by model and workflow (notably elevated procedural RFs in Llama 3.2 and Qwen 3 variants under agentic control), and general per-hypothesis failures have the highest average prevalence. RCA-specific reasoning remains insufficient, and general (non-domain-specific) RFs continue to dominate outputs.

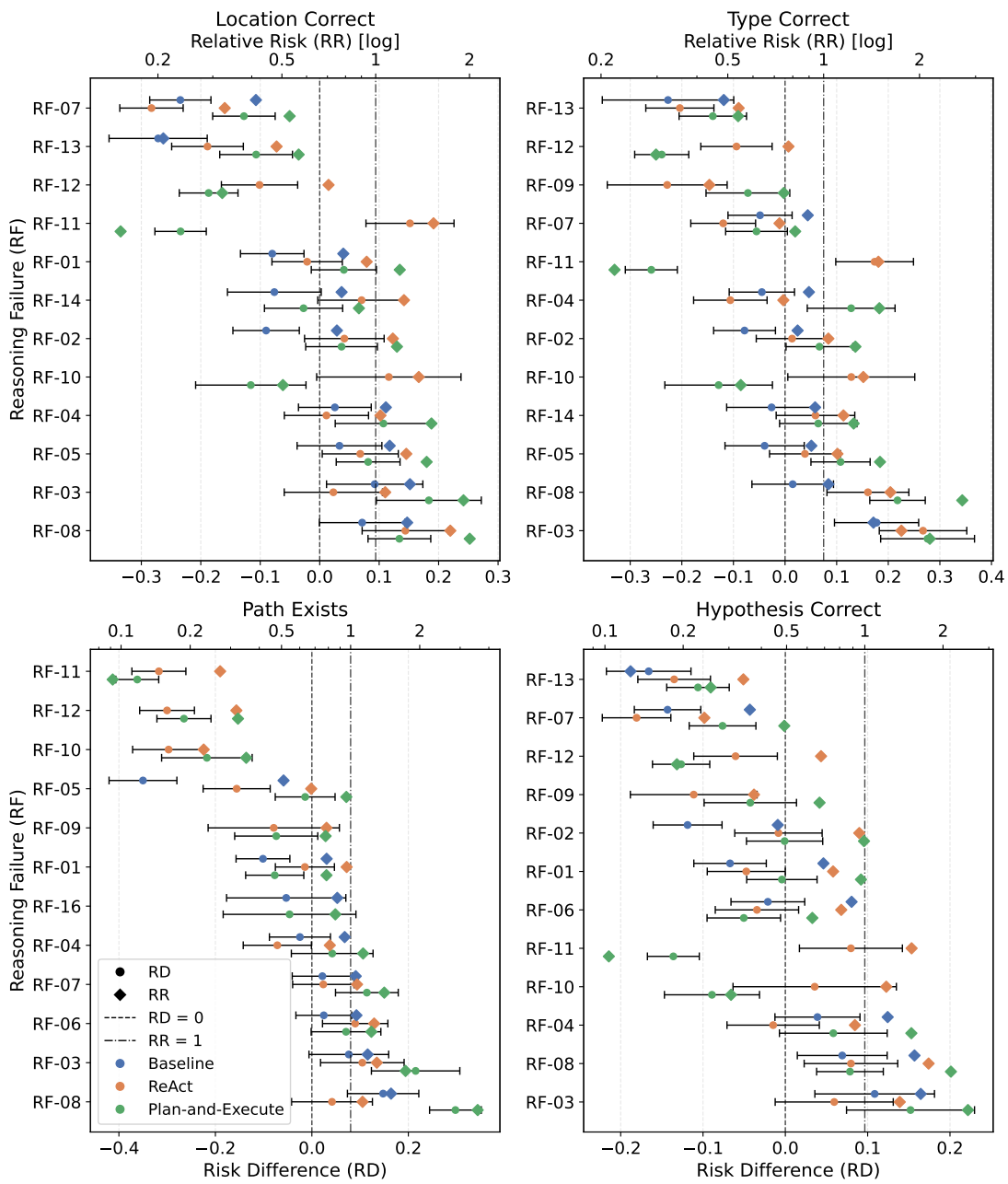


Figure 5.4: Risk difference (RD, Wilson 95% CI) and relative risk (RR, log scale) for the top-12 RFs.

5.4 RQ_4 : How does the presence of reasoning failures affect the likelihood of generating correct RCA hypotheses?

To answer RQ_4 , we quantify the association between detected RFs (by the LLM-as-a-Judge) and hypothesis correctness (generated during RCA Inference) using two measures: *risk difference* $RD = p_1 - p_0$ and *relative risk* $RR = p_1/p_0$, where $p_1 = P(\text{correct}|\text{RF present})$ and $p_0 = P(\text{correct}|\text{RF absent})$. Figure 5.4 summarizes results across correctness types. By convention, $RD < 0$ (or $RR < 1$) indicates an RF associated with *reduced* correctness, while $RD > 0$ (or $RR > 1$) indicates an association with *improved* correctness.

Observation 12: *The clearest negative predictors of correctness are anchoring bias (RF-13), repetition or stalled progress (RF-12), arbitrary evidence selection (RF-07), and failure to update belief (RF-09).* Across location, type, and hypothesis, these RFs correspond to at least a 15% drop in correctness ($RD < -0.15$) and make correct predictions at least 45% less likely than when the failure is absent ($RR < 0.55$). For path existence, a distinct set—simulation or role confusion (RF-10), excessive speculation (RF-11), and RF-12—shows the strongest negative associations ($RD < -0.29$, $RR < 0.32$), suggesting cases where reasoning becomes decoupled from the system knowledge grounding.

Observation 13: *Agentic workflows tend to be associated with smaller absolute risk differences (RDs closer to zero) than the non-agentic Straight-Shot across metrics.* This pattern indicates that correct outcomes are less strongly associated with the presence of RFs under agentic workflows, consistent with the view that iterative reasoning and answer formulation may dilute the marginal influence of individual reasoning failures. Notably, *Plan-and-Execute* often exhibits significantly higher RD values (closer to zero) than *Straight-Shot*. However, agentic traces contain more RFs on average (REACT 5.1; *Plan-and-Execute* 4.8; *Straight-Shot* 4.6). RF co-occurrence (multiple RFs in one trace) and early error compounding (an RF that cascades) could contribute to the observation from RQ_1 (agentic workflows do not reliably improve accuracy). Thus, although per-RF associations are weaker under agentic workflows, compounding effects can produce net accuracy loss.

Observation 14: *The relative association between RFs and correctness differs substantially between ReAct and Plan-and-Execute.* In most cases, *Plan-and-Execute* shows higher (less negative) RD and RR values (though not always statistically significant), but two exceptions stand out. Repetition or failure to resume (RF-12) is more strongly associated with reduced correctness under *Plan-and-Execute* than

under REACT for location, type, and hypothesis correctness ($\Delta RR \approx 0.38$), suggesting that the replanning step may correlate with persistence of stalled reasoning rather than its resolution, even when prior steps and actions are available. A more extreme divergence occurs for RF-10 and RF-11, which appear positively associated with correctness under REACT ($RR > 1.49$) but negatively under *Plan-and-Execute* ($RR < 0.22$). These RFs co-occur frequently with execution errors (e.g., recursion limit or replan errors) in 68.6%/80.6% of *Plan-and-Execute* versus 7.4%/0.5% of REACT samples for RF-10/RF-11, respectively. This pattern reflects the frequent co-occurrence of speculative or fabricated assumptions and role-confused reasoning with workflow errors in *Plan-and-Execute*. Regardless of workflow, both RF-10 and RF-11 show strong negative associations with path recovery ($RD < -0.29$, $RR < 0.32$), consistent with loss of grounding in the system knowledge (KG).

A small subset of RFs—notably confused provenance (RF-03) and evidential insufficiency (RF-08)—exhibit positive RD ($RR > 1$), an ostensibly counterintuitive outcome for failure labels. This arises from our per-sample aggregation scheme: correctness is marked if *any* attribute is correct, and per-hypothesis RFs are flagged if they occur in *any* hypothesis within the sample. Consequently, an RF appearing in one hypothesis can co-occur with a different correct hypothesis in the same sample, producing an apparent net positive association. Qualitatively, we observed that models often “reach” for additional (second/third) hypotheses (as required by the task) when such RFs are flagged. This artifact reflects the forced three-hypothesis task (even if confident in only one or two) rather than model behaviour.

Answer to RQ₄: *The presence of reasoning failures—particularly anchoring bias (RF-13), repetition or stalled progress (RF-12), arbitrary evidence selection (RF-07), and failure to update belief (RF-09)—is associated with substantially reduced RCA correctness. Agentic workflows are often associated with smaller negative risk differences, but do not eliminate them. However, RF co-occurrence and early failure compounding remain plausible contributors to the accuracy degradations observed in RQ₁.*

Chapter 6

Discussion

Based on the results presented in Chapter 5, we identify four key implications for the design and evaluation of LLM-based RCA agents.

6.1 The “Agentic Tax” and Competency Threshold

The results from RQ_1 and RQ_3 reveal a counter-intuitive finding: adding agentic complexity (via REACT and *Plan-and-Execute*) does not reliably or uniformly improve performance. We observe diminishing or negative returns for recovering the root-cause location and type accuracy (Observation 2). In fact, agentic complexity, in many cases, degrades performance for smaller models.

We identify a competency threshold: below a certain model size, the cognitive overhead of managing an agentic workflow (planning, tool selection, memory management) exceeds the model’s effective capability and is associated with procedural reasoning failures, such as excessive speculation (RF-11) and repetition or failure to resume (RF-12). Consistent with this view, RQ_1 shows that smaller models not only underperform but frequently fall below random-guessing baselines in LA, TA, and HA as workflow complexity increases, indicating that agentic scaffolding can amplify rather than correct latent weaknesses. These findings suggest that, for RCA, single prompts or lightweight architectures are often sufficient and more robust than complex agentic loops when using currently available open-source models.

Furthermore, this complexity imposes a severe latency penalty: our experiments show that REACT and *Plan-and-Execute* increased average inference time by 40% and 354%,

respectively, compared to the *Straight-Shot* baseline (see Section A.3.2). Consequently, future RCA systems that target lower-cost (smaller) models should favour linear or minimally orchestrated pipelines over dynamic agentic loops to avoid this “agentic tax.”

6.2 Domain-Grounding vs. General Reasoning

In RQ_3 , we observed that RCA-specific failures are broadly consistent across models, whereas procedural failures varied. Scaling a model (e.g., from 4B to 32B) or employing “thinking” models (e.g., Qwen vs. Command R+) reduces procedural instability (the model is better at following instructions within the agentic confines) but does not necessarily fix domain-interpretation issues (the model still struggles to understand how a specific alert implies a specific fault).

Thus, general reasoning capacity is necessary but not sufficient for RCA. The consistent prevalence of RCA-specific failures suggests that general-purpose LLMs lack the inductive biases required for reliability engineering. Notably, RQ_4 shows that arbitrary evidence selection (RF-07)—ignoring standard triage heuristics—is strongly negatively correlated with correctness. This indicates that LLMs do not inherently follow systematic software-reliability procedures and motivates domain-specific interventions, such as fine-tuning on RCA tasks or explicitly embedding domain-specific heuristics into the workflow. In practice, the latter can be supported by “expert” tools that either implement concrete triage heuristics or perform domain-relevant analytic steps (e.g., interpreting metric anomalies or generating statistical summaries). These mechanisms make the diagnostic procedure explicit, rather than expecting the model to infer such practices solely from data.

At the same time, we observe persistent general reasoning failures that are independent of domain heuristics. In particular, anchoring biases (RF-13) frequently stifle graph exploration and inhibit multi-hypothesis formation. These behaviours suggest that the models’ internal search processes are too myopic, even when the relevant evidence is correctly identified. We hypothesize that mechanisms enforcing self-consistency or promoting diversity of considerations (e.g., Tree-of-Thought or structured branching) could help mitigate this second class of failures by compelling the model to explore alternatives earlier in the inference process.

6.3 More Data is Not Always Better

The RQ_2 findings indicate that excluding traces often improves accuracy, while excluding metrics and logs generally degrades it. In distributed systems, traces are voluminous and frequently reflect tangled or many-to-many direct dependencies (i.e., “spaghetti” architectures [10]). As a result, such trace alerts can distract models and induce overfocus on call relationships at the expense of other signals (metrics, logs) or other relationship types (e.g., deployment).

This confusion is quantified in RQ_3 , where provenance confusion (RF-03) and incorrect instance-specificity (RF-06) remain prevalent even when all alert modalities are available. This suggests that simply increasing data availability does not guarantee better grounding; rather, without clear signal separation, additional telemetry contributes to the overload we observe even after alert extraction. Such observations challenge naïve Retrieval-Augmented Generation (RAG) approaches that retrieve and dump all relevant telemetry into the model context. This suggests that multi-modal data can interact competitively, not just cooperatively. For traces, higher-level abstractions of anomalous behaviour (for example, “latency bottleneck detected in *webservice1*”) are preferable inputs to an LLM than unaggregated trace dumps. In short: *context selection* is more important than *context length*.

6.4 The Danger of “Right for the Wrong Reasons”

The results from RQ_1 and RQ_4 reveal a disconnect between HA and PA. A model can produce correct root-cause hypotheses (high HA) while hallucinating propagation paths (low PA) or exhibiting severe reasoning failures (e.g., fabricated evidence). This indicates insufficient grounding in the provided scenario context (alerts, KG).

From RQ_1 , we also observe a pronounced gap between partial accuracy (LA, TA) and full-hypothesis accuracy (HA), particularly on \mathcal{B} . Relatively high LA and TA coinciding with low HA reflect non-overlapping correctness: models often identify plausible attributes (i.e., fault location or type), yet fail to combine the correct attributes within the same explanation.

In operational settings, correct guesses that are poorly justified (or worse, contain hallucinated explanations) undermine operator trust. In over 70% of judged samples, we observed evidential insufficiency (RF-08). The possible consequences range from wasted operator time to erroneous fixes that introduce regressions. This risk is compounded in agentic workflows: while individual RF impacts may appear statistically diluted (RQ_4),

agentic workflow traces nonetheless contain a higher frequency of RFs, and early failures can compound to produce the degraded accuracy reported in RQ_1 .

Operators must be able to investigate the *why*, not just the result. Crucially, RQ_4 identifies several RFs—anchoring, arbitrary evidence selection, and failure to update beliefs—as strong negative predictors of correctness. This implies that high-level output metrics hide underlying fragility and that evaluation should explicitly detect and penalize these reasoning patterns for future improvement. Therefore, evaluation metrics for RCA should penalize “lucky guesses” and include checks on intermediate reasoning traces and provenance, not only final labels, to support safe deployment.

Chapter 7

Limitations and Threats to Validity

7.1 Limitations

Our study adopts several controlled simplifications to isolate and evaluate reasoning, which naturally introduce limitations. First, the alerts provided to the agent are due to a *single* known root-cause fault. In practice, concurrent faults can occur, producing overlapping and interleaved alerts that require alert correlation and attribution to disambiguate causality. Since our goal is to isolate reasoning in the RCA task, we therefore restrict to single-fault scenarios. Although we follow practices common in prior work for alert extraction, the alerts provided to the agent are not gold-labelled and may contain noise. We minimize this as much as possible to study reasoning in isolation; however, imperfect monitoring and signal quality remain a realistic concern in real-world production systems. Moreover, longer inference times and LLM context-window constraints may emerge for larger systems and higher alert volumes. Optimized alert selection, prioritization, or representation (e.g., OBServation Snapshot Key (OBSK) [73]), may help address this in future work.

We manually construct the system KGs from available documentation to ensure correctness. While this guarantees graph quality for controlled evaluation, it does not reflect real-world deployment scenarios where such knowledge must be inferred automatically. In principle, these graphs could be feasibly constructed, likely with more errors than in our experiments, via retrieval-based or expert-agent pipelines, but we leave such integration to future work. We further serialize the KGs into natural language using a Graph2Text strategy [67, 23, 84], prioritizing interpretability and prompting compatibility over recent optimization-oriented graph encodings [6, 16, 7] which could improve performance.

7.2 Internal Validity

To isolate and assess the reasoning behaviour of LLM agents, we minimize confounding factors through explicit natural-language prompting and data representation, and by reducing the action space to deterministic tools with clearly defined semantics. However, results may still be sensitive to the phrasing of prompts, tool descriptions, or representations of system information (e.g., alerts, KG), which can subtly influence agent behaviour. We mitigate some of these effects by incorporating different representation strategies of system information.

Taxonomy elicitation relies on qualitative coding and human interpretation, which can introduce subjectivity. Structured definitions, exemplars, and paired intermediate LLM–human annotation rounds were used to align interpretations, but some coder bias may remain. The taxonomy is also induced from a limited number of samples (170), which may not capture rare reasoning failure types; however, we believe it adequately captures common and impactful failures. Finally, LLM-based judging introduces uncertainty due to model preferences and inductive biases. We mitigate this by employing a distinct, strong evaluator model (GPT-5), adhering to best practices for LLM-as-a-Judge systems, and conducting an independent two-reviewer review of a random subset of annotations to increase confidence in the reliability of our results.

7.3 External Validity

Our evaluation is limited to a small set of open-source benchmarks and LLM models. While these case studies represent diverse and realistic microservice-based systems and we consider LLM models from a variety of model families, our findings may not generalize to other domains, large-scale production environments, or proprietary models. Moreover, our findings are specific to the agent workflows and input modalities explored in this work. To mitigate this threat, we make our methodology and evaluation strategy publicly available [56] to encourage replication and extension studies across different settings to further assess the generalizability of the findings.

Chapter 8

Related Work

We discuss related work in three dimensions: (a) data-driven RCA (Section 8.1), (b) LLM-based RCA (Section 8.2), and (c) evaluation of LLM reasoning (Section 8.3).

8.1 Data-driven RCA

RCA is essential in postmortem investigations in identifying underlying issues that lead to system failures. In complex, distributed software systems such as cloud-native environments with dense dependency structures, asynchronous communication, high degrees of concurrency, and the potential for failure propagation with wide-reaching impact, RCA aims to rapidly localize causative faults to improve system stability and reduce potential downtime [62].

Modern RCA approaches are automated and data-driven [71], contrasting earlier manual or rule-based methods, and typically rely on telemetry data (metrics, traces, and logs). Initial approaches primarily relied on a single data modality, including metric-based methods [8, 36, 41, 54, 29, 27, 34, 66], log-based-methods [40, 31, 57, 95, 94, 64], and trace-based methods [85, 37, 87].

However, recognizing that a holistic system view is essential, recent advances focus on multi-modal data integration [86, 32, 88, 22, 98, 82, 97, 72, 24, 90], combining metrics, traces, and/or logs to enhance diagnostic accuracy and speed by capturing diverse dimensions of system performance and interactions. By correlating information across different data types, these advanced approaches can capture more complex failure patterns, leading to substantial improvements in diagnostic performance. Beyond data source, approaches

are further broadly categorized by technique, such as statistical-based [40, 57, 70, 61], machine learning (ML)-based [34, 59, 98], deep learning (DL)-based [18, 88], and graph-based methods [78, 86, 22]. Notably, causal-, dependency-, and knowledge-graphs generally exemplify graph-based RCA, enabling comprehensive mapping of complex inter-service dependencies and supporting causal inference.

8.2 LLM-based RCA

The promise of LLMs has led to a surge of interest in using them for RCA [1, 73, 96, 58, 9, 19, 92, 25, 48, 79, 55]. Recent work has explored the use of LLMs in both end-to-end prompting and agent workflows. Early LLM-based approaches to RCA have explored fine-tuning [1] or in-context learning [96, 19, 58, 9] for domain adaptation, typically using labelled historical incident reports. These incident reports—generally comprising a title, description, and root cause of an incident—, while informative, often contain lengthy and noisy information, such as raw logs, which can obscure the core details of the incident. Consequently, LLMs are also frequently tasked with summarizing these reports before proceeding with RCA [9, 96, 19, 58]. While such approaches effectively leverage LLMs’ pattern-matching and information-aggregation abilities, they provide the LLM with a limited (and often summarized) view of the system, potentially failing to capture the complexities of entity interactions or provide useful diagnostic signals (e.g., multi-modal telemetry data). Further, limiting LLM input context to static or historical information can hinder its ability to adapt to the system’s current state and evolving interactions.

Other works explore integrating LLMs as agents into retrieval- or tool-augmented frameworks [58, 55, 93, 35] or multi-agent systems [73, 92, 25, 48, 79]. These workflows generally equip an LLM with actions to retrieve additional system or diagnostic information, such as service dependencies [92], more detailed incident information [58, 73], similar historical incidents [58], relevant code snippets and execution paths [35], or other LLM expert agents (e.g., as analytical tools like code [73] or fault probability analyzers [92], or python executors [79]).

Although many approaches show promise, confounding framework factors often make it challenging to assess LLMs’ actual ability to reason about RCA tasks. OpenRCA [79] presents an evaluation framework for LLM-based RCA tasks, but frames each task as a sequence of sub-tasks, including anomaly detection, failure identification, and fault analysis. Additionally, their RCA-agent relies on open-ended code execution and telemetry analysis, introducing multiple sources of potential failure. This conflates the core RCA ability with unrelated skills such as anomaly detection, code generation, and error handling. Instead,

we perform anomaly detection as a pre-processing step. Crucially, the inherent complexity and entanglement of reasoning with auxiliary tasks in these frameworks complicates the isolation and reliable assessment of the LLM’s core diagnostic reasoning, which motivates the need for simplified evaluation settings and more expressive metrics discussed next.

8.3 Evaluation of LLM Reasoning

8.3.1 General Evaluation Strategies for LLM Reasoning

To date, a standardized methodology for assessing the reasoning capabilities of LLMs is lacking [44]. Nonetheless, some evaluation frameworks have emerged for analyzing the behaviour of LLMs in reasoning tasks. *Rationale-based* (where emphasis is placed on the reasoning generated by the model, as opposed to *conclusion-based* evaluation) evaluation options for LLM outputs generally include structured parsing, quantitative metrics, or qualitative inspection [44].

Since the rationales produced in our framework are generally unstructured (i.e., they cannot be parsed into formalized representations such as first-order logic or causal graphs), we cannot use such methods. Furthermore, the inherent complexity of the RCA task—which requires dynamic tool use, multi-step inquiry, and multi-hypothesis formation—limits the applicability of current quantitative unsupervised methods. Current quantitative unsupervised reasoning evaluation frameworks (e.g., ROSCOE [20], ReCEval [50] BERTScore [91]) are built primarily for monotonic *chain-of-thought* outputs. At the core, these rely on semantic alignment with the reasoning task and its previous steps. Although our inference traces do contain some such reasoning traces, they are inherently non-monotonic, with multi-hypothesis formation, graph and alert exploration, and interleaved tool calls. We therefore find that such metrics, which heavily rely on surface-level lexical overlaps, fail to capture deeper nuances and do not provide sufficient insight into the exploration, considerations, and rationale of the proposed root-cause hypotheses.

Finally, qualitative inspection is commonly employed for less structured rationales, either through human judgments or diagnostic agents. Human judgments, however, are time-consuming and labour-intensive. The primary gap thus lies in the lack of an automated, systematic framework for evaluating the quality of the intermediate reasoning steps and the correctness of the inferred fault propagation path—a critical component of RCA in distributed systems.

LLM-as-a-Judge. To address the scalability and reproducibility issues inherent in manual human grading, recent research has leveraged the advanced reasoning capabilities of LLMs themselves to evaluate the quality of other LLMs’ outputs, a technique known as *LLM-as-a-Judge* [99, 68, 39]. This paradigm replaces human evaluators with a highly capable LLM that is prompted with clear scoring criteria, context, and the output to be judged [33]. The LLM-as-a-Judge method is particularly well-suited for evaluating complex, unstructured reasoning traces, as it allows for fine-grained, criterion-based assessment of logical flow and factual correctness (relative to the given context), all while maintaining the speed and scalability necessary for extensive empirical studies. LLM judges have been widely adopted across specialized fields such as finance (e.g., quantitative investment and credit scoring), law (e.g., legal consultation and reasoning benchmarks), medical diagnostics, and education (e.g., automated essay scoring) [21].

8.3.2 Evaluation Approaches in LLM-based RCA

Most current evaluation practices for LLM-based RCA predominantly focus on the accuracy of the final predicted root-cause entity or fault classification, often overlooking the quality of the diagnostic process itself. Methods based on lexical and semantic similarity are commonly employed to compare a model’s root-cause hypothesis against a reference description from historical incident reports [1, 73, 96, 58, 19]. While useful, such metrics can overlook critical reasoning flaws, e.g., identifying the failure type but mislocalizing its source or inferring an incorrect propagation path. Some works supplement their evaluation with coarse-grained human assessments, such as a “usefulness” score[92], “correctness” score[19], or count of reasoning errors [58]. However, these subjective, coarse-grained scores are non-reproducible and scale poorly, making large-scale comparative studies infeasible. Qiu et al. [55] forgo automated evaluation approaches for this reason, adopting a manual evaluation approach by grading LLM RCA outputs on binary (yes/no) criteria (i.e., concept accuracy, no inaccuracies (actuality), relevance, and correct and complete). Further, many of these approaches are closed-source or do not publish their datasets [73, 58, 9, 96, 19, 48], limiting further analysis. This deficiency motivates the need for a scalable, automated, and reproducible methodology capable of assessing diagnostic reasoning quality for the RCA task.

To the best of our knowledge, no prior work has implemented and validated an LLM-as-a-Judge setup specifically for the RCA task. Our work pioneers this application by utilizing the LLM judge to identify reasoning failures within the agent’s diagnostic trace. This enables a systematic and transparent diagnosis of LLM limitations in complex fault-propagation scenarios.

Chapter 9

Conclusion

In this thesis, we present a reasoning-centred evaluation framework for LLM-based RCA agents that isolates the reasoning process from framework complexity. By simplifying the RCA setup, our approach enables transparent assessment of LLM reasoning capabilities for RCA. We find that the simplified setup still poses ample challenge: accuracy is low across models and workflows, and reasoning failures are widespread. Agentic workflows often yield diminishing or negative returns, particularly for smaller models. Yet, interactive exploration remains necessary—providing full system context upfront is infeasible as systems scale in production environments. This tension suggests that progress will require carefully selecting and structuring the context, and improving specific reasoning capabilities of underlying LLMs rather than agentic structure alone.

Our analysis shows that RCA-specific reasoning by LLMs is frequently inconsistent with standard diagnostic heuristics and causal propagation mechanisms, while general and procedural failures (e.g., anchoring, arbitrary evidence selection, stalled reasoning) strongly predict incorrect outcomes. Addressing these will likely require encouraging early hypothesis diversification (e.g., via tree-of-thought), self-consistency or critique mechanisms, evidence sufficiency checks, and explicit domain guidance for triage and causal reasoning. Finally, understanding how reasoning failures co-occur and compound remains an important open direction, particularly in multi-step or multi-agent settings where early missteps cascade.

Overall, our findings highlight the need for transparency and reasoning quality in RCA agent design. Our reasoning failure taxonomy and rationale-based evaluation scheme offer reusable components for diagnosing and improving RCA reasoning. More broadly, this work moves toward RCA agents that are not only automated but also interpretable, in-

spectable, and better aligned with the practical demands of fault diagnosis in complex systems.

References

- [1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1737–1749, 2023. doi:[10.1109/ICSE48619.2023.00149](https://doi.org/10.1109/ICSE48619.2023.00149).
- [2] Meta AI. Llama 3.2. https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md, 2024. Accessed 2025-11-04.
- [3] Meta AI. Llama 3.3. https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/MODEL_CARD.md, 2024. Accessed 2025-11-04.
- [4] Meta AI. Llama 4. https://github.com/meta-llama/llama-models/blob/main/models/llama4/MODEL_CARD.md, 2025. Accessed 2025-11-04.
- [5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1(1):11–33, 2004. doi:[10.1109/TDSC.2004.2](https://doi.org/10.1109/TDSC.2004.2).
- [6] Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiang Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. GraphLLM: Boosting Graph Reasoning Ability of Large Language Model. *IEEE Transactions on Big Data*, pages 1–9, 2025. doi:[10.1109/TBDATA.2025.3627488](https://doi.org/10.1109/TBDATA.2025.3627488).
- [7] Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. GraphWiz: An Instruction-Following Language Model for Graph Computational Problems. In *Proceedings of Conference on Knowledge Discovery and Data Mining (KDD)*, page 353–364, New York, NY, USA, 2024. Association for Computing Machinery. doi:[10.1145/3637528.3672010](https://doi.org/10.1145/3637528.3672010).

- [8] Pengfei Chen, Yong Qi, and Di Hou. CauseInfer: Automated End-to-End Performance Diagnosis with Hierarchical Causality Graph in Cloud Environment. *IEEE Transactions on Services Computing*, 12(2):214–230, 2019. doi:[10.1109/TSC.2016.2607739](https://doi.org/10.1109/TSC.2016.2607739).
- [9] 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)*, page 674–688, New York, NY, USA, 2024. Association for Computing Machinery. doi:[10.1145/3627703.3629553](https://doi.org/10.1145/3627703.3629553).
- [10] Binildas Christudas and Tarun Telang. *Distributed Computing Architecture Landscape*, pages 1–25. Apress, Berkeley, CA, 2025. doi:[10.1007/979-8-8688-1606-2_1](https://doi.org/10.1007/979-8-8688-1606-2_1).
- [11] CloudWise OpenSource. GAIA: Generic AIOps Atlas. <https://github.com/CloudWise-OpenSource/GAIA-DataSet>, 2022. Accessed 2025-06-03.
- [12] Cohere. Command R+. <https://docs.cohere.com/v2/docs/command-r-plus>, 2024. Accessed 2025-11-04.
- [13] Juliet Corbin and Anselm Strauss. *Basics of Qualitative Research (3rd ed.): Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, Inc., Thousand Oaks, California, 2025. doi:[10.4135/9781452230153](https://doi.org/10.4135/9781452230153).
- [14] DeepSeek-AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025. arXiv:[2501.12948](https://arxiv.org/abs/2501.12948).
- [15] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, 2017. doi:[10.1007/978-3-319-67425-4_12](https://doi.org/10.1007/978-3-319-67425-4_12).
- [16] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a Graph: Encoding Graphs for Large Language Models. In *NeurIPS Workshop: New Frontiers in Graph Learning*, 2023. URL: <https://openreview.net/forum?id=7CAJpRo1Q8>.
- [17] Nan Fu, Guang Cheng, Yue Teng, Guangye Dai, Shui Yu, and Zihan Chen. Intelligent Root Cause Localization in MicroService Systems: A Survey and New Perspectives. *ACM Comput. Surv.*, 57(12), July 2025. doi:[10.1145/3736755](https://doi.org/10.1145/3736755).

- [18] Yu Gan, Guiyang Liu, Xin Zhang, Qi Zhou, Jiesheng Wu, and Jiangwei Jiang. Sleuth: A Trace-Based Root Cause Analysis System for Large-Scale Microservices with Graph Neural Networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, volume 4, page 324–337, New York, NY, USA, 2024. Association for Computing Machinery. doi:[10.1145/3623278.3624758](https://doi.org/10.1145/3623278.3624758).
- [19] Drishti Goel, Fiza Husain, Aditya Singh, Supriyo Ghosh, Anjaly Parayil, Chetan Bansal, Xuchao Zhang, and Saravan Rajmohan. X-Lifecycle Learning for Cloud Incident Management using LLMs. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*, page 417–428, New York, NY, USA, 2024. Association for Computing Machinery. doi:[10.1145/3663529.3663861](https://doi.org/10.1145/3663529.3663861).
- [20] Olga Golovneva, Moya Peng Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. ROSCOE: A Suite of Metrics for Scoring Step-by-Step Reasoning. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL: <https://openreview.net/forum?id=xYlJRpzZtsY>.
- [21] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A Survey on LLM-as-a-Judge, 2025. [arXiv:2411.15594](https://arxiv.org/abs/2411.15594).
- [22] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Haifeng Shen, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. TrinityRCL: Multi-Granular and Code-Level Root Cause Localization Using Multiple Types of Telemetry Data in Microservice Systems. *IEEE Transactions on Software Engineering (TSE)*, 49(5):3071–3088, 2023. doi:[10.1109/TSE.2023.3241299](https://doi.org/10.1109/TSE.2023.3241299).
- [23] Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. GPT4Graph: Can Large Language Models Understand Graph Structured Data? An Empirical Evaluation and Benchmarking, 2023. [arXiv:2305.15066](https://arxiv.org/abs/2305.15066).
- [24] Yongqi Han, Qingfeng Du, Ying Huang, Pengsheng Li, Xiaonan Shi, Jiaqi Wu, Pei Fang, Fulong Tian, and Cheng He. Holistic Root Cause Analysis for Failures in Cloud-Native Systems Through Observability Data. *IEEE Transactions on Services Computing*, 17(6):3789–3802, 2024. doi:[10.1109/TSC.2024.3478759](https://doi.org/10.1109/TSC.2024.3478759).

- [25] Yongqi Han, Qingfeng Du, Ying Huang, Jiaqi Wu, Fulong Tian, and Cheng He. The Potential of One-Shot Failure Root Cause Analysis: Collaboration of the Large Language Model and Small Classifier. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, page 931–943, New York, NY, USA, 2024. Association for Computing Machinery. doi:[10.1145/3691620.3695475](https://doi.org/10.1145/3691620.3695475).
- [26] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40, 2017. doi:[10.1109/ICWS.2017.13](https://doi.org/10.1109/ICWS.2017.13).
- [27] Zilong He, Pengfei Chen, Yu Luo, Qiuyu Yan, Hongyang Chen, Guangba Yu, and Fangyuan Li. Graph based Incident Extraction and Diagnosis in Large-Scale Online Systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, New York, NY, USA, 2023. Association for Computing Machinery. doi:[10.1145/3551349.3556904](https://doi.org/10.1145/3551349.3556904).
- [28] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D’Amato, Goran Kasneci, Gianluca Demidov, Stefan Dietze, Anna G. Ferrandez, Jeff Z. Pan, Axel-Cyrille Ngonga Ngomo, Christoph Schwabe, Sabrina Kirrane, Valeria de Castro D., Amelie Gyrard, Tomi Kauppinen, Harsh Khare, Jie Song, and Gijs van Houwelingen. Knowledge Graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2021. doi:[10.1145/3447772](https://doi.org/10.1145/3447772).
- [29] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. Root Cause Analysis of Failures in Microservices through Causal Discovery. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 31158–31170. Curran Associates, Inc., 2022.
- [30] Mohammad Saiful Islam, Mohamed Sami Rakha, William Pourmajidi, Janakan Sivaloganathan, John Steinbacher, and Andriy Miranskyy. Anomaly Detection in Large-Scale Cloud Systems: An Industry Case and Dataset. In *IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 377–388, 2025. doi:[10.1109/ICSE-SEIP66354.2025.00039](https://doi.org/10.1109/ICSE-SEIP66354.2025.00039).
- [31] Tong Jia, Pengfei Chen, Lin Yang, Ying Li, Fanjing Meng, and Jingmin Xu. An Approach for Anomaly Diagnosis Based on Hybrid Graph Model with Logs for Distributed Services. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 25–32, 2017. doi:[10.1109/ICWS.2017.12](https://doi.org/10.1109/ICWS.2017.12).

- [32] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data. In *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1750–1762, 2023. doi:[10.1109/ICSE48619.2023.00150](https://doi.org/10.1109/ICSE48619.2023.00150).
- [33] Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. From Generation to Judgment: Opportunities and Challenges of LLM-as-a-judge. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2757–2791, Suzhou, China, November 2025. Association for Computational Linguistics. doi:[10.18653/v1/2025.emnlp-main.138](https://doi.org/10.18653/v1/2025.emnlp-main.138).
- [34] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 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)*, page 3230–3240, New York, NY, USA, 2022. Association for Computing Machinery. doi:[10.1145/3534678.3539041](https://doi.org/10.1145/3534678.3539041).
- [35] 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. In *IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 1346–1358, 2025. doi:[10.1109/ICSE55347.2025.00234](https://doi.org/10.1109/ICSE55347.2025.00234).
- [36] Jinjin Lin, Pengfei Chen, and Zibin Zheng. Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments. In Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu, editors, *International Conference on Service-Oriented Computing (ICSOC)*, pages 3–20, Cham, 2018. Springer International Publishing. doi:[10.1007/978-3-030-03596-9_1](https://doi.org/10.1007/978-3-030-03596-9_1).
- [37] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. MicroHECL: High-Efficient Root Cause Localization in Large-Scale Microservice Systems. In *IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 338–347, 2021. doi:[10.1109/ICSE-SEIP52600.2021.00043](https://doi.org/10.1109/ICSE-SEIP52600.2021.00043).
- [38] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation Forest. In *IEEE International Conference on Data Mining*, pages 413–422, 2008. doi:[10.1109/ICDM.2008.17](https://doi.org/10.1109/ICDM.2008.17).

- [39] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2511–2522, Singapore, December 2023. Association for Computational Linguistics. doi: [10.18653/v1/2023.emnlp-main.153](https://doi.org/10.18653/v1/2023.emnlp-main.153).
- [40] Siyang Lu, BingBing Rao, Xiang Wei, Byungchul Tak, Long Wang, and Liqiang Wang. Log-based Abnormal Task Detection and Root Cause Analysis for Spark. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 389–396, 2017. doi: [10.1109/ICWS.2017.135](https://doi.org/10.1109/ICWS.2017.135).
- [41] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. AutoMAP: Diagnose Your Microservice-based Web Applications Automatically. In *Proceedings of The Web Conference (WWW)*, page 246–258, New York, NY, USA, 2020. Association for Computing Machinery. doi: [10.1145/3366423.3380111](https://doi.org/10.1145/3366423.3380111).
- [42] Leonardo Mariani, Mauro Pezzè, Oliviero Riganelli, and Rui Xin. Predicting failures in multi-tier distributed systems. *Journal of Systems and Software*, 161:110464, 2020. doi: [10.1016/j.jss.2019.110464](https://doi.org/10.1016/j.jss.2019.110464).
- [43] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical Report SP 800-145, National Institute of Standards and Technology (NIST), 2011. doi: [10.6028/NIST.SP.800-145](https://doi.org/10.6028/NIST.SP.800-145).
- [44] Philipp Mondorf and Barbara Plank. Beyond Accuracy: Evaluating the Reasoning Behavior of Large Language Models - A Survey. In *Conference on Language Modeling (COLM)*, 2024. URL: <https://openreview.net/forum?id=Lmjgl2n11u>.
- [45] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. A Survey of AIOps Methods for Failure Management. *ACM Trans. Intell. Syst. Technol.*, 12(6), November 2021. doi: [10.1145/3483424](https://doi.org/10.1145/3483424).
- [46] OpenAI. Introducing GPT-5. <https://openai.com/index/introducing-gpt-5/>, 2025. Accessed 2025-11-04.
- [47] OpenAI. Introducing OpenAI o3 and o4-mini. <https://openai.com/index/introducing-o3-and-o4-mini>, 2025. Accessed 2025-11-04.

- [48] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui 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 on Web Conference, WWW '25*, page 422–431, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3701716.3715225.
- [49] Google Cloud Platform. Online Boutique. <https://github.com/GoogleCloudPlatform/microservices-demo>, 2020. Accessed 2025-06-03.
- [50] Archiki Prasad, Swarnadeep Saha, Xiang Zhou, and Mohit Bansal. ReCEval: Evaluating Reasoning Chains via Correctness and Informativeness. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 10066–10086, Singapore, December 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.emnlp-main.622.
- [51] Friedrich Pukelsheim. The Three Sigma Rule. *The American Statistician*, 48(2):88–91, 1994. doi:10.1080/00031305.1994.10476030.
- [52] Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking Agentic Workflow Generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL: <https://openreview.net/forum?id=vunPX0Fmoi>.
- [53] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024. URL: <https://openreview.net/forum?id=dHng200Jjr>.
- [54] Juan Qiu, Qingfeng Du, Kanglin Yin, Shuang-Li Zhang, and Chongshu Qian. A Causality Mining and Knowledge Graph Based Method of Root Cause Diagnosis for Performance Anomaly in Cloud Applications. *Applied Sciences*, 10(6), 2020. doi:10.3390/app10062166.
- [55] Siyu Qiu, Muzhi Wang, Raheel Afsharmazayejani, Mohammad Moradi Shahmiri, Benjamin Tan, and Hammond Pearce. Towards LLM-based Root Cause Analysis of Hardware Design Failures. In *IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–6, 2025. doi:10.1109/COINS65080.2025.11125748.
- [56] Evelien Riddell. Online Repository and Supplementary Material, 2026. URL: <https://github.com/boerste/rca-llm-reasoning>.

- [57] Carl Martin Rosenberg and Leon Moonen. Spectrum-Based Log Diagnosis. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, New York, NY, USA, 2020. Association for Computing Machinery. [doi:10.1145/3382494.3410684](https://doi.org/10.1145/3382494.3410684).
- [58] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. Exploring LLM-Based Agents for Root Cause Analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*, page 208–219, New York, NY, USA, 2024. Association for Computing Machinery. [doi:10.1145/3663529.3663841](https://doi.org/10.1145/3663529.3663841).
- [59] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. AutoTSG: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, page 1477–1488, New York, NY, USA, 2022. Association for Computing Machinery. [doi:10.1145/3540250.3558958](https://doi.org/10.1145/3540250.3558958).
- [60] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023.
- [61] Julien Siebert. Applications of statistical causal inference in software engineering. *Information and Software Technology*, 159:107198, 2023. [doi:10.1016/j.infsof.2023.107198](https://doi.org/10.1016/j.infsof.2023.107198).
- [62] Jacopo Soldani and Antonio Brogi. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.*, 55(3), February 2022. [doi:10.1145/3501297](https://doi.org/10.1145/3501297).
- [63] Cindy Sridharan. *Distributed Systems Observability*. O’Reilly Media, Inc., 2018.
- [64] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, Dan Pei, Xiao Yang, and Li Yu. LogKG: Log Failure Diagnosis Through Knowledge Graph. *IEEE Transactions on Services Computing*, 16(5):3493–3507, 2023. [doi:10.1109/TSC.2023.3293890](https://doi.org/10.1109/TSC.2023.3293890).

- [65] Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung yi Lee, and Yun-Nung Chen. Let Me Speak Freely? A Study on the Impact of Format Restrictions on Performance of Large Language Models, 2024. [arXiv:2408.02442](https://arxiv.org/abs/2408.02442).
- [66] Dongjie Wang, Zhengzhang Chen, Jingchao Ni, Liang Tong, Zheng Wang, Yanjie Fu, and Haifeng Chen. Interdependent Causal Networks for Root Cause Localization. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, page 5051–5060, New York, NY, USA, 2023. Association for Computing Machinery. [doi:10.1145/3580305.3599849](https://doi.org/10.1145/3580305.3599849).
- [67] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can Language Models Solve Graph Problems in Natural Language? In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 30840–30861. Curran Associates, Inc., 2023.
- [68] Jiaan Wang, Yunlong Liang, Fandong Meng, Zengkui Sun, Haoxiang Shi, Zhixu Li, Jinan Xu, Jianfeng Qu, and Jie Zhou. Is ChatGPT a Good NLG Evaluator? A Preliminary Study. In Yue Dong, Wen Xiao, Lu Wang, Fei Liu, and Giuseppe Carenini, editors, *Proceedings of the 4th New Frontiers in Summarization Workshop*, pages 1–11, Singapore, December 2023. Association for Computational Linguistics. [doi:10.18653/v1/2023.newsum-1.1](https://doi.org/10.18653/v1/2023.newsum-1.1).
- [69] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the Association for Computational Linguistics (ACL)*, volume 1, pages 2609–2634, Toronto, Canada, July 2023. Association for Computational Linguistics. [doi:10.18653/v1/2023.acl-long.147](https://doi.org/10.18653/v1/2023.acl-long.147).
- [70] 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)*, pages 142–150, 2020. [doi:10.1109/ICWS49710.2020.00026](https://doi.org/10.1109/ICWS49710.2020.00026).
- [71] Tingting Wang and Guilin Qi. A Comprehensive Survey on Root Cause Analysis in (Micro) Services: Methodologies, Challenges, and Trends, 2024. [arXiv:2408.00803](https://arxiv.org/abs/2408.00803).
- [72] Yidan Wang, Zhouruixing Zhu, Qiurai Fu, Yuchi Ma, and Pinjia He. MRCA: Metric-level Root Cause Analysis for Microservices via Multi-Modal Data. In *Proceedings*

- of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), page 1057–1068, New York, NY, USA, 2024. Association for Computing Machinery. [doi:10.1145/3691620.3695485](https://doi.org/10.1145/3691620.3695485).
- [73] 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)*, page 4966–4974, New York, NY, USA, 2024. Association for Computing Machinery. [doi:10.1145/3627673.3680016](https://doi.org/10.1145/3627673.3680016).
- [74] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.
- [75] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. [doi:10.2307/3001968](https://doi.org/10.2307/3001968).
- [76] Xue Wu and Kostas Tsioutsoulis. Thinking with Knowledge Graphs: Enhancing LLM Reasoning Through Structured Data, 2024. [arXiv:2412.10654](https://arxiv.org/abs/2412.10654).
- [77] Shuaiyu Xie, Jian Wang, Hanbin He, Zhihao Wang, Yuqi Zhao, Neng Zhang, and Bing Li. TVDiag: A Task-oriented and View-invariant Failure Diagnosis Framework with Multimodal Data, 2025. [arXiv:2407.19711](https://arxiv.org/abs/2407.19711).
- [78] Ruyue Xin, Peng Chen, and Zhiming Zhao. CausalRCA: Causal inference based precise fine-grained root cause localization for microservice applications. *Journal of Systems and Software*, 203:111724, 2023. [doi:10.1016/j.jss.2023.111724](https://doi.org/10.1016/j.jss.2023.111724).
- [79] 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 (ICLR)*, 2025. URL: <https://openreview.net/forum?id=M4qNizQYpd>.
- [80] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang,

- Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report, 2025. [arXiv:2505.09388](https://arxiv.org/abs/2505.09388).
- [81] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL: https://openreview.net/forum?id=WE_vluYUL-X.
- [82] Zhenhe Yao, Changhua Pei, Wenxiao Chen, Hanzhang Wang, Liangfei Su, Huai Jiang, Zhe Xie, Xiaohui Nie, and Dan Pei. Chain-of-Event: Interpretable Root Cause Analysis for Microservices through Automatically Learning Weighted Event Causal Graph. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*, page 50–61, New York, NY, USA, 2024. Association for Computing Machinery. [doi:10.1145/3663529.3663827](https://doi.org/10.1145/3663529.3663827).
- [83] Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. Justice or Prejudice? Quantifying Biases in LLM-as-a-Judge. In *International Conference on Learning Representations (ICLR)*, 2025. URL: <https://openreview.net/forum?id=3GTtZFiajM>.
- [84] Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. Language is All a Graph Needs. In Yvette Graham and Matthew Purver, editors, *Findings of the Association for Computational Linguistics (EACL)*, pages 1955–1973, St. Julian’s, Malta, March 2024. Association for Computational Linguistics.
- [85] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *Proceedings of the Web Conference (WWW)*, page 3087–3098, New York, NY, USA, 2021. Association for Computing Machinery. [doi:10.1145/3442381.3449905](https://doi.org/10.1145/3442381.3449905).
- [86] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 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)*, page 553–565, New York, NY, USA, 2023. Association for Computing Machinery. [doi:10.1145/3611643.3616249](https://doi.org/10.1145/3611643.3616249).
- [87] Chenxi Zhang, Xin Peng, Tong Zhou, Chaofeng Sha, Zhenghui Yan, Yiru Chen, and Hong Yang. TraceCRL: contrastive representation learning for microservice trace analysis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, page 1221–1232, New York, NY, USA, 2022. Association for Computing Machinery. [doi:10.1145/3540250.3549146](https://doi.org/10.1145/3540250.3549146).
- [88] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibao Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, Dai Zhang, and Dan Zhu, Zhenyu andlas Pei. Robust Failure Diagnosis of Microservice System Through Multimodal Data. *IEEE Transactions on Services Computing*, 16(6):3851–3864, 2023. [doi:10.1109/TSC.2023.3290018](https://doi.org/10.1109/TSC.2023.3290018).
- [89] Shenglin Zhang, Sibao Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, and Dan Pei. Failure diagnosis in microservice systems: A comprehensive survey and analysis. *ACM Trans. Softw. Eng. Methodol.*, 35(1), December 2025. [doi:10.1145/3715005](https://doi.org/10.1145/3715005).
- [90] Shenglin Zhang, Yongxin Zhao, Sibao Xia, Shirui Wei, Yongqian Sun, Chenyu Zhao, Shiyu Ma, Junhua Kuang, Bolin Zhu, Lemeng Pan, Yicheng Guo, and Dan Pei. No More Data Silos: Unified Microservice Failure Diagnosis With Temporal Knowledge Graph. *IEEE Transactions on Services Computing*, 17(6):4013–4026, 2024. [doi:10.1109/TSC.2024.3489444](https://doi.org/10.1109/TSC.2024.3489444).
- [91] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT. In *International Conference on Learning Representations (ICLR)*, 2020. URL: <https://openreview.net/forum?id=SkeHuCVFDr>.
- [92] Wei Zhang, Hongcheng Guo, Jian Yang, Zhoujin Tian, Yi Zhang, Yan Chaoran, Zhoujun Li, Tongliang Li, Xu Shi, Liangfan Zheng, and Bo Zhang. mABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics (EMNLP)*, pages 4017–4033, Miami, Florida, USA, November 2024. Association for Computational Linguistics. [doi:10.18653/v1/2024.findings-emnlp.232](https://doi.org/10.18653/v1/2024.findings-emnlp.232).

- [93] Xiao Zhang, Qi Wang, Mingyi Li, Yuan Yuan, Mengbai Xiao, Fuzhen Zhuang, and Dongxiao Yu. TAMO:Fine-Grained Root Cause Analysis via Tool-Assisted LLM Agent With Multi-Modality Observation Data in Cloud-Native Systems. *IEEE Transactions on Services Computing*, 18(6):4221–4233, 2025. doi:[10.1109/TSC.2025.3629066](https://doi.org/10.1109/TSC.2025.3629066).
- [94] Xu Zhang, Chao Du, Yifan Li, Yong Xu, Hongyu Zhang, Si Qin, Ze Li, Qingwei Lin, Yingnong Dang, Andrew Zhou, Saravanakumar Rajmohan, and Dongmei Zhang. HALO: Hierarchy-aware Fault Localization for Cloud Systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, page 3948–3958, New York, NY, USA, 2021. Association for Computing Machinery. doi:[10.1145/3447548.3467190](https://doi.org/10.1145/3447548.3467190).
- [95] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, Murali Chintalapati, Saravanakumar Rajmohan, and Dongmei Zhang. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, page 1253–1263, New York, NY, USA, 2021. Association for Computing Machinery. doi:[10.1145/3468264.3473919](https://doi.org/10.1145/3468264.3473919).
- [96] Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. 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)*, page 266–277, New York, NY, USA, 2024. Association for Computing Machinery. doi:[10.1145/3663529.3663846](https://doi.org/10.1145/3663529.3663846).
- [97] Ziming Zhao, Tiehua Zhang, Zhishu Shen, Hai Dong, Xingjun Ma, Xianhui Liu, and Yun Yang. CHASE: A Causal Heterogeneous Graph based Framework for Root Cause Analysis in Multimodal Microservice Systems, 2024. arXiv:[2406.19711](https://arxiv.org/abs/2406.19711).
- [98] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. MULAN: Multi-modal Causal Structure Learning and Root Cause Analysis for Microservice Systems. In *Proceedings of the ACM Web Conference (WWW)*, page 4107–4116, New York, NY, USA, 2024. Association for Computing Machinery. doi:[10.1145/3589334.3645442](https://doi.org/10.1145/3589334.3645442).
- [99] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In

A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 46595–46623. Curran Associates, Inc., 2023.

APPENDICES

Appendix A

Additional Data

In this appendix, we provide supplementary material and additional data to support the experiments. All code, experimental results, and supporting documents for this thesis can be found in our online repository [56].

A.1 Dataset Details

To ensure that fault records used in our analysis were temporally sound and supported by multi-modal telemetry, we applied a sequence of filtering steps to \mathcal{A} . Starting from 16,205 injected fault records, we first removed temporally overlapping injections by discarding any fault whose duration intersected the preceding one or whose inter-fault gap was shorter than 45 seconds, yielding 12,522 (77.3%) non-overlapping faults. We then identified extended

Table A.1: Distribution of Injected Faults for Dataset \mathcal{A}

Fault Type	Count	Injection Locations	Fault Granularity
High Memory Usage	40	10	Service instance
Session Timeout	40	2	Service instance
File Missing	36	2	Service instance
Unexpected Process Termination	20	4	Service instance
Internal Permission Misconfiguration	16	2	Service instance

Table A.2: Distribution of Injected Faults for Dataset \mathcal{B}

Fault Type	Count	Injection Locations	Fault Granularity
Container CPU Load	13	12	Service, service instance
Container Memory Load	13	11	Service, service instance
Container Packet Corruption	13	12	Service, service instance
Container Packet Retransmission	13	12	Service, service instance
Container Packet Loss	8	7	Service, service instance
Container Network Latency	8	8	Service, service instance
Container Process Termination	7	6	Service, service instance
Container Read I/O Load	17	12	Service, service instance
Container Write I/O Load	5	5	Service, service instance
Node Disk Write I/O	10	3	Host
Node Disk Read I/O	9	6	Host
Node Disk Space Consumption	10	6	Host
Node Memory Consumption	9	5	Host
Node CPU Load	9	4	Host
Node CPU Spike	4	3	Host

telemetry gaps—periods with more than 30 minutes of missing data—and excluded fault records whose duration overlapped these gaps, resulting in 4,343 (26.8%) fault records.

Because the two datasets differ substantially in size and class composition, we next balanced \mathcal{A} so that its final scale approximated that of \mathcal{B} and no single fault class dominated. To alleviate sparsity among minority classes, we minimally augmented rare faults by introducing time-shifted duplicates placed in non-overlapping synthetic windows. We then downsampled the majority classes, producing a final \mathcal{A} dataset that combines original minority-class samples, their augmented variants, and a controlled subset of majority-class faults. This yielded 152 faults in total (0.94% of the original; 3.5% of the filtered set), as reported in Table 4.1. No additional balancing was required for \mathcal{B} , since the dataset was already balanced by the authors of [79]. The final per-fault-type distributions for \mathcal{A} and \mathcal{B} are summarized in Tables A.1 and A.2.

A.2 System Knowledge Graph Entity Specifications

Table A.3 lists the entity specifications used for our benchmark datasets.

Table A.3: High-level System Entity Types

Category	Entity Types	Description	Examples
Software	Service	An aggregation of software that satisfies an end-use function.	(Micro)service, API
	Database	An organized persistent collection of data and information that allows for its retrieval.	MySQL, key-value store, file
	Cache	An organized non-persistent collection of data and information that allows for its retrieval.	Redis
	Coordination Manager	Manages metadata, state synchronization, and coordination tasks.	ZooKeeper, Consul
	Software Component Instance	A specific and identifiable runtime execution of the <i>Software Component</i> (i.e., service, database, cache).	Service instance, database instance, cache instance
Infrastructure	Host	A virtual or physical computer, hardware device, or environment where software components or programs are deployed, installed, or executed.	Virtual machine, container, server

A.2.1 Reasoning Failure Taxonomy

Figure A.1 provides a visual representation of the reasoning failure taxonomy. Columns partition failure *categories* and horizontal swimlanes indicate the *scope* in which the failure manifests: per-hypothesis (local errors during evaluation of a single hypothesis), full-trace (errors that emerge across the agent’s whole reasoning trajectory), and cross-cutting (systemic biases or errors that pervade multiple stages). Reasoning failure (RF) codes correspond to those introduced in Table 3.2.

A.3 Results

A.3.1 Accuracy

To complement the results reported in Table 5.1 and the aggregated visualization in Figure 5.1 of the main text (Section 5.1), Figure A.2 presents per-dataset accuracy plots.

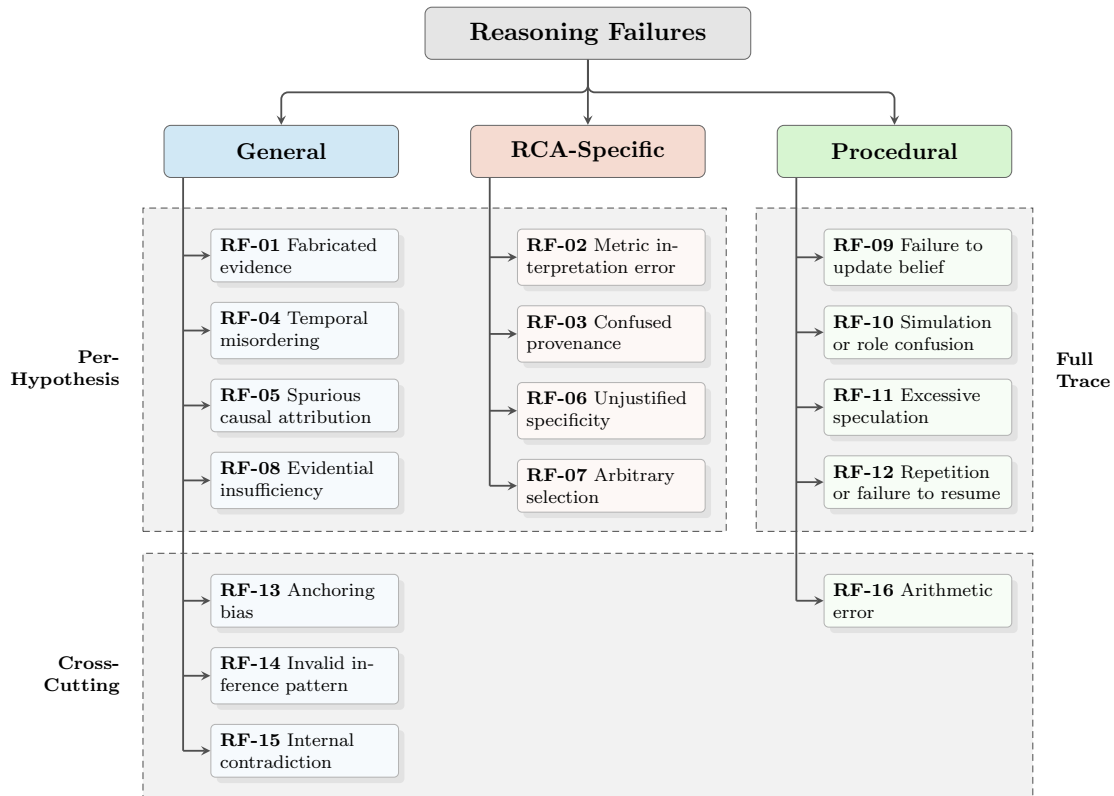


Figure A.1: Reasoning failures taxonomy. Columns partition failure *categories* and horizontal swimlanes indicate the *scope* in which the failure manifests.

A.3.2 LLM Inference Times

To provide context on the operational cost of using different workflows, we analyze the inference time (i.e., time-to-result) for all evaluated models. Table A.4 details the average inference time per scenario in minutes across workflows. The percent change ($\% \Delta$) is calculated relative to the *Straight-Shot* baseline. Our experiments show that REACT and *Plan-and-Execute* increased average inference time by 40% and 354%, respectively, compared to the *Straight-Shot* baseline.

A.3.3 Sensitivity to Input Representation Strategies

This section summarizes the input-representation strategies we evaluated and highlights the key empirical trends. We examine two orthogonal aspects of input construction: (1) the

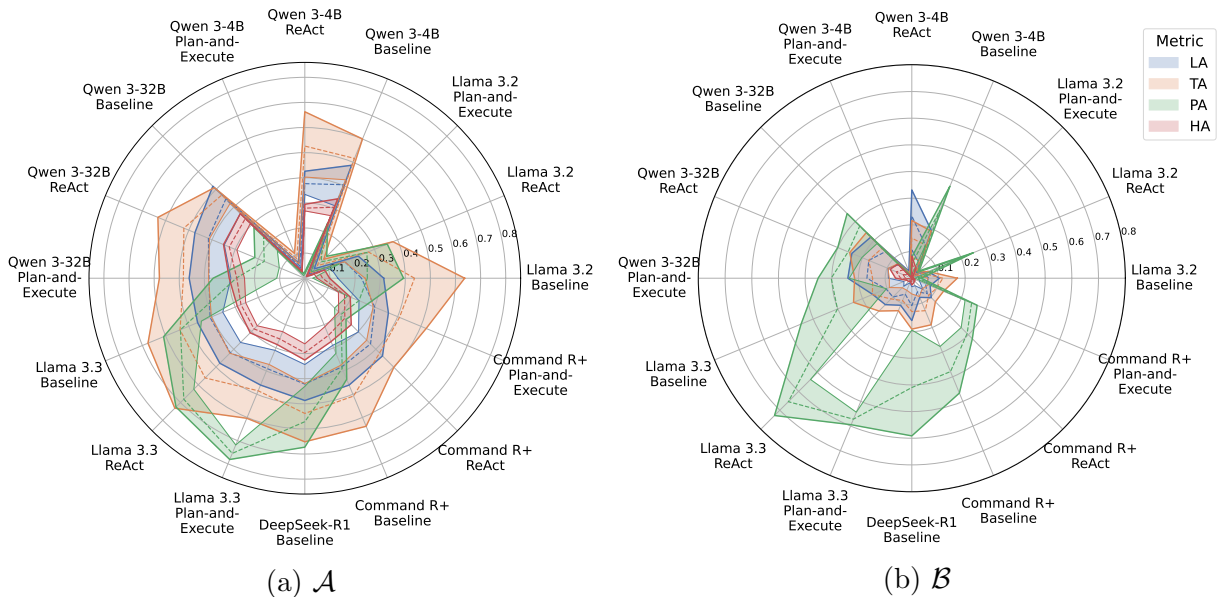


Figure A.2: Accuracy results for datasets (a) \mathcal{A} and (b) \mathcal{B} (lower bound is A@1, upper bound is A@3, and dotted line is Avg@3).

alert unification strategy and (2) the *KG representation strategy*. For (1), we compare *time-based* unification, which preserves the chronological sequence of alerts, and *element-based* unification, which groups alerts by their corresponding KG element. For (2), we compare a *list-based* representation with a *JSON-based* representation of nodes and edges.

Accuracy differences between strategies are reported as Δ A-Avg@3 in Figure A.3. Statistical significance is assessed using the Wilcoxon signed-rank test [75]. In both figures, solid bars denote changes significant at $p < 0.05$, while faded bars denote non-significant differences.

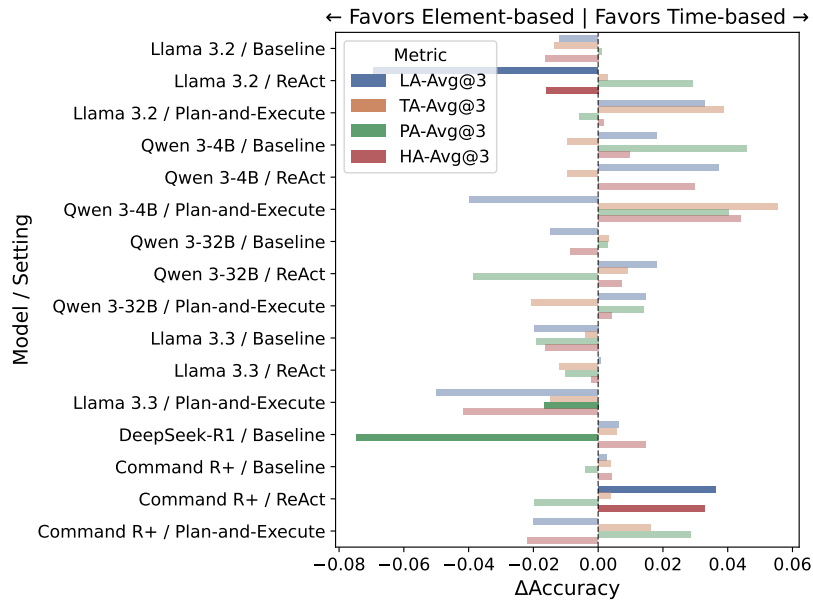
Figure A.3a shows that the optimal alert unification strategy varies substantially by model and workflow, and that significant effects are relatively uncommon. Under REACT, Llama 3.2 displays a statistically significant preference for *element-based* unification (negative Δ). Conversely, Command R+ favours *time-based* unification (positive Δ), suggesting that it benefits more from preserving temporal proximity when inferring causality. Notably, DeepSeek-R1 exhibits a preference for *element-based* grouping for PA (Δ PA ≈ -0.07), indicating that grouping alerts by system topology aids in reconstructing fault-propagation paths. Overall, the mixed trends suggest that the efficacy of a unification strategy reflects underlying training biases—temporal versus categorical structuring—rather than a universally optimal format.

Table A.4: Average LLM Inference Time Per Scenario

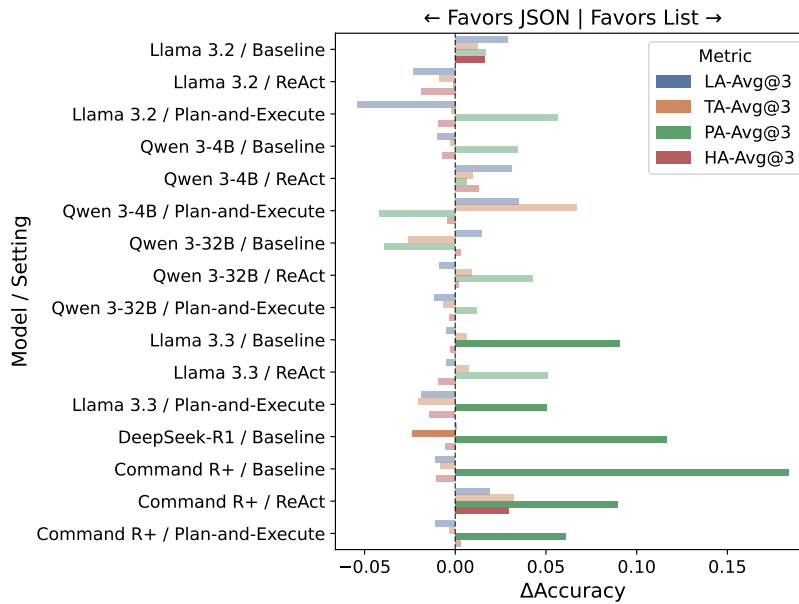
Model	<i>Straight-Shot</i> <i>min</i>	ReAct <i>min (%Δ)</i>	<i>Plan-and-Execute</i> <i>min (%Δ)</i>
Llama 3.2 (3B)	0.76	1.8 (+139.1%)	8.3 (+987.0%)
Qwen 3 (4B)	3.2	4.8 (+49.1%)	6.1 (+88.2%)
Qwen 3 (32B)	5.0	5.0 (+0.9%)	10.8 (+116.3%)
Llama 3.3	6.0	5.7 (-4.9%)	10.7 (+78.3%)
DeepSeek-R1	4.3	N/A	N/A
Command R+	4.6	5.4 (+17.6%)	27.3 (+499.5%)

Time in minutes (*min*); parentheses show percent change ($\% \Delta$) relative to *Straight-Shot*.

The results in Figure A.3b reveal a strong, statistically significant preference for *list-based* KG representations for Llama 3.3, DeepSeek-R1, and Command R+ for inferring failure-propagation paths. These models exhibit large positive gains for PA (e.g., Command R+ $\Delta PA > 0.15$), indicating that the concise, token-efficient structure of adjacency lists is substantially more suitable for traversal-style reasoning than the more verbose JSON format. However, this advantage in pathfinding does not consistently translate to other metrics, where results are mixed or negligible. Taken together, the findings suggest that while list-based KGs substantially aid models in “walking” the graph, they do not consistently improve the model’s ability to recover non-topological attributes of the root-cause fault.



(a) *Alert unification* strategies



(b) *KG representation* strategies

Figure A.3: Change in accuracy ($\Delta A\text{-Avg@3}$) for (a) *alert unification* and (b) *KG representation* strategies. In both figures, **solid bars** denote statistically significant differences ($p < 0.05$) according to the Wilcoxon signed-rank test, while **faded bars** indicate changes that are not statistically significant.

A.4 Prompts

Figure A.4 shows the RCA input-prompt template we used for the REACT workflow. Figure A.5 contains the full LLM-as-a-Judge prompt template, including the failure taxonomy and the step-by-step annotation guidelines provided to the judge model.

System Message

You are a helpful and rigorous assistant who is an expert in root cause analysis for complex cloud-based software systems.

SYSTEM OVERVIEW

Consider a cloud-based software system composed of multiple interconnected components (both software and hardware). Errors or issues originating in one component may propagate to others due to dependencies, communication links, or shared resources. These errors often manifest as observable symptoms (e.g., anomalies or alerts) in different system components.

The system is represented by an explicit, typed, directed knowledge graph, where:

- Nodes represent system components or entities
- Edges indicate relationships between them

KNOWLEDGE GRAPH

Entity types

{*entity schema*}

Relationship Types

{*relationship schema*}

Note: The schema defines *abstract* entity/relationship types and therefore only serves as a *guide*. The actual knowledge graph instantiates these as specific nodes and edges, which you can access exclusively through the provided tools. You have access to the following tools: {*tools by name*}.

OBSERVED SYMPTOMS

You will be provided a set of observed symptoms/alerts detected by an anomaly detector, ordered {*alert representation strategy*}. Metric and trace alerts are reported based on the first observed anomalous value, not the full duration of the abnormal behavior. For example, if a metric remains abnormal for several seconds, only the initial timestamp is included.

TASK

Use the knowledge graph and the provided observed alerts/symptoms to identify the **three most likely root cause faults** that could explain the symptoms.

Each fault:

- Must be localized to a single system component (node in the graph) of type {*root cause fault entity types*}.
- Must be restricted to the fault types listed in the INSTRUCTIONS section below.
- Should include a plausible propagation path through the system that justifies how the fault led to the observed symptoms. There may be multiple plausible propagation paths. You may select the most likely or explanatory one, but must justify your choice clearly and refer to relationships in the knowledge graph.
- Must be well-justified using explicit reasoning through the graph.

INSTRUCTIONS:

You should think step-by-step in order to fulfill the objective. The step-by-step workflow should follow a “Thought/Action/Observation” loop that can repeat multiple times if needed. Here is how you should go about it:

1. **Thought:** reflect internally on the current task, the available information, and what to do next.

2. **Action:** if further information is needed, choose one appropriate tool to call. Any and all "Thoughts" must be included in the 'reasoning' field in the tool input. Output only the tool call object and do not include any other text.

3. **Observation:** The tool will return a result, which will be provided to you.

Repeat this loop as needed until you have enough information to answer the original task.

When ready, output your final answer starting with the prefix "Final Answer:".

Rules: do NOT write "Thought:" or "Action:" separately. All reasoning must go inside the 'reasoning' field of the tool input, only one tool can be called per step.

Your "Final Answer" should consist of ALL THREE likely root cause faults.

For each root cause fault, provide:

- **Type:** one of: *{root cause fault types}*
- **Description:** an explanation of what the fault looks like in the system.
- **Location:** the *exact* node at which the fault occurs; must be of type: *{root cause fault entity types}*
- **Justification:** a step-by-step reasoning based on the alerts and knowledge graph. Reference any relevant alerts.
- **Propagation path:** a plausible propagation path in the knowledge graph that would make the root cause possible, formatted as `node1 --(edge_label1)-->node2 --(edge_label2)-->node3`, using only nodes and edge labels present in the knowledge graph.

Your output should follow the field structure in the same order (Type, Description, Location, Justification, Propagation Path).

Rank faults in order of most likely to least likely.

Human Message

OBSERVED SYMPTOMS

The following symptoms/alerts were detected by an anomaly detector, ordered by *{alert representation strategy}*.

{alerts}

Think step by step and ensure your reasoning is traceable through the knowledge graph.

CLARIFICATIONS:

- **Alert Coverage:** The alerts represent the *full* set of detected anomalies. Some system components may lack observability, so absence of alerts does not imply no involvement in fault propagation.
- **Alert Extraction:**
 - *Metric alerts:* Detected via 3-sigma rule
 - *Trace alerts:* Detected via isolation forest
 - *Log alerts:* Extracted using log templates (i.e., rule-based)
- **Log Alerts:** Preprocessed to prioritize errors and infrequent logs. Quantity does not always correlate with importance—some critical issues may occur only once in the logs.
- **Trace Alerts:**
 - *PD* = Performance Degradation, indicating increased API latency and degraded system performance.
 - *400/500* = 400-level and 500-level error codes that occurred during the communication between two entities.

Figure A.4: Input prompt template for REACT workflow.

System Message

You are a rigorous assistant with excellent critical thinking skills. Your task is to qualitatively analyze the reasoning of an LLM agent in a root cause analysis task and identify any reasoning failures according to a given taxonomy. Work through the annotation workflow step by step. Only mark failures you are reasonably confident in.

Human Message

REASONING FAILURE TAXONOMY

RF-01 — Fabricated evidence (hallucinated alerts/metrics/logs/traces)

Scope: per-hypothesis (evidence existence)

Definition: Model asserts the existence of a specific alert/metric/log/trace that cannot be found in the provided alerts/metrics/logs/traces after up to 3 quick scans.

Example: Claims 2025-09-01 12:05 | METRIC | dbservice1 | disk_io | up or “The disk_io metric was up for dbservice”, but no such record (or any reasonably equivalent entry for that alert) exists in the provided alerts.

Signals: Model quotes an alert absent in the alert set; model uses confident language about a concrete alert that cannot be located.

Annotation rule: Perform up to 3 quick scans (exact or close fuzzy match on component + metric/endpoint + time) for each piece of evidence mentioned. If no match found, mark RF-01 and paste model claim + NO MATCH FOUND.

Severity: 1-5

- 1 = single fabricated alert/metric/log/trace in 1 hypothesis
- 2 = multiple occurrences in 1 hypothesis
- 3 = single occurrence in 2 hypotheses
- 4 = multiple occurrences in 2 hypotheses
- 5 = present in all 3+ hypotheses

RF-02 — Metric-interpretation error (directionality / semantic misread)

Scope: per-hypothesis (evidence interpretation)

Definition: Misunderstands/misinterprets metric semantics (up $\implies +3\sigma$, down $\implies -3\sigma$), confuses counters/gauges, or inverts meaning.

Example: “docker_memory_rss_pct is down, indicating high memory usage.” (direction inverted: down for this metric means memory measure decreased); “mem_usage is down, indicating a memory leak” (misinterpretation: a memory leak would typically correlate with increased memory usage).

Signals: Interpretation directly contradicts the standard metric meaning or contradicts $\pm 3\sigma$ (i.e., up/down) semantics for the metric.

Annotation rule: If the alert exists but interpretation contradicts metric semantics \rightarrow label RF-02 and paste the model claim(s) + alert(s) it referenced.

Severity: 1-5

- 1 = single metric misinterpretation in 1 hypothesis
- 2 = multiple occurrences in 1 hypothesis
- 3 = single occurrence in 2 hypotheses
- 4 = multiple occurrences in 2 hypotheses
- 5 = present in all 3+ hypotheses

RF-03 — Confused provenance (symptom-observer blamed as cause)

Scope: per-hypothesis (provenance)

Definition: Model treats the component that observed/logged a symptom as the origin/root cause rather than tracing upstream/downstream sources.

Example: Webservice log contains “an error occurred in a downstream service”; model concludes “webservice is root cause” instead of investigating downstream services.

Signals: Log text contains explicit downstream/propagation language; model names observer as cause.

Annotation rule: If evidence indicates an observed downstream symptom and model blames the observer, mark RF-03.

Severity: 1-5

- 1 = single confused provenance instance in 1 hypothesis
- 2 = multiple occurrences in 1 hypothesis
- 3 = single occurrence in 2 hypotheses
- 4 = multiple occurrences in 2 hypotheses

- 5 = present in all 3+ hypotheses

RF-04 — Temporal misordering (timeline error)

Scope: per-hypothesis (timestamps)

Definition: Model assigns causation to an event occurring after the observed effect or otherwise violates alert timestamp ordering.

Example: A log says background save (BGSAVE) started at 12:20, but multiple memory/I/O anomalies began at 12:15; model claims BGSAVE caused the earlier anomalies.

Signals: Claimed cause timestamp is later than the earliest effect timestamp.

Annotation rule: Extract model-cited timestamps and compare to alerts; if the causal claim violates the real timeline, mark RF-04. If ordering is implied and contradicts alert order, still mark RF-04.

Severity: 1-5

- 1 = single temporal misordering in 1 hypothesis

- 2 = multiple occurrences in 1 hypothesis

- 3 = single occurrence in 2 hypotheses

- 4 = multiple occurrences in 2 hypotheses

- 5 = present in all 3+ hypotheses

RF-05 — Spurious causal attribution (weak/unsupported causation OR mechanism depends on nonexistent KG link)

Scope: per-hypothesis (mechanism/causal chain)

Definition: Model asserts $X \rightarrow Y$ causation without adequate support, or uses a causal mechanism that depends on knowledge-graph relationships that do not exist. Plausible speculation consistent with the KG and alerts is acceptable and should not be penalized. KG relationships that are close-enough without detracting from the point being made OR aligned more closely with trace alerts should also not be penalized (e.g., `webservice1 --(instance_of)--> webservice --(control_flow)--> redisservice --(has_instance)--> redisservice2` vs `webservice1 -(control_flow)--> redisservice2`)

Example: Claims node-6 disk writes cause shippingservice-0 latency because `node-6 --(hosts)--> shippingservice-0`, but that host relationship is nonexistent in the KG.

Signals: Use of causal language (“caused”, “because of”, “therefore”) plus no plausible KG/alert support, or explicit citation of nonexistent KG edges (based on the information available).

Annotation rule: Check KG & alerts for the mechanism; if mechanism unsupported or relies on absent KG links, mark RF-05.

Severity: 1-5

- 1 = single spurious causal attribution in 1 hypothesis

- 2 = multiple occurrences in 1 hypothesis

- 3 = single occurrence in 2 hypotheses

- 4 = multiple occurrences in 2 hypotheses

- 5 = present in all 3+ hypotheses

RF-06 — Unjustified instance/granularity specificity

Scope: per-hypothesis (granularity)

Definition: Model asserts an instance-level root-cause location when evidence supports only service-/node-level effect, unless unique per-instance multi-modal evidence exists.

Example: Service-wide alerts, but model blames `loginservice2` instance without unique evidence.

Signals: No unique instance differentiator (multi-modal alerts, unique timestamps, volume, frequency).

Annotation rule: If instance claim lacks per-instance unique evidence, mark RF-06. Only relevant for cases where the root-cause location can be more than only instance-level.

Severity: 1-5

- 1 = present for 1 hypothesis

- 3 = present for 2 hypotheses

- 5 = present in all 3 hypotheses

RF-07 — Arbitrary / non-systematic evidence selection (bad triage)

Scope: per-hypothesis (evidence selection)

Definition: Model focuses on a seemingly arbitrary alert subset inconsistent with simple triage heuristics (i.e., first-seen, highest-frequency, highest-volume, multi-modal alerts vs single-modal alerts) without rationale, and selection plausibly alters diagnostic trajectory/conclusions.

Example: Investigates loginservice2 though loginservice1 has identical metric/trace alerts earlier in time; or investigates mobservice1 while dbservice2 has more metric alerts.

Signals: Chosen evidence is seemingly arbitrary and does not follow logical selection procedures: earliest/most frequent/highest volume multi-modal.

Annotation rule: If model's chosen evidence subset is plausibly arbitrary or contradicts simple triage heuristics and that choice affected the top hypothesis or multiple hypotheses → mark RF-07 (higher severity). If it did not materially change outcome → mark with low severity.

Severity: 1-5

- 1 = present for 1 hypothesis
- 3 = present for 2 hypotheses
- 5 = present in all 3 hypotheses

RF-08 — Evidential insufficiency (supported but weak / non-specific for the claim)

Scope: per-hypothesis (evidence sufficiency)

Definition: Evidence exists but its temporal precision, frequency, mechanism link, discriminability, provenance clarity, or granularity is insufficient to support the specific claim. This is an inferential-sufficiency error, not a hallucination.

Signals: Checklist failures: temporal precedence; frequency; mechanism; discriminability; provenance clarity; granularity alignment.

Annotation rule: After prior per-hypothesis checks, apply Sufficiency Checklist; if required items fail for claim type, mark RF-08 and provide model claim(s) + matched alert(s) + list which checklist items failed.

Severity: 1-5

- 1 = single insufficiency in 1 hypothesis
- 2 = multiple occurrences in 1 hypothesis
- 3 = single occurrence in 2 hypotheses
- 4 = multiple occurrences in 2 hypotheses
- 5 = present in all 3+ hypotheses

RF-09 — Failure to update belief (non-monotonic updating error)

Scope: full-history

Definition: Model does not revise or retract a previous claim after later evidence or Tool Message contradicts it. This is specifically a failure to update in light of new evidence (distinct from anchoring, which is a failure to explore alternatives).

Example: Claims Redis eviction; later Tool Message shows normal Redis; final answer still claims eviction. Speculates webservice2 is hosted on host4 and therefore failures on host4 affect webservice2; later Tool Message shows webservice is hosted on host2; final answer still includes basis of webservice2 being hosted on host4.

Signals: Later Tool Messages/alerts contradict earlier claims, and claims persist.

Annotation rule: Extract the original claim and the contradicting evidence; if model fails to revise → mark RF-09. Severity high if final answer relies on unchanged, contradicted claim.

Severity: 1-5

- 1 = single untrue claim impacts 1 hypothesis
- 2 = multiple impact 1 hypothesis
- 3 = single impacts 2 hypotheses
- 4 = multiple impact 2 hypotheses
- 5 = untrue claim(s) impact all 3 hypotheses

RF-10 — Simulation / role confusion (pretend tool output used as factual evidence)

Scope: full-history

Definition: Model explicitly states it cannot call tools and “assumes” or “simulates” tool outputs, then treats simulated outputs as factual in final conclusions. Consider a Tool Message “real” if there is an explicit tool header like “==== Tool Message =====” (allowing variable ‘=’ counts).

Example: “I cannot call the tools; I will assume the log shows ERROR: connection refused”; final answer treats the assumed log as observed.

Signals: Phrases like: “I cannot call”, “I can’t call”, “I’ll assume”, “I will pretend”, “simulate the response”, “assume the tool returns”, followed by conclusive claims. No Tool Message corresponding to the assumed call.

Annotation rule: Consider all text prior to the final answer and search for signal phrases and Tool Messages. If simulated outputs are used as factual evidence without a Tool Message, mark RF-10. If simulated exploration was not used as final evidence or Tool Messages were later present, mark RF-10 but lower severity.

Severity: 1-5

- 1 = simulated output noted, but not used for final claims
- 2 = simulated output used for a secondary claim only
- 3 = simulated output(s) used as evidence for 1 hypothesis
- 4 = simulated output(s) across 2 hypotheses
- 5 = simulated output(s) are the core evidence for 3+ hypotheses

RF-11 — Excessive speculative / rambling reasoning (ungrounded token waste)

Scope: full-history

Definition: Considerable portion of the chat is spent being confused or speculating about the system architecture, knowledge graph, meaning semantics of the alerts, available tools, and deciding what steps to take instead of performing tool calls to confirm/refute KG characteristics or check evidence; especially when tools were available but unused.

Example: KG-schema theorizing while no Tool Messages are present to confirm/refute KG characteristics.

Signals: High density of hedging or rambling language (including “wait”); paragraphs without alert/metric/log/trace citations; round-about or circular thoughts; none-to-little Tool Message usage.

Annotation rule: Consider all text prior to the final answer and search for hedging or speculative language. If high and blocked/replaced necessary data checks OR prevented a conclusive unstructured final answer, mark RF-11.

Severity: 3-5

- 3 = heavy speculation/rambling, blocked some important checks
- 4 = very heavy speculation/rambling, significantly blocked analysis and tests
- 5 = entire session dominated by speculation/rambling, no meaningful evidence work, final answer driven by speculation

RF-12 — Repetition / failure to resume (looping across turns)

Scope: full-history

Definition: Model repeats the same planning, intro text, or deliberation across consecutive replies and fails to resume earlier progress, typically after truncation.

Example: Consecutive replies (marked by “==== AI Message =====”) begin with similar “First I will check...” paragraphs and add little new content. Consecutive replies contain similar deliberation about the semantics of a ‘down’/‘up’ metric alert.

Signals: High n-gram overlap across AI messages; or repeated planning text.

Annotation rule: If repetition caused stalled progress or omitted checks, mark RF-12.

Severity: 3-5

- 3 = repetition across multiple turns omitted some checks
- 4 = repetition stalled significant parts of the analysis
- 5 = repetition prevented completion and changed final answer

RF-13 — Anchoring / premature commitment (insufficient hypothesis exploration)

Scope: cross-cutting (search behaviour)

Definition: Model fixates early on a single hypothesis and fails to enumerate OR to explore other plausible alternatives/hypotheses (e.g., component or type of fault).

Example: Model immediately focuses on “high memory usage” as the cause and never lists or considers other plausible causes (e.g., network, disk) despite relevant alerts. Model claims it should explore host relationships for loginservice2, webservice1, and dbservice2; calls thetool for loginservice2; does not follow through for webservice1 and dbservice2.

Signals: < 2 reasonable alternatives (w.r.t. component or fault type) listed across chat; no follow-through on planned exploration without good rationale.

Annotation rule: If the model provides fewer than 2 reasonable alternative hypotheses and goes straight to a definitive cause, mark RF-13. Only consider RF-13 if there was an opportunity to explore (i.e., through tools, using the reasoning fields, think tags <think></think>, etc.).

Severity: 3-5

- 3 = some diversity in planned exploration and some follow-through
- 4 = some diversity in planned exploration but no follow-through
- 5 = anchoring dominated the analysis and impacted all hypotheses

RF-14 — Invalid logical inference patterns (formal fallacies)

Scope: cross-cutting (invalid inference)

Definition: Model applies invalid inference patterns in deriving diagnostic claims. Look for: affirming the consequent, denying the antecedent, post hoc, composition/division, ecological fallacy, hasty generalization.

Example: Model sees one trace with a 500 for endpoint /login on a single instance and concludes “the whole service is down” (hasty generalization).

Signals: Clear leap from limited premises to broad/systemic conclusion without intermediate mechanism or checks.

Annotation rule: Identify fallacy, quote the premise(s) and conclusion, mark RF-14.

Severity: 1-5

- 1 = single minor fallacy with negligible impact
- 2 = multiple minor fallacies with negligible impact
- 3 = fallacy(s) used to support 1 hypothesis
- 4 = fallacy(s) used to support 2 hypotheses
- 5 = fallacies pervasive across all 3+ hypotheses

RF-15 — Internal contradiction (explicit inconsistency)

Scope: cross-cutting

Definition: Model makes mutually incompatible statements in the chat history. This is different from RF-09: RF-15 is an explicit contradiction rather than a failure to revise.

Example: “No trace 500 errors”, then “multiple 500 trace errors”.

Signals: Pairwise contradictory sentences when compared; contradiction can be related (but not limited) to a metric, timestamp, evidence existence, component relationships, etc.

Annotation rule: Quote contradictions, mark RF-15.

Severity:

- 1 = single contradiction with negligible impact
- 2 = multiple contradictions with negligible impact
- 3 = contradiction(s) impact 1 hypothesis
- 4 = contradiction(s) impact 2 hypotheses
- 5 = contradiction(s) impact 3+ hypotheses

RF-16 — Arithmetic / aggregation mistake

Scope: cross-cutting

Definition: Numeric miscalculations or wrong aggregations that change interpretation.

Example: Reports “error rate increased 200%” when correct is 20%.

Signals: Numeric expressions in the text; automatic recomputation disagrees with reported number.

Annotation rule: Recompute numeric or aggregation claims; if inconsistent and materially affects conclusions, mark RF-16 + include corrected value. (Severity generally low unless numeric error changed final diagnosis.)

Severity: 1-5

- 1 = single small numeric mismatch with negligible impact
- 2 = multiple minor numeric mismatches with negligible impact
- 3 = numeric/aggregation error(s) that impacts 1 hypothesis
- 4 = numeric/aggregation error(s) that impact 2 hypotheses
- 5 = numeric/aggregation error(s) that impact 3+ hypothesis

TASK

Your task is to label the given reasoning by an LLM agent below according to the failure taxonomy.

ANNOTATION GUIDE – practical rules

1. Multilabel: Assign all RFs that apply.
2. RF-00 precedence: If the structured final response (JSON) is None or nan, do NOT mark RF-00. Otherwise, compare the unstructured final answer with the structured final response; if divergent, mark RF-00 and use the unstructured final answer as the basis for the rest of the annotation.
3. Tool message rule: If the chat contains a real Tool Message corresponding to the claimed tool call → treat associated evidence as real. If the model claimed a call but no Tool Message exists and the model used the assumed output as fact → consider RF-10. Consider a Tool Message “real” if there is an explicit tool header like “==== Tool Message =====” (allowing variable ‘=’ counts).
4. Document evidence: For every RF, paste the triggering model sentence(s) and the matched alert(s) or NO MATCH FOUND.
5. Severity per RF (1-5): Give severity for each RF assigned.
6. Ground-truth is context only: Do not mark RFs solely because the model disagrees with ground truth; only flag when model claims/presents evidence incorrectly relative to accessible context.

ANNOTATION WORKFLOW (step-by-step)

Step 1 — Global gate

1. Compare final structured output response vs final unstructured answer → mark RF-00 if divergent; record severity.

Step 2 — Per-hypothesis loop (for each hypothesis in the final answer, process top-1 first)

2. RF-01: quick-scan up to 3 times for EACH; if no match → RF-01 for that hypothesis → stop per-hypothesis checks for *this* hypothesis.
3. RF-02: if metric alerts used, verify semantics; if inverted/misread → RF-02.
4. RF-03: examine provenance; if model blames observer despite propagation language → RF-03.
5. RF-04: compare timestamps; if cause occurs after effect → RF-04.
6. RF-05: evaluate causal mechanism vs KG/alerts; if mechanism relies on non-existent KG edges or is otherwise unsupported/not plausible → RF-05.
7. RF-06: if model claims instance-level root cause location, verify unique per-instance multi-modal evidence; if absent → RF-06.
8. RF-07: assess whether chosen evidence selection contradicts simple triage heuristics and whether selection changed conclusions; if so → RF-07 (severity scaled by impact).
9. RF-08: apply Sufficiency Checklist (Temporal, Frequency, Mechanism, Discriminability, Provenance, Granularity); if required items fail for claim type → RF-08.

Step 3 — Full-history checks (scan all “AI Message” and “Tool Message” instances prior to the Final Answer/Response)

10. MANDATORY: perform a full history scan and search for evidence for RF-09–RF-12. Produce a compact “Full History Summary” of the LLM agent’s behaviour. Provide exact quote(s) (verbatim) to support RF-09–RF-12.
11. RF-09: scan chronological history for later Tool Messages/alerts that contradict earlier claims; if no revision → RF-09.
12. RF-10: detect simulated tool outputs used as facts without Tool Message → RF-10.
13. RF-11: measure speculative text percentage; if >30% and blocked checks → RF-11.
14. RF-12: detect repeated planning text that stalled progress → RF-12.

Step 4 — Cross-cutting checks (scan the entire chat history)

15. MANDATORY: perform an entire chat history scan and search for evidence for RF-13–RF-16.
16. RF-13: did the agent ever enumerate ≥ 2 plausible alternatives? If not → RF-13.
17. RF-14: detect formal fallacies anywhere → RF-14.
18. RF-15: find explicit contradictions elsewhere → RF-15.
19. RF-16: recompute numeric claims or aggregations across chat; if mismatches materially affect reasoning → RF-16.

Step 5 — Finalize

20. Assign all applicable RFs and record per-RF severity (1-5). For the per-hypothesis RFs, make sure the severity accurately reflects the number of occurrences across hypotheses.

21. Out of the RFs identified, list the RFs that directly affected/impacted the #1 hypothesis.

22. Output a json object (using (‘‘json) and (’’) as delimiters) with the Failures Identified Output Schema.

FAILURES IDENTIFIED OUTPUT SCHEMA

```
``json
{
  "failures_identified": [
    {
      "type": "The RF identifier, e.g. 'RF-01'",
      "model_claim": "Model claim or behaviour in the chat history that supports the RF",
      "rationale": "A description of the issue and a justification/rationale that the RF applies",
      "severity": "Severity of the RF."
    },
    ...
  ],
  "affected_top_hypothesis": "List of RFs that directly affected the #1 hypothesis, e.g., ['RF-01', 'RF-13']"
}
``
```

Below is the original root cause analysis task and the associated ‘final response’ (the LLM’s structured output), all enclosed in <begin chat history> and <end chat history>.

The ground-truth root cause for this scenario was: {*root cause location*} (location) and {*root cause type*} (type).

Work through the annotation workflow step by step.

<begin chat history>

{*chat history*}

Final response (structured output):

{*final answer*}

<end chat history>

Figure A.5: LLM-as-a-Judge prompt template with the reasoning failure taxonomy and step-by-step annotation workflow.