AIDA: Accelerating Root Cause Analysis for Multi-Vendor Device Failures with LLM-Powered Reasoning

Haoran Xu[1,2], Xuan Zeng[2], Xumiao Zhang[2], Xiaoxi Zhang[1,*], Yang Lv[2], Peng Zhang[2], Deke Guo[1], Ennan Zhai[2,*]

[1]*Sun Yat-sen University* [2]*Alibaba Cloud*

Abstract

Root cause analysis (RCA) of network device failures is critical to cloud reliability. While monitoring can identify *which* device has failed, diagnosing *why* remains a slow, manual process, increasing the risk of recurring failures and cascading service disruptions. Existing automated methods are inadequate: traditional methods lack precision, while prior machine learning (ML) and large language model (LLM) approaches are often too coarse-grained, require heavy manual configuration, or fail to produce verifiable reasoning essential for operator trust. This paper presents AIDA, the first system to deliver automated, fine-grained RCA of network device failures, deployed at scale in Alibaba Cloud's production network. AIDA's contributions include: (1) fine-tuning an LLM with reinforcement learning to distill expert logic into interpretable reasoning chains; (2) synthesizing these chains into an evolving knowledge graph (KG) to support retrieval-augmented generation (RAG); and (3) employing RAG-driven multi-step inference wherein the LLM is sequentially guided by the KG to construct robust, verifiable reasoning. Deployed for over a year, AIDA has achieved 95.4% precision with interpretable output and reduced the median RCA time from 72.6 hours to 1.6 minutes. Notably, it curtails the 90th-percentile diagnosis latency from 329.9 hours to 19.4 hours.

CCS Concepts

• **Networks** → **Network reliability**; **Network management**; • **Computing methodologies** → **Natural language generation**; **Information extraction**; **Causal reasoning and diagnostics**.

Keywords

AIOps, Root Cause Analysis, Network Device Failures, Large Language Models, Retrieval-Augmented Generation

ACM Reference Format:

Haoran Xu[1,2], Xuan Zeng[2], Xumiao Zhang[2], Xiaoxi Zhang[1,*], Yang Lv[2], Peng Zhang[2], Deke Guo[1], Ennan Zhai[2,*], [1]*Sun Yat-sen University* [2]*Alibaba Cloud* . 2026. AIDA: Accelerating Root Cause Analysis for Multi-Vendor Device Failures with LLM-Powered Reasoning. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3789240.3829126

Work completed during Haoran Xu's internship at Alibaba Cloud.

*Xiaoxi Zhang and Ennan Zhai are the co-corresponding authors.

SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829126

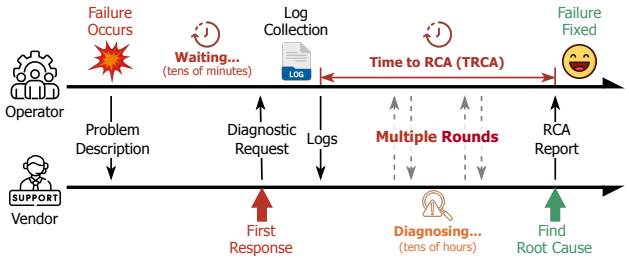


Figure 1: Typical RCA workflow for network devices.

1 Introduction

Alibaba Cloud provides continuous cloud services to millions of users globally. These services are supported by a large and growing network infrastructure, spanning tens of interconnected data centers. At this scale, network incidents pose significant threats to service reliability and operational efficiency. Leveraging state-of-the-art monitoring systems [3, 13, 28, 35, 36, 44], faulty devices can be rapidly localized and temporarily mitigated. However, this is insufficient. Ensuring long-term network stability requires root cause analysis (RCA) to accurately identify the underlying hardware or software faults, *e.g.,* port flapping due to optical module failures and BGP session instability due to software bugs. It allows a cloud provider to proactively prevent failures due to the same root cause from recurring on other devices, avoiding cascading incidents that simple device isolation may fail to avert.

However, performing RCA for device failures in a diverse, multi-vendor network environment is challenging. Root cause patterns are often vendor-proprietary and diverse, while cloud operators typically lack such specialized expertise. Consequently, current practice relies heavily on vendor technical support through a manual, iterative process involving multiple rounds of email-based communication (Figure 1). We define Time to RCA (TRCA) as the interval from initial log collection to the generation of an RCA report. Our measurements (Figure 2) show that the median TRCA ranges from tens to hundreds of hours across devices from different vendors. This leads to high operational costs from expert consultation and undermined network service (*e.g.,* congestion, packet loss, and video streaming quality degradation), as isolating faulty devices often saturates backup links during the remediation window. More critically, delayed diagnosis increases the risk of recurring failures in the production network, causing significant service disruptions and financial losses. As a large volume of interactive emails contains vendor-proprietary expertise, required by this task, a natural question arises: *why not automate RCA for network device failures by learning from historical incidents?*

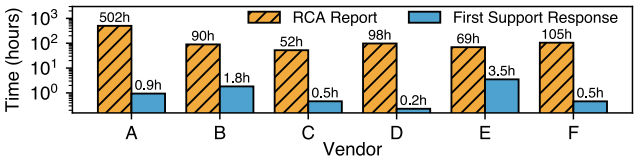


Figure 2: Median TRCA and initial vendor response time across devices from six vendors, showing that TRCA dominates failure resolution time and is the primary optimization target of AIDA.

Existing automated approaches for incident RCA are inadequate for practical operational needs. Traditional methods for *service-level* or *device-level* fault localization [3, 10, 14, 18, 43] require "white-box" system visibility, which is unavailable in our environment. More critically, they identify *which* service component or device failed, but not *why* the device malfunctioned, a problem we define as *network device RCA*. Conventional ML techniques also fall short in three aspects: (1) they struggle to adapt to rapidly evolving hardware and software failure types that change monthly, while frequently retraining models to adapt to these changes is impractical (Appendix C); (2) the long-tail distribution of failure types (Figure 3) makes model fine-tuning prone to overfitting; (3) log-based anomaly detection methods [7, 11, 33, 34, 49] struggle to synthesize heterogeneous data sources and capture complex cross-event correlation (see evidence shown in Appendix D). Recently, LLMs have been explored for reasoning over vendor-proprietary data to infer failures and their causes in third-party devices [2, 4, 32, 36, 37, 48], but still exhibit key limitations. They require substantial manual intervention (*e.g.,* handcrafted data collection rules), operate on lossy summaries that omit critical diagnostic details (see Appendix D), or fail to precisely pinpoint underlying root causes (*e.g.,* software, hardware, unknown cases).

To bridge this gap, we present AIDA, a practical and comprehensive system for automated network device RCA in Alibaba Cloud's production network. It significantly reduces TRCA, by learning from historical incidents and expert knowledge encoded in emails. Unlike online monitoring, which detects and localizes faults in real time, AIDA focuses on **offline RCA** of faulty devices that have been localized and isolated from production, aiming to identify specific error types and affected modules (*e.g.,* memory, network interface). Based on operational experience, we identify three critical requirements for such a system: *high precision* in conclusion, *interpretability* of reasoning, and *generalizability* to unseen faults. Fulfilling these requirements in practice requires addressing the following core challenges.

Challenge 1: Complexity and heterogeneity of historical emails. Automating RCA is hindered by the complexity and heterogeneity of historical diagnostic emails, our primary data source of operational knowledge. These emails mix technical jargon, natural language, and embedded data like logs or configuration snippets. For an LLM to process an entry like *"'Socket memory error...' on CSR-SVC-CITY1. Version check confirmed no known bugs. Diagnosed as hardware failure, requires RMA,"* it should not only parse content but also infer the multi-step causal reasoning across diagnostic steps. However, general-purpose LLMs typically lack sufficient network domain expertise to perform RCA reliably. Some works adopt end-to-end LLM fine-tuning to implicitly inject domain knowledge [2, 21, 32], but rely heavily on large-scale, high-quality data and lack interpretability. Others explicitly leverage historical knowledge via retrieval-augmented generation (RAG). However, they are ineffective at extracting structural, causal relationships from interactive emails (Observations 1 and 2 in §2.4), which are essential for accurate downstream LLM-based RCA.

To address these limitations, we design *RCA reasoning chains*, a structured representation that captures causal relationships and forms the basis of a knowledge graph (KG) for RAG. Extracting such structured knowledge from large volumes of historical emails is challenging, as it must satisfy stringent constraints such as content completeness and logical coherence (see §3.3). We therefore combine supervised and reinforcement fine-tuning [30] to train a specialized LLM to convert unstructured emails into structured RCA reasoning chains, explicitly enforcing these constraints via reward feedback while mitigating severe label scarcity (§3.3.1).

Challenge 2: Reasoning chains contain noisy, redundant, and outdated historical information. Our production network spans diverse devices and software versions, creating a vast and continuously evolving landscape. We observe that even when individual reasoning chains are accurately extracted from historical data, directly incorporating them into a KG introduces redundant and weakly relevant nodes (Figure 8-right), degrading retrieval efficiency and inference effectiveness.

To address these issues, we employ a SimRank-based [17] graph consolidation approach with adaptive weights to capture structural consistency while prioritizing high-specificity signals. By merging semantically equivalent nodes in the reasoning chains, it transforms case-specific insights into reusable general knowledge, reinforces critical causal pathways, and enables the automatic derivation of generic log regex templates [47] across cases, which help pinpoint relevant logs during LLM inference (§3.3.2). To maintain KG freshness, we periodically remove reasoning chains associated with deprecated devices or modules and insert new reasoning chains extracted from recent emails.

Challenge 3: Complexity of multi-step reasoning. Even with a condensed KG, automated RCA remains challenging at inference time, due to scalability, reliability, and efficiency limits. First, the massive volume of real-time logs can overwhelm chain-of-thought (CoT)-based [40] LLM reasoning. Second, the probabilistic nature of LLMs makes them susceptible to reasoning drift, error propagation, and hallucinations [16]. Finally, a single incident may correspond to multiple reasoning chains, leading to redundant validation and complicating diagnosis synthesis.

To address this, we implement a three-stage inference process (§3.4). First, we retrieve and filter reasoning chains by problem descriptions, ensuring symptom coherence and avoiding misdiagnoses from isolated log patterns. Next, we apply semantic templates derived from these chains to prune logs to fault-relevant entries, reducing LLM load. We then decompose analysis into lightweight verification steps, injecting step-specific context (*e.g.,* exemplar logs) to narrow the LLM's focus. Finally, we validate chain consistency and confidence to pinpoint the most probable root cause.

Fundamental novelty and key results. AIDA's novelty is to transform deployment-driven domain knowledge embedded in

noisy, unstructured industrial RCA emails into a structured knowledge graph, and leverage this knowledge to guide LLM reasoning for real-time diagnosis of new incidents. AIDA has been deployed in our production cloud for over one year, handling 846 network device failures. It reduces median TRCA from 72.6 hours to 1.6 minutes and lowers the 90th-percentile TRCA for complex cases from 329.9 hours to 19.4 hours. These results achieve a precision of 95.4%, providing our operators with rapid, trustworthy RCA.

This work does not raise any ethical issues.

2 Background and Motivation

Alibaba Cloud operates a large-scale global network infrastructure supporting worldwide services. The network has grown to over hundreds of thousands of devices across dozens of data centers spanning tens of geographic regions, with hundreds of types of hardware and software failures observed (see Figure 3). Failures in network devices can trigger widespread and cascading disruptions across regions and services. For example, Google reported that a latent software defect in a backbone router caused high latency and error rates for multiple Google Cloud services in the asia-southeast2 region [1] on November 16, 2024. While immediate mitigation restores service availability, effective RCA is needed to prevent recurrence and sustain long-term reliability. Ensuring the reliability of this large-scale infrastructure is the primary responsibility of our network operations team.

2.1 Network Device RCA Basics

Incident response in cloud-scale networks generally consists of two stages: (1) rapid fault localization and temporary mitigation (*e.g.,* device isolation) to restore service availability; and (2) device-level RCA—our focus—to pinpoint the specific error type and affected modules. Since restarts or reconfiguration often leave device faults unresolved, accurate RCA is critical to prevent recurrence.

Delays, costs, and risks of network device RCA. However, the current manual RCA process (Figure 1) typically involves multiple rounds of back-and-forth communication, in which vendor experts guide operators to run diagnostic commands, collect logs, analyze the results, and request additional data. Figure 2 summarizes statistics from the past year. This labor-intensive process, compounded by a high alert volume, burdens operators and incurs substantial vendor support costs. More critically, prolonged TRCA increases the risk of catastrophic, cascading service disruptions.

2.2 Our Goal

We aim to build a system for efficient RCA in our large-scale heterogeneous network infrastructure, significantly reducing TRCA. Based on our operators' hands-on experience, such a system must satisfy three requirements:

- **High Precision**: the system identifies the actual root cause with trustworthiness.
- **Interpretability**: diagnostic results are explainable and independently verifiable by operators.
- **Generalizability**: the system supports easy integration of new diagnostic knowledge and methods.

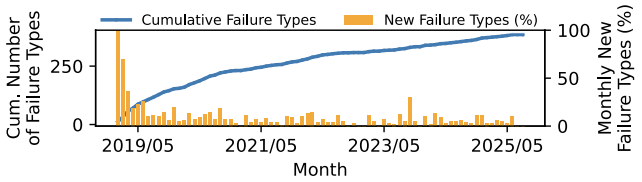


Figure 3: Growth of distinct software and hardware failure types and the monthly share of newly types over 5-year period (27% of failure cases involve failure types that occur fewer than 10 times a year).

2.3 LLMs for Network Device RCA

Achieving our goal is critical yet challenging. We leverage historical diagnostic emails, a largely untapped repository of expert knowledge, as the foundation of our approach.

Why LLMs offer a new path forward? Trained with large text corpora, large language models (LLMs) offer powerful natural language understanding and reasoning capabilities [40, 41]. They can process raw or semi-structured data (*e.g.,* diagnostic reports, log entries, and tables) without destructive parsing. By enabling holistic, cross-source correlation that mirrors human operators, they sidestep the rigid schemas and shallow reasoning that bottlenecks traditional models.

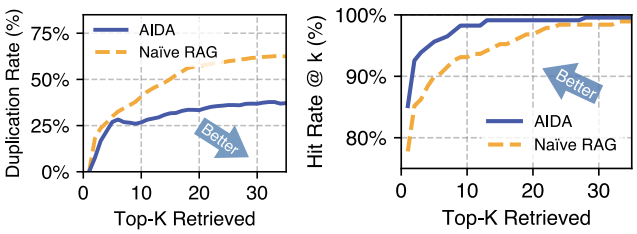
Bridging the domain knowledge gap with RAG. Prior work [2, 21, 32] relies on fine-tuning to encode proprietary device knowledge, but this approach is data-intensive, costly to adapt to new failure types (Figure 3), and difficult for operators to verify or detect hallucinations. We therefore adopt RAG [22] as a flexible and transparent alternative. Instead of generating reasoning paths via generic chain-of-thought (CoT) [40], AIDA retrieves pre-extracted RCA reasoning chains from a curated KG to guide LLM inference, grounding diagnosis in traceable, domain-specific evidence. This design enables low-cost, incremental knowledge updates, keeping the system accurate and adaptable to evolving devices, modules, and failure patterns.

2.4 Key Observations

Observation 1: Data redundancy degrades the retrieval efficiency of naïve RAG. Historical network incidents are inherently duplicative: distinct cases share identical evidence nodes and root causes. This yields a long-tailed distribution (Figure 3): a small number of common root causes (*e.g.,* optical module failure) generates a large volume of recurring events (*e.g.,* port flapping). The duplication rate over retrieved cases presents a critical challenge to RAG-based diagnostic systems [4, 37, 48], by biasing similarity-based retrieval towards the dense common case clusters. As a result, context saturates (Figure 4a), which marginalizes rare failure types and impairs diagnostic efficiency, leading to diagnostic *blind spots* and reduced hit rate[1] (Figure 4b).

Observation 2: Generic knowledge graphs (KGs) fail to capture causal semantics for RCA. As a natural solution to mitigate data redundancy [8, 12, 29, 33, 46], generic KGs are ill-suited for network RCA due to limitations in both their construction and

[1] Hit Rate @ k (HR@k) is calculated as $\text{HR@k} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(\text{rank}_i \le k)$, where rank_i is the rank of the ground-truth item for the i-th case, and $\mathbb{I}(\cdot)$ is the indicator function (1 for a hit, 0 for a miss).



(a) Higher duplication rate in naïve RAG's retrievals. **(b) Lower hit rate of naïve RAG due to redundant retrievals.**

Figure 4: The impact of data redundancy on naïve RAG retrieval. (a) Naïve RAG system retrieves a high rate of duplicate root causes, which (b) consequently leads to a lower hit rate of relevant root causes.

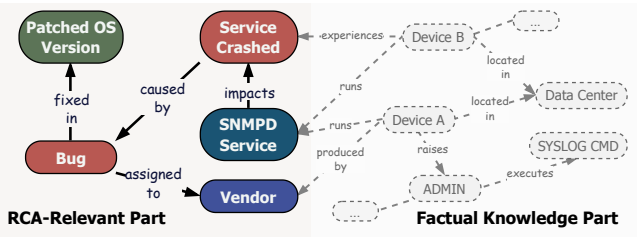


Figure 5: Generic knowledge graphs conflate the essential causal subgraph for RCA (left) with a large volume of noisy, correlational facts (right), where dotted-style links are non-causal, introducing noise into reasoning.

application. The primary issue lies in construction: treating RCA documents as plain text, these methods mistake statistical correlation for true causation. As shown in Figure 5, the resulting graph generated by GraphRAG [8] is dominated by *noisy, non-causal links* that obscure meaningful diagnostic signals. Using such a graph for RAG, semantically noisy and structurally flat information will be retrieved and fed to the LLM. The LLM must perform the difficult task of inferring causal relationships from a vast amount of irrelevant data. These methods can tell the LLM *what information might be relevant* but fail to guide it on *how to reason through it*, leading to incorrect analysis.

Observation 3: Fine-tuning LLM with domain knowledge improves causal coherence. Given the limitations of naïve RAG and generic KGs, we propose constructing causal KGs by extracting and structuring *reasoning chains* that formalize the *expert-asserted causality* already embedded in historical RCA emails. A critical question then arises: *where do these high-quality reasoning chains come from?* While prompting a general-purpose LLM can generate such chains, our experiments (Figure 6) show that limited domain expertise leads to poor structural quality, particularly in step granularity and causal coherence, despite reasonable content (see §3.3 for formal definitions of these metrics). In contrast, a specialized LLM fine-tuned with our domain knowledge significantly improves both metrics, necessitating an effective mechanism to reliably extract structured reasoning chains from historical emails.

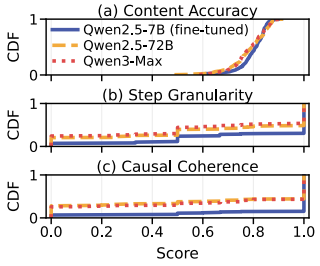


Figure 6: Quality advantage of reasoning chains from fine-tuned model.

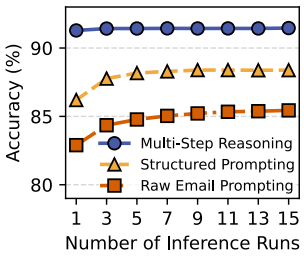


Figure 7: Superiority of multi-step reasoning for RCA inference.

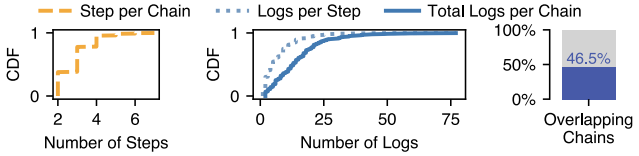


Figure 8: Key properties of the extracted reasoning chains: (1) multi-step structure (left), (2) massive evidence log volume (center), and (3) high redundancy with 46.5% shared steps between chains (right).

Observation 4: Multi-step LLM reasoning enhances RCA performance. Figure 7 shows that Structured Prompting, which augments LLM inference with retrieved structured reasoning chains[2], outperforms Raw Email Prompting, based on unstructured emails, as structured context guides LLM's attention to relevant attributes, *e.g.*, component, failure severity. However, directly applying these reasoning chains introduces two challenges: First, their sheer length (Figure 8-left) and data volume (Figure 8-center) can overwhelm single-pass LLM analysis. Second, naïve, chain-by-chain execution incurs significant redundancy (Figure 8-right), due to overlapping steps across reasoning chains. To address this, we propose Multi-Step Reasoning, a divide-and-conquer strategy that decomposes extracted reasoning chains into simpler, verifiable steps that incrementally guide the LLM (§3.4). This reduces average token consumption per case by 64.8% (from 41,723 to 14,691 tokens).

3 Design

Driven by these observations, we design AIDA, an LLM-based system that distills unstructured diagnostic knowledge into a KG and leverages RAG-assisted multi-step reasoning for automated network device RCA. This section presents the core design of AIDA, guided by our observations in §2.4.

3.1 Overview

As shown in Figure 9, AIDA operates in two phases.

Knowledge fusion. This process begins by classifying emails between vendors and network operators into different categories (§3.2), and then using a fine-tuned LLM to extract reasoning chains with structural constraints and label scarcity (§3.3.1). A coherent KG is then formed using these chains through a dynamic fusion

[2]We perform Structured Prompting in a multi-round manner, which injects one retrieved reasoning chain per inference round and deciding the final result via majority voting. It is generally preferable for long context inference, and we adopt it to enhance our baseline RCACopilot in §5.

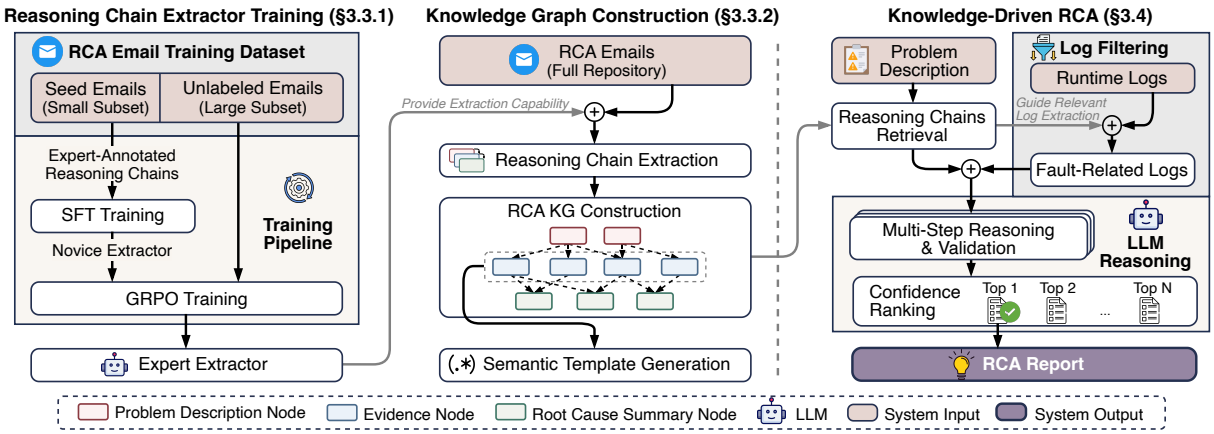


Figure 9: System architecture of AIDA.

process, with log templates generated for log filtering during online diagnosis (§3.3.2).

Structured RCA reasoning. When a new incident occurs, AIDA leverages the KG for automated diagnosis. First, it retrieves relevant reasoning chains and applies the log templates to pinpoint critical evidence in raw logs. Next, AIDA guides the LLM through a step-by-step validation of the reasoning chain. The analysis concludes with a diagnostic report that provides the root cause and a verifiable evidence trail showing how the conclusion was reached (§3.4).

3.2 Preprocessing

AIDA first processes an archive of 21,482 emails and identifies two types relevant for our KG construction: (1) **Problem descriptions:** Initial incident reports specifying observed symptoms. (2) **RCA reports:** Conclusive analyses with expert insights and proprietary failure signatures. Since applying LLM to the entire corpus is unnecessary and computationally prohibitive, AIDA instead employs a lightweight support vector machine (SVM) classifier to efficiently filter the data. We manually labeled 5,411 emails from 981 failures[3]. We used Qwen's text-embedding-v3 with SVM hyperparameters (kernel, penalty) optimized via grid search and 5-fold cross-validation on failure-stratified splits of the 5,411 labeled emails. This lightweight classifier achieved ~95% accuracy, ensuring high-fidelity filtering of historical emails for downstream KG construction. The classifier is then trained on text embeddings of labeled emails and used to filter the corpus for the relevant emails for KG construction. The dataset comprises thousands of unique devices, averaging 3.7 RCA reports per device.

3.3 Knowledge Graph Construction

As general-purpose LLMs struggle to enforce domain-specific constraints (§2.4), the complex, unstructured classified emails (§3.2) are fed into a tailored KG construction pipeline: (1) extracting reasoning chains via a specialized model fine-tuned by RL (§3.3.1) and (2) merging semantically equivalent nodes to remove redundancy and generate templates (§3.3.2).

3.3.1 Reasoning Chain Extraction. AIDA formalizes historical cases into RCA reasoning chains, each of which comprises three types of nodes: (1) a problem description node encoding initial incident symptoms; (2) multiple evidence nodes, capturing atomic pairs of diagnostic findings and their supporting log evidence; and (3) a root cause summary node, synthesizing evidence into a conclusion.

Progressive two-stage fine-tuning. AIDA tackles the challenges of scarce chain labels and complex domain-specific constraints in following steps. It first bootstraps a novice extractor via supervised fine-tuning (SFT) on a small, high-quality corpus of expert-annotated reasoning chains. Two experts labeled 50 reasoning chains (~5% of corpus) in 2.5 person-hours; labels in the SFT procedure are structured chains shown in Figure 10. AIDA then iteratively groups relative policy optimization (GRPO) [30] to refine the model into an expert, with the following reward function proposed to capture extraction constraints.

RFT reward function design. We use a composite reward to jointly optimize the structural fidelity and causal validity of extracted reasoning chains:

$$r = \alpha r_c + (1 - \alpha) r_l, \tag{1}$$

where $\alpha = 0.6$ is a weight determined via empirical search to balance the two components.

• **Content completeness (r_c)** measures the extraction's fidelity to the source RCA report. For natural language components (*e.g.*, symptoms, root causes), r_c computes the N-gram similarity (*e.g.*, ROUGE-L [23]) against the source. For log evidence associated with evidence nodes, we enforce *strict string matching* to penalize hallucinations, ensuring that extracted logs remain bit-accurate for downstream analysis.

• **Logical coherence (r_l)** assesses the causal integrity of the reasoning chain. We employ a larger LLM as an automated judge to evaluate two properties: (1) *step granularity*, verifying if each evidence node's finding is logically supported by its log and describes a single, atomic event; and (2) *causal coherence*, checking if the entire sequence forms a coherent causal narrative consistent with the original report's logic.

3.3.2 RCA Knowledge Graph Construction. We construct a unified KG, $G = (V, E)$, to distill reusable diagnostic knowledge from a corpus of historical incidents $\mathcal{I}$. For each incident $I_i \in \mathcal{I}$, an RCA reasoning chain, denoted as $C_i = (p^{(i)}, n_1^{(i)}, n_2^{(i)}, \ldots, n_{k_i}^{(i)}, rc^{(i)})$, is

[3]This labeling is a one-time bootstrap with negligible overhead—only 2 experts classified 5,411 emails into 3 categories in 10 person-hours.

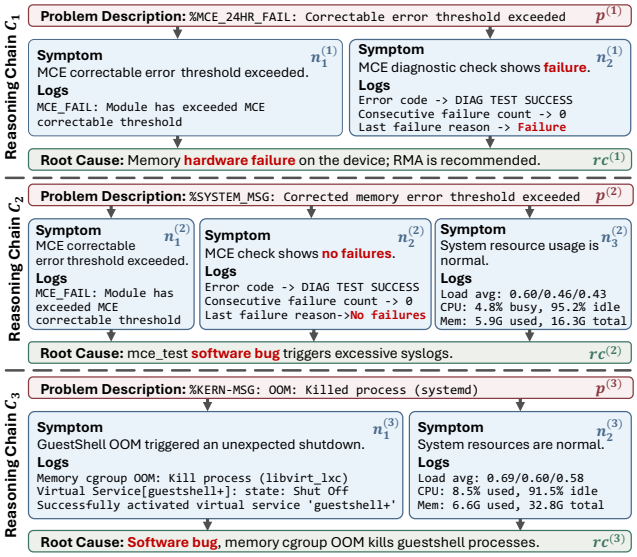


Figure 10: Illustration of the limitation of content-based similarity for node merging: $n_2^{(1)}, n_2^{(2)}$ are prone to being merged due to high lexical overlap, yet they represent opposite diagnostic states and should remain distinct causal pathways; incorporating their divergent root causes $rc^{(1)}$ and $rc^{(2)}$ prevents this mis-merging (introducing $S_{\text{prop}}^{(t)}(u,v)$ in (2)). Conversely, $n_3^{(2)}$ and $n_2^{(3)}$ should be merged despite low similarity across their root causes, motivating the use of a low pairwise weight (w_{uv} in (2)) and our adaptive weight mechanism.

extracted via our fine-tuned model (Figure 10). The chain represents a diagnostic path from problem description $p^{(i)}$, through a sequence of evidence nodes $n_j^{(i)}$, to final root cause summary node $rc^{(i)}$ (§3.3.1). We next consolidate incident-specific reasoning chains into a unified KG that captures reusable diagnostic knowledge while preserving alternative causal pathways.

Step 1: Context-based partitioning. To improve scalability and accuracy, we partition incidents $\mathcal{I}$ into $\mathcal{P} = \{P_1, \ldots, P_M\}$ based on metadata (*e.g.,* vendor and product model), such that incidents in each partition share identical attribute values. We will construct a graph for each partition by merging incident-specific reasoning chains within the partition in the following step, preventing erroneous merges across different vendors and device types.

Step 2: Structure-aware merging of evidence and root-cause nodes. We then merge semantically equivalent nodes within each partition. As shown in Figure 10, relying solely on embedding-based similarity is insufficient as it fails to capture *semantic nuance* and ignores *relational context*. We therefore define node equivalence using *content similarity* and *relational consistency*. Specifically, we model evidence and root-cause nodes as a bipartite graph and iteratively refine the same-type node similarity $S(u,v)$ for each node pair (u,v), inspired by SimRank [17]:

$$S^{(t+1)}(u,v) = (1 - w_{uv}) \cdot S_{\text{cont}}(u,v) + w_{uv} \cdot S_{\text{prop}}^{(t)}(u,v), \quad (2)$$

where the final score combines the initial content similarity (S_{cont}) and the structurally propagated similarity ($S_{\text{prop}}^{(t)}$). This mechanism has two key components: (1) **Propagated similarity ($S_{\text{prop}}^{(t)}$) (detailed in Appendix A.2.1):** Based on the principle that "two nodes are similar if their neighbors are similar," we iteratively propagate similarity across the connected nodes, capturing the relational context in the bipartite graph. (2) **Adaptive weight (w_{uv}):** This weight gauges a node's *diagnostic specificity*, using evidence–root-cause associations captured by similarity scores in the KG. For discriminative evidence (*e.g.,* "MCE failure"), it prioritizes structural refinement (S_{prop}). For ubiquitous states (*e.g.,* "resource normal"), that are weakly connected to root causes, a smaller w_{uv} increases the impact of content similarity (S_{cont}). Specifically, let $\mathbf{W}_{\text{rc}}$ and $\mathbf{W}_{\text{ev}}$ denote the adaptive weight matrix for root-cause and evidence-node pairs, respectively. Formally, for a root cause pair (rc_i, rc_j), the element $(\mathbf{W}_{\text{rc}})_{ij}$, corresponding to w_{rc_i,rc_j}, is:

$$(\mathbf{W}_{\text{rc}})_{ij} = \min\left(\sqrt{\bar{\sigma}(rc_i)\,\bar{\sigma}(rc_j)},\ \lambda_{\text{rc}}\right), \quad (3)$$

where λ_{rc} is a clipping hyperparameter and $\bar{\sigma}(u)$ quantifies the node's **specificity** with its derivation provided in Appendix A.2.2. For evidence-node pairs, the matrix $\mathbf{W}_{\text{ev}}$ is defined analogously.

Intuition. The geometric mean in Eq. (3) ensures that relational influence is amplified only when *both* nodes are specific.

After $S^{(t)}(u,v)$ converges, we perform agglomerative clustering on evidence and root cause nodes separately to consolidate them into canonical entities. In parallel, problem description nodes ($p^{(i)}$) are clustered using content similarity only, forming canonical entry points for the KG's reasoning chains and targets for matching incoming queries.

Step 3: Semantic-aware log regex template generation. Existing log template generation methods distinguish constants from variables (*e.g.,* Drain [15], Spell [5], LILAC [20]), but often mask variable fields containing failure-specific patterns, producing generic templates that match both normal and anomalous logs, thereby degrading retrieval accuracy. Moreover, they cannot prioritize critical logs capturing failure symptoms. We address this as follows. (1) We measure the cosine similarity between each word embedding in the logs and the evidence node's symptom embedding using Qwen's text-embedding-v3, selecting the top-N log lines ($N = 2$ in Figure 11; empirically $N \in [1,3]$ suffices). (2) We construct log regex templates by selecting KG-prevalent and symptom-aligned words (*e.g.,* "Last", "Failure" in Figure 11). Each word is characterized by its prevalence (frequency in the KG) and its similarity to the symptom sequence. We first discard words with similarity below 0.5 and group the remaining words by frequency. The groups are then processed in descending frequency order, and all words in a group are added to the template until the preset template size (typically 4) is nearly reached. For the final group whose inclusion would exceed the template size, words are ranked by descending similarity, and only the required number of highest-similarity words are selected.

3.4 Structured RCA Reasoning

Direct LLM-based RCA is limited by scalability (log volumes exceed context limits) and veracity (unguided hallucinations). AIDA addresses both with a three-stage inference pipeline.

Reasoning chain retrieval. Our design builds on the insight that *similar problem descriptions often share diagnostic logic*. For a new incident, AIDA performs a semantic search on the KG using

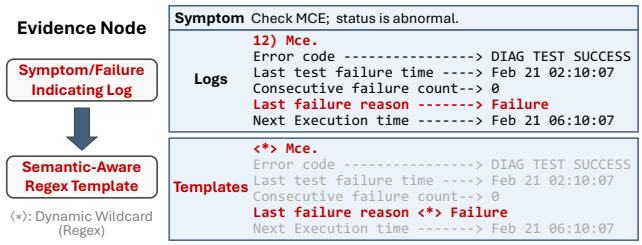


Figure 11: An example of semantic-aware log template generation.

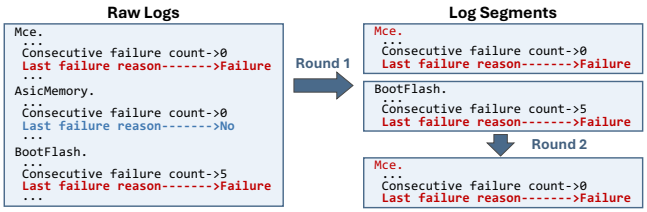


Figure 12: An example of AIDA's two-round filtering. Round 1 uses a single template *"Last failure reason <*> Failure"* to extract the Mce and BootFlash segments; Round 2 combines it with *"Mce ."* to isolate the target.

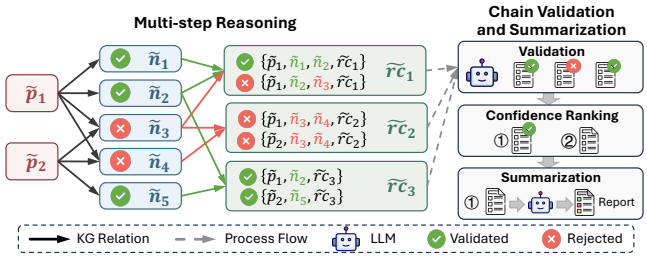


Figure 13: Example of AIDA's structural reasoning process: step-wise validation (✓) using retrieved reasoning chains, root cause ranking, and RCA report generation.

the problem description as a query, which first filters by vendor, OS, and hardware type (*e.g.,* modules or fabrics) to ensure knowledge applicability on a new failure and then retrieves the top-K most relevant reasoning chains as candidate diagnostic hypotheses, $C_{\text{cand}} = \{C_1, C_2, \ldots, C_K\}$.

Trace generation and multi-step reasoning. For each C_k, AIDA attempts to construct a corresponding *reasoning trace*, an incident-specific instantiation of C_k supported by newly-collected logs. This is achieved by validating each evidence node in C_k via a two-round filtering method, illustrated in Figure 12.

• **Round 1**: AIDA prunes irrelevant logs that do not satisfy the OR-combined log regexes of the node[4].

• **Round 2:** To match the multi-line (*e.g.,* $N = 2$ in Figure 11) template pattern, AIDA further prunes the logs that fail to pass the AND-combined log regexes of the node.

Evidence nodes with no matched logs after the above filtering steps are rejected.

Node validation. For each surviving evidence node, *e.g.,* $\tilde{n}_1$ in Figure 13, AIDA prompts the LLM to verify whether the extracted

[4]Unified regexes accelerate matching via automaton-level optimization.

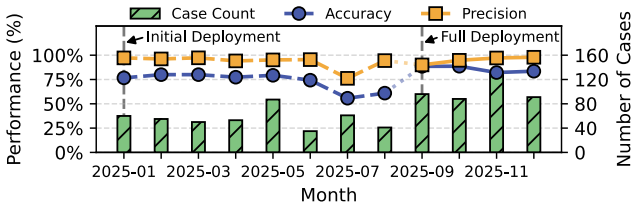


Figure 14: One-year production performance and workload of AIDA.

logs (1) factually support the node's symptom description (*e.g., "BGP session flaps"*) and (2) align with historical log samples. Nodes failing either check are rejected, which leads to the invalidation of the entire trace, ensuring local consistency within each node.

Holistic chain validation and confidence-based ranking. A subset of the initial K candidates that successfully pass the step-wise validation then undergo a final *chain validation* to ensure end-to-end coherence. This check ensures that all logs in a trace originate from the same device module (logical consistency) and produces the set of validated reasoning traces, $\mathcal{T}_{\text{val}}$. Next, AIDA assigns a confidence score $\gamma(T_i)$[5] to each trace $T_i \in \mathcal{T}_{\text{val}}$:

$$\gamma(T_i) = \underbrace{\text{Rel}(C_i)}_{\text{Chain Reliability}} \cdot \underbrace{\prod_{j=1}^{k_i} \text{Conf}(n_j^{(i)} | \ell_j^{(i)})}_{\text{LLM-inferred Evidence Confidence}} \tag{4}$$

Here, $\text{Rel}(C_i)$ is computed from historical incident frequency, and $\text{Conf}(n_j^{(i)} | \ell_j^{(i)})$ denotes the node-specific confidence that log evidence $\ell_j^{(i)}$ factually supports the corresponding hypothesis of evidence $n_j^{(i)}$, with its formal formulation provided in Appendix B.4. This directly penalizes conclusions built on hallucinated evidence. Since multiple, distinct reasoning traces may point to the same root cause, AIDA aggregates their scores into an overall confidence $\Gamma(rc_a)$ via $\Gamma(rc_a) = 1 - \prod_{T_i \in \mathcal{T}_a}(1 - \gamma(T_i))$, where $\mathcal{T}_a$ contains all validated traces for root cause rc_a. Finally, AIDA selects the root cause with the highest aggregated score ($\tilde{rc} = \arg\max \Gamma(rc_a)$) and prompts the LLM to synthesize a concise RCA report.

4 Deployment Experience

AIDA has been deployed in Alibaba Cloud for over a year, serving as an automated post-incident RCA engine. Integrated into the operational workflow, it is triggered after a monitoring system localizes a fault to a specific device. Upon receiving the device's logs and an initial alert, AIDA autonomously performs RCA. To date, it has processed 846 failure cases, handling an average of 101 cases per month since full deployment. To support operator trust and interpretability, AIDA generates comprehensive RCA reports augmented with key log snippets as evidence, citations to historical cases with similar reasoning chains from its knowledge graph, and confidence scores for its diagnosis.

4.1 Performance in Production

RCA accuracy and precision. We evaluate AIDA using two metrics. *Accuracy* is the fraction of all cases where AIDA provides the correct root cause. *Precision* is computed only over cases where

[5]Formal derivations of $\gamma(T_i)$ are shown in Appendix B.

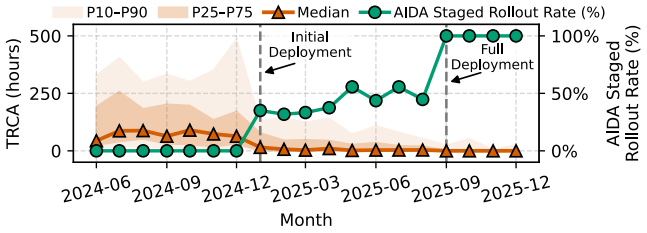


Figure 15: Time to RCA (TRCA) of AIDA in production.

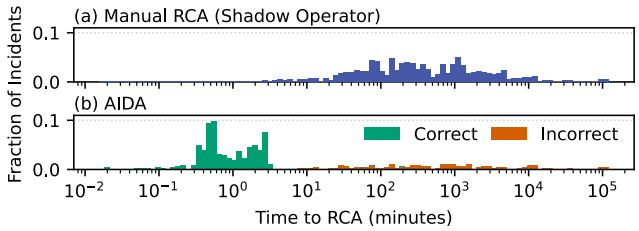


Figure 16: TRCA of manual RCA vs. AIDA.

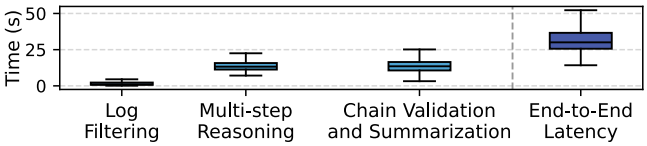


Figure 17: End-to-end latency breakdown of AIDA.

AIDA provides a diagnosis and measures the fraction of those diagnoses that are correct. As shown in Figure 14, AIDA achieves consistently high precision, averaging 95.4% over the full deployment period. Its accuracy is lower due to a conservative design that prioritizes operator trust: AIDA provides a diagnosis only when there is high-confidence evidence, since an incorrect diagnosis is often more harmful than providing none at all. Nevertheless, its accuracy improved over time, rising from 70.2% during the initial deployment to 85.4% in the full deployment phase. This demonstrates AIDA's ability to learn and expand its coverage while remaining trustworthy.

Handling new incidents. AIDA can also handle previously unseen cases with a precision of 75.3%. When confidence is insufficient, such as for novel error types or failures involving new devices or modules, AIDA abstains and reports the case as out of scope, rather than producing an unreliable diagnosis. Consistent with operator preferences, such cases are seamlessly handed off for manual analysis. AIDA generalizes to unseen device models by leveraging shared causal mechanisms across related series. Despite log variances, it applies consistent symptom-level causal reasoning to perform RCA. However, AIDA remains limited when failures arise from entirely novel causal mechanisms specific to new hardware or OS absent from its KG or LLM's knowledge.

End-to-end RCA latency. AIDA has significantly reduced the median Time to RCA (TRCA) from 72.6 hours to 1.6 minutes (Figure 15) with increasing rollout. Here, rollout rate denotes the percentage of device failure categories handled by AIDA. The speedup is even more pronounced for complex cases, cutting the 90th percentile for long-tail incidents from 329.9 hours (nearly 14 days) to 19.4 hours. A shadow evaluation performed by human operators running in parallel with AIDA confirms this speedup (Figure 16). Comparing manual RCA distribution (top blue in Figure 16) with the incidents requiring manual fallback after AIDA (bottom orange) reveals that AIDA successfully automates cases across the entire spectrum of RCA difficulty, including many incidents that would otherwise require hours or even days of manual investigation. While the manual process exhibits a wide, long-tailed distribution spanning hours to days, AIDA resolves the vast majority of incidents (green) in minutes. The small fraction of unresolved cases (orange) requiring manual fallback underscores its effectiveness in automating the bulk of the RCA workload, thus reserving operator expertise for the most complex exceptions. This reduction in TRCA is also supported by AIDA's efficient pipeline (Figure 17). The initial Log Filtering stage based on regex-based semantic templates completes within 12 seconds at the 90th percentile. Multi-step Reasoning and the stage of Chain Validation and Summarization incur the largest latency but remain fast, with 90th-percentile latencies of 18.3 and 22.0 seconds, respectively, due to our parallelized design. Together, they yield an average core analysis time of 37.4 seconds and a 90th percentile of 54.4 seconds, ensuring that AIDA's computation is never a bottleneck.

4.2 Case Study

We present three representative cases illustrating software, hardware, and unknown root causes. We focus on how AIDA validates reasoning-chain evidence against device logs. In the following cases, each figure demonstrates the step-by-step workflow of AIDA, from the initial problem description to the final RCA conclusion, focusing on how reasoning chains are validated against device logs.

Root cause in software. In this case, AIDA identified the root cause of a virtual service failure as a documented software bug. Triggered by a process-level memory alert *"cgroup out of memory"*, AIDA searched the KG for similar symptoms and retrieved a relevant chain, involving three nodes: (A) service reactivated, (B) memory state recovered and (C) no core dump file. It then validated these symptoms by matching extracted logs against node-specific symptoms and log exemplars. By confirming that the logs substantiated all chain conditions, AIDA pinpointed the documented software bug linked to the chain as the root cause.

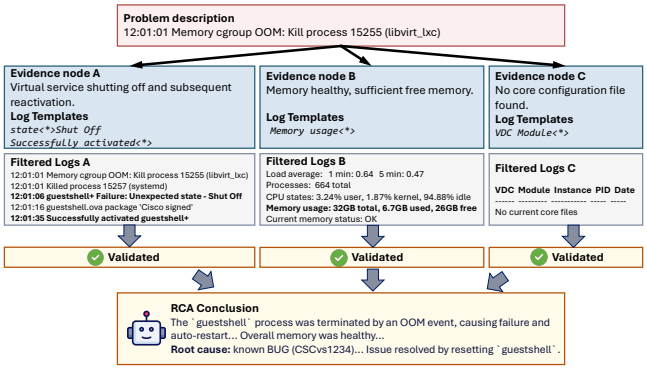


Figure 18: Case 1: root cause in software.

Case 1. Figure 18 shows a software-bug diagnosis triggered by a "`Memory cgroup out of memory`" alert. AIDA retrieves a reasoning chain with three evidence nodes (A, B, C) and validates each using its log template and filtered logs. Node C illustrates a non-trivial

negative check: with a fixed log window, the regex template extracts diagnostic command output, allowing the LLM to interpret "No current core files" as evidence that no core dump was generated. Since all nodes are validated, AIDA confirms the chain and identifies the known software bug and its resolution.

Root cause in hardware. In this case, a device entered read-only mode, triggering multiple alerts such as: "`journal commit I/O error`". AIDA queried KG for similar symptoms and retrieved a relevant chain with two nodes: 1) "`BootFlash model is Micron_5100_MTFD`" and 2) "`BootFlash self-test failed`". It then examined the extracted log segments, and confirmed that the BootFlash model on the device was indeed Micron_5100_MTFD and its self-test had failed. As the chain was linked to a BootFlash hardware failure in the KG, AIDA concluded that it was indeed the root cause, prompting module replacement to operators.

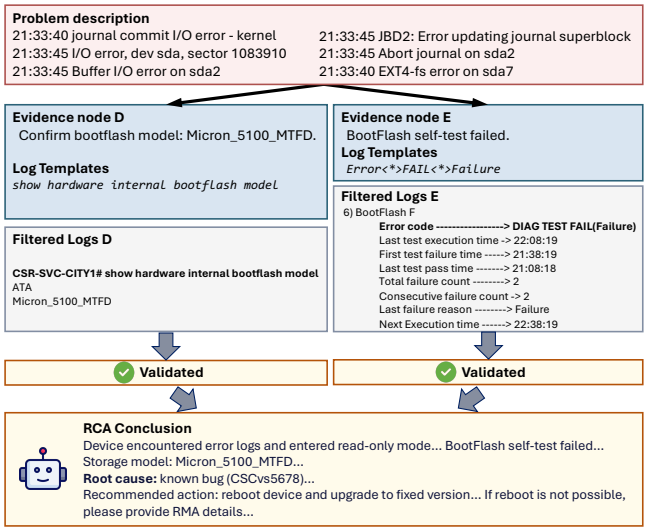


Figure 19: Case 2: Root cause in hardware.

Case 2. Figure 19 shows a hardware-failure diagnosis from I/O-error alerts. The retrieved chain verifies the BootFlash model and self-test status. For node D, the filtered logs confirm the "`Micron_5100_MTFD`" model; for node E, regex filtering with a fixed log window captures a table-like diagnostic snippet with failure counts and timestamps, avoiding a premature judgment from a single "`DIAG TEST FAIL`" line. With both nodes validated, AIDA confirms the chain, pinpoints the hardware issue, and recommends replacement.

Handling unknown root causes. Not all failures can be matched to existing knowledge. In one incident, BGP sessions malfunctioned due to an unknown software bug (root cause not in the KG when we processed it). When queried with the monitor alert "`Physical interface: et-0/0/0:1, Enabled, Physical link is Down`", AIDA retrieved a reasoning chain with two evidence nodes: F verifying "`session status goes up after port isolation`" and G verifying "`CBR port remains up`". After examining the filtered log, AIDA found that the session went up after port isolation, but the CBR port went down, contradicting the expected pattern from node G. Therefore, holistic validation rejected this reasoning chain (and other chains for similar reasons). The unresolved issue was handed to the vendor expert for further investigation. This shows that AIDA reliably avoids drawing false conclusions from partial evidence.

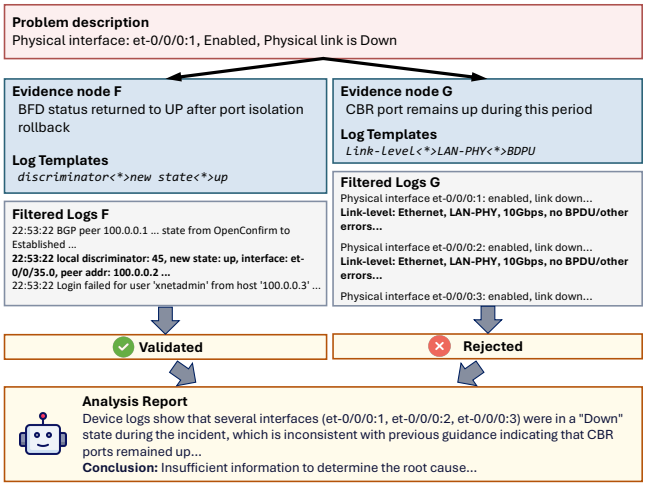


Figure 20: Case 3: Handling unknown root causes.

Case 3. Figure 20 demonstrates how AIDA avoids misdiagnosis when faced with a novel issue. Given a "`Physical link is Down`" alert, AIDA tests a candidate reasoning chain with two nodes (F, G). The figure shows that while the log evidence validates node F (BFD status recovery), it contradicts the symptom described in node G (CBR port status up). The LLM thus rejects node G, leading to the invalidation of the entire reasoning chain. Consequently, instead of a wrong conclusion, AIDA produces an analysis report indicating that the root cause is indeterminate with existing knowledge, and hand over the issue to vendor expert for manual investigation.

5 Performance Evaluation

We first compare AIDA with state-of-the-art approaches to assess its core design (§5.1) and evaluate LLM backbones. We then conduct ablation studies to quantify the contribution of each major component (§5.2), followed by microbenchmarks analyzing individual components, including model reliability, KG quality, and log filtering efficiency (§5.3).

Experiment setup. Our evaluation is based on a dataset of 180 real-world, non-trivial failures from our production network, collected over the past year. Each case includes raw device logs and associated diagnostic emails. We run AIDA and all baselines on a server with an Intel Xeon Platinum 8369B CPU and 32 GB RAM. We fine-tune the LLM to extract reasoning chains on a machine with eight NVIDIA A100 GPUs (80 GB VRAM each).

Models. We fine-tune `Qwen2.5-7B-Instruct` for reasoning chain extraction (§3.3) [27]. The more powerful `Qwen3-Max` serves as both the reward-scoring model during fine-tuning and our primary end-to-end inference model (§3.4). We use `text-embedding-v3` for all embedding tasks, including email classification, KG semantic retrieval, and semantic-aware log-template generation. We evaluate AIDA against other leading LLMs in §5.1.

5.1 Algorithm Evaluation

Baselines. We compare AIDA with three representative RAG-based baselines. For all methods, a hyperparameter K controls the amount

Table 1: Accuracy (%) and Precision (%) of AIDA and baselines across different backbone LLMs. AIDA consistently outperforms all baselines, which show only marginal gains from model scaling. Δ_{Acc} and Δ_{Prec} denote the absolute gains of AIDA over the best-performing baseline in percentage points (pp).

Model	Size	Accuracy (%, ↑)					Precision (%, ↑)				
		AIDA	RCACopilot	RCACopilot+	LightRAG	Δ_{Acc}	AIDA	RCACopilot	RCACopilot+	LightRAG	Δ_{Prec}
Qwen3	8B	83.1	64.0	72.5	75.4	**+7.7**	93.3	68.2	76.2	75.9	**+17.1**
Qwen2.5	72B	84.9	61.7	73.1	76.0	**+8.9**	95.9	71.2	83.6	84.5	**+11.4**
GLM-4.7	358B	82.6	64.4	73.7	67.4	**+8.9**	93.2	77.9	82.6	88.4	**+4.8**
DeepSeek-V3.2	685B	82.4	64.9	69.6	76.0	**+6.4**	95.0	71.8	70.2	84.0	**+11.0**
GPT-5.2	–	80.0	66.7	68.6	71.9	**+8.1**	95.0	78.2	81.8	91.5	**+3.5**
Qwen3-Max	1T	83.7	62.8	74.3	76.0	**+7.7**	96.6	75.3	78.1	85.6	**+11.0**
Average		82.8	64.1	72.0	73.8	**+8.0**	94.8	73.8	78.8	85.0	**+9.8**

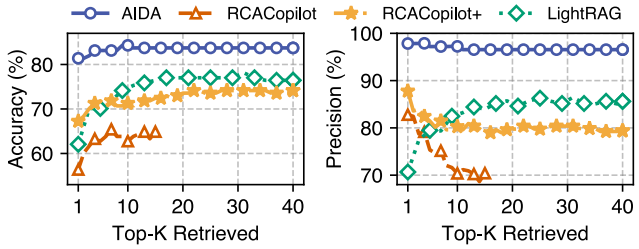


Figure 21: AIDA outperforms baselines both in accuracy and precision.

of retrieved context, representing the number of historical RCA emails for RAG baselines and the number of retrieved reasoning chains for AIDA.

• RCACopilot [4]: A standard RAG-based RCA system that, for a new incident, retrieves the top-K most relevant historical emails, each associated to a distinct error type, concatenates them with a summarized log into a single context, and prompts the LLM to infer the root causes.

• RCACopilot+: An extension of RCACopilot that mitigates long-context limitations and does not require the top-K retrieved documents to correspond to distinct error types. It queries the LLM separately for each retrieved document and determines the final result via majority voting, representing the multi-query RAG paradigm in recent RCA studies [26, 50].

• LightRAG [12]: A lightweight approach that improves GraphRAG [8] in graph construction efficiency and retrieval quality. It serves as a representative generic graph retrieval [8, 32], in contrast to our domain-specific KG design.

Effectiveness and stability. Figure 21 evaluates performance as K varies, demonstrating AIDA's superior accuracy and precision as well as its remarkable stability.

• *Accuracy.* AIDA consistently achieves ~83% accuracy, whereas RCACopilot's quickly plateaus. First, it suffers from *long-context distraction* [24], where concatenated emails overwhelm the LLM's attention and obscure fine-grained diagnostic cues. Second, it retrieves only one historical email for one error type while a single root cause can have diverse diagnostic paths, thereby failing to provide a relevant template for the current incident. While RCACopilot+ (via multi-query) and LightRAG (via better retrieval) mitigate some of these issues, they remain fundamentally limited by shallow reasoning over unstructured historical data rather than structured causal knowledge encoded in reasoning chains.

• *Precision.* High precision is critical in RCA to avoid misleading operators. AIDA achieves over 96% precision, whereas RCACopilot's precision drops sharply as K increases. This is because it requires retrieved emails to cover distinct error types, causing excessive context and diverse error semantics to obscure the diagnostic results. RCACopilot+ addresses these issues, but its noisy intermediate diagnoses are still amplified during aggregation. LightRAG's precision is also volatile. At small K, its generic entity graph with non-causal links results in irrelevant retrieved emails that miss the correct root cause; at larger K, the retrieved emails are more likely to cover correct root cause, but reasoning quality remains limited because the LLM still operates on unstructured raw emails.

Finding 1: AIDA supplies LLMs with structured reasoning chains that encode generalizable diagnostic knowledge, shifting them from interpreting noisy historical emails to reasoning over reusable causal patterns, thereby achieving higher accuracy, precision, and stability than existing RAG-based baselines.

Advantages across LLMs. Table 1 shows that AIDA consistently improves performance across different LLMs, improving accuracy by 8.0 pp and precision by 9.8 pp on average over the strongest baseline. Across all baselines, scaling up the LLM yields only marginal gains, indicating that the performance is constrained more retrieved knowledge rather than model size, as general-purpose LLMs struggle to distinguish critical causal signals from noise in unstructured RCA emails. AIDA alleviates the bottleneck by encoding expert diagnostic logic into structured reasoning chains and multi-step inference, significantly boosting precision (§5.2). As a result, AIDA enables smaller models to remain competitive: AIDA with an 8B model is more accurate than LightRAG with a 72B model (83.1% vs. 76.0%); the precision of the baselines drops with 8B models (*e.g.*, RCACopilot+ drops from 83.6% to 76.2%) while AIDA remains high precision at 93.3%.

Finding 2: AIDA's architectural benefits are independent of the LLM backbone. It enables even smaller, efficient models to deliver reliable RCA performance.

5.2 Ablation Study

We then evaluate the contributions of each component.

Impact of knowledge fusion strategy (§3.3.2). We compare AIDA's node-merging strategy against two variants in Figure 22.

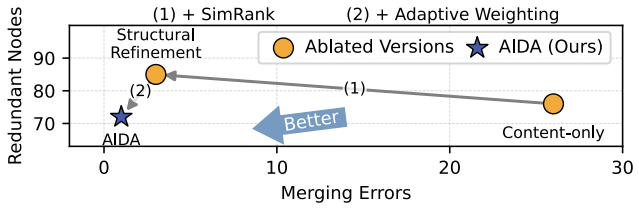


Figure 22: Effectiveness of our structure-aware node merging in KG construction.

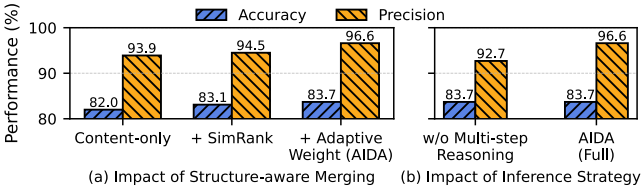


Figure 23: Performance contribution of structure-aware node merging and our inference strategy.

Content-Only relies solely on content embedding similarity and suffers from high merging errors as it fails to distinguish nuanced semantic differences, ultimately corrupting diagnostic causal paths. SimRank leverages causal relations and neighborhood information to resolve ambiguities and greatly reduce mismerges. Our full method further uses *adaptive weighting* to amplify diagnostically specific signals, achieving near-zero merging errors and low redundancy. As shown in Figure 23a, this superior KG improves RCA precision from 93.9% to 96.6% by preserving the structural integrity of reasoning chains.

Impact of inference strategy (§3.4). Compared with AIDA, the baseline w/o Multi-step Reasoning (MR) feeds the entire reasoning chain and all logs into the LLM at once, decreasing precision by 3.9% (Figure 23b). This degradation is caused by information overload: monolithic prompts cause the model to overlook subtle log constraints and critical evidence, leading to incorrect root cause conclusions In contrast, AIDA adopts MR to decompose the task into focused sub-steps with targeted log subsets, reducing information density to ensure key diagnostic signals and model reliability.

5.3 Microbenchmarks

Training time and performance stability (§3.3.1). AIDA utilizes RLF to fine-tune an LLM for reasoning-chain extraction. This process completes in 8.3 hours using four months of historical RCA emails, facilitating seamless retraining as new data becomes available. Over one year, the evaluation reveals no performance degradation, confirming the model's long-term reliability (Figure 24). This sustained performance supports our core insight: decoupling diagnostic structure from specific technical content. By internalizing the underlying logic of expert investigations rather than memorizing the myriad and ever-evolving technical details, the model maintains a high-fidelity reasoning chain extraction even as the underlying network environment evolves.

Knowledge fusion performance (§3.3.2). For a batch of 100 RCA emails, the end-to-end pipeline takes 5.86 minutes on average: reasoning-chain extraction takes 1.01 minutes, LLM-driven RCA KG construction takes 4.16 minutes (71% of the total), and log-template generation takes 0.69 minutes. This process significantly

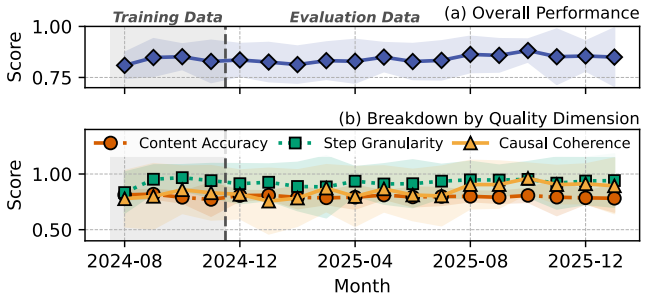
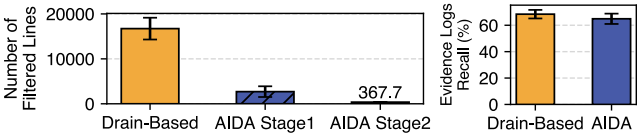


Figure 24: Sustained performance shows learning RCA procedural logic over content for temporal generalization. Shaded regions show ±1 standard deviation.



(a) Number of filtered log lines. **(b) Filtering recall.**

Figure 25: AIDA's log filtering reduces log volume 45.6× with minimal reduction (< 3.5%) on evidence logs recall.

reduces the total number of KG nodes by 62.7% (from 2,367 to 882), with reductions for individual node types ranging from 51% to 73%, yielding a compact graph with 1,272 edges. This curated graph forms the foundation for high-precision inference, which in turn delivers the consistent RCA accuracy to earn operator trust.

Log filtering performance (§3.4). We evaluate our two-round log filtering method using (1) the number of log lines fed into LLM and (2) the recall of evidence logs[6]. We compare AIDA with Drain [15], a representative log template mining method. As shown in Figure 25a, the baseline averages 16,729.4 lines per case, often exceeding LLM context limits due to normal operational noise. In contrast, our semantic-aware multi-template matching reduces log volume to 2,702.7 lines after the first stage (6.1× reduction) and to 367.7 lines after the second (45.6×). By further partitioning logs by evidence nodes, AIDA presents only 32.2 lines per LLM call. Moreover, this drastic reduction preserves diagnostic information. As shown in Figure 25b, AIDA maintains a 64.9% recall, only 3.4% lower than the baseline 68.3%[7]. These results show that AIDA achieves order-of-magnitude log compression with negligible impact on evidence retention.

6 Lessons and discussion

Ensuring objective and comprehensive RCA. While designed to accelerate RCA, AIDA also helps to mitigate human cognitive biases. Investigators often focus on preconceived hypotheses (*e.g.,* familiar or recently seen root causes) and seek confirmatory evidence rather than systematically evaluating alternatives. In contrast, AIDA maps symptoms to potential root causes and evaluates all of them against evidence comprehensively to produce more objective results.

[6]Namely, proportion of ground truth RCA logs retained after filtering.
[7]Some evidence logs in the test set may not appear in the training set used to construct KG, so recall cannot always reach 100%.

Enhancing matching on descriptions increases RCA precision. Our deployments demonstrate that symptom-indicated context, such as temporal trends and trigger conditions of failures, plays a critical role in mitigating misdiagnosis, as these contexts may define severity and causal propagation. For instance, the diagnostic interpretation of the same log evidence "`<*> BGP Hold Timer Expired`" differs based on the symptoms: "`BGP session lost immediately after Route Injection`" implies a BGP-update-triggered software bug within the route policy, whereas "`Infrequent BGP Down`" points to underlay instability independent of BGP update operations. Therefore, AIDA adopts symptom-based retrieval and a validation method capturing symptom coherence, leading to increased RCA precision.

Log severity, frequency, and temporal concentration are important signals for precise RCA. Our deployment experience reveals that disparate root causes can manifest through nearly identical log signatures, with distinction only in granular details, including counter magnitudes, error frequency, and temporal concentration of logs. For instance, the log template "`tx errors since last run TXErr=<*>`" may indicate either a negligible transient error or a critical transceiver hardware failure, depending on the reported numerical value's magnitude after "`TXErr=`". Driven by this RCA characteristic, we leverage LLMs with prompts augmenting these signals to interpret quantitative and temporal nuances, accurately resolving disparate root causes.

Continuous device monitoring for comprehensive RCA. Currently, AIDA performs RCA as a one-shot process by retrieving logs at the time of failure and conducting RCA based on that snapshot. However, accurate diagnosis often requires continuous monitoring of device status beyond the failure event, taking into account the state evolvement of the device. For example, a single restart from the device may indicate a software bug, whereas repeated restarts over time suggest an underlying hardware issue. Without ongoing observation, AIDA may misclassify hardware problems as software issues due to limited temporal context analysis. We leave incorporating continuous monitoring as future work.

Quality-centric KG construction improves RCA precision. We initially prioritized data quantity to maximize root cause type coverage. However, we found that 6.3% of our diagnostic records—primarily RCA misclassifications, unexplained claims, or unsubstantiated conclusions, accounted for most of AIDA's false positive diagnosis. When incorporated into the KG, they created erroneous associations between evidence logs and root causes. To address this, we integrated an LLM-based filtering agent into our pre-processing pipeline to exclude records lacking sufficient evidence or explanation. This shift toward a quality-centric approach significantly improved KG quality and reduced misdiagnosis.

7 Related Work

System-level fault localization. Various works focus on localizing faults in distributed cloud services and datacenter networks. Some approaches leverage graph models to infer fault causality. For instance, Murphy [14], Flock [13], and Sage [10] employ service dependency graphs and probabilistic models to trace anomalies back to faulty microservices or network components. Other works, such as Zeno [43] and ExplainIt [18] perform deeper causal reasoning by building temporal causal graphs from system traces. In parallel, network-oriented approaches [3, 28, 35, 38] leverage network measurements and statistical analysis to pinpoint problematic links or devices. Unlike these works, which primarily rely on performance metrics or predefined system dependencies, AIDA identifies fundamental root causes within physical hardware by semantically understanding device logs.

Automatic fault mitigation and recovery. Complementary to localization, efforts have also been made in automated mitigation to rapidly restore service. SWARM [25], CorrOpt [51], and NetPilot [42] aim to recommend, rank, or apply immediate recovery actions, such as rerouting traffic or disabling faulty links, to quickly restore service performance. Others recommend troubleshooting guides [19, 31], leveraging natural language processing to match incidents with documented solutions. While these works prioritize rapid mitigation, AIDA aims to precisely analyze root causes and ensure the long-term health of the network.

LLMs for network device RCA. Recent efforts have explored the application of LLMs to automate network fault management, primarily by building systems for active diagnosis [2, 4, 21, 26, 32, 36, 39, 45, 48]. For example, Ahmed *et al.* [2] fine-tune GPT models for RCA using only incident titles and summaries. RAG-based inference, such as RCACopilot [4], collect diagnostic data based on human-defined rules and retrieve historical diagnostic summaries to guide the model's analysis. Tool-using agents like RCAgent [39] empower LLMs to orchestrate and utilize external tools for complex cloud incident analysis. Complementary work enhances diagnostic utility and interpretability. Oasis [21] automates post-incident summarization and outage impact assessment to accelerate operator comprehension. LM-Pace [45] introduces a confidence estimation method to evaluate the trustworthiness of RCA outputs generated by LLMs. However, these approaches often produce coarse-grained or unverifiable reasoning, lacking the diagnostic precision and interpretability essential for production use.

8 Conclusion

We introduce AIDA, the first LLM-based system for automated, fine-grained, and interpretable RCA of network device failures in production networks. AIDA encodes vendor-proprietary diagnostic knowledge as structured *reasoning chains*, extracted from historical emails via supervised and reinforcement fine-tuning for KG construction. A multi-stage LLM inference pipeline, empowered by RAG, enables efficient log filtering and multi-step reasoning. Deployed in production for over one year, AIDA demonstrates accurate root cause identification and significant TRCA reduction, while providing interpretable reasoning that aligns with operator workflows and fosters trust in automated diagnosis.

Acknowledgments

We thank our shepherd and the SIGCOMM reviewers for their insightful comments. This work was supported by Alibaba Cloud through Alibaba Research Intern Program and Alibaba Innovative Research Program, National Key Research and Development Program of China (Grant No. 2025YFB4506600), NSFC (Grant Nos. 62472460 and U23B2004), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2024A1515010161), and Guangdong Major Project of Basic Research (Grant No. 2026B0303000007).

References

[1] 2024. Google Cloud Service Health. https://status.cloud.google.com/incidents/kDBRnSgQCPw93E8vKKat. (2024).

[2] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1737–1749.

[3] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically Finding the Cause of Packet Drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 419–435.

[4] 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. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 674–688.

[5] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[6] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[7] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[8] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[9] Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. 2025. Deep Think with Confidence. (2025). arXiv:cs.LG/2508.15260 https://arxiv.org/abs/2508.15260

[10] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: practical and scalable ML-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 135–151.

[11] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. LogBERT: Log Anomaly Detection via BERT. In *2021 International Joint Conference on Neural Networks (IJCNN)*. 1–8.

[12] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and Fast Retrieval-Augmented Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, 10746–10761.

[13] Vipul Harsh, Tong Meng, Kapil Agrawal, and Philip Brighten Godfrey. 2023. Flock: Accurate Network Fault Localization at Scale. *Proc. ACM Netw.* 1, CoNEXT1, Article 3 (2023), 22 pages.

[14] Vipul Harsh, Wenxuan Zhou, Sachin Ashok, Radhika Niranjan Mysore, Brighten Godfrey, and Sujata Banerjee. 2023. Murphy: Performance Diagnosis of Distributed Cloud Applications. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 438–451.

[15] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. 33–40.

[16] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* 43, 2 (2025), 1–55.

[17] Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 538–543.

[18] Vimalkumar Jeyakumar, Omid Madani, Ali Parandeh, Ashutosh Kulshreshtha, Weifei Zeng, and Navindra Yadav. 2019. ExplainIt! – A Declarative Root-cause Analysis Engine for Time Series Data. In *Proceedings of the 2019 International Conference on Management of Data*. 333–348.

[19] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1410–1420.

[20] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R Lyu. 2024. LILAC: Log Parsing using LLMs with Adaptive Parsing Cache. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 137–160.

[21] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, Shilin He, Federica Sarro, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Assess and Summarize: Improve Outage Understanding with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1657–1668.

[22] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.

[23] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Barcelona, Spain, 74–81.

[24] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.

[25] Pooria Namyar, Arvin Ghavidel, Daniel Crankshaw, Daniel S. Berger, Kevin Hsieh, Srikanth Kandula, Ramesh Govindan, and Behnaz Arzani. 2025. Enhancing Network Failure Mitigation with Performance-Aware Ranking. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*.

[26] 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. 2025. Flow-of-Action: SOP Enhanced LLM-Based Multi-Agent System for Root Cause Analysis. In *Companion Proceedings of the ACM on Web Conference 2025*. 422–431.

[27] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. (2025). arXiv:cs.CL/2412.15115 https://arxiv.org/abs/2412.15115

[28] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C. Snoeren. 2017. Passive Realtime Datacenter Fault Detection and Localization. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.

[29] Amrita Saha and Steven CH Hoi. 2022. Mining root cause knowledge from cloud service incident investigations for aiops. In *Proceedings of the 44th international conference on software engineering: Software engineering in practice*. 197–206.

[30] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. (2024). arXiv:cs.CL/2402.03300

[31] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. 2022. 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*. 1477–1488.

[32] Binpeng Shi, Yu Luo, Jingya Wang, Yongxin Zhao, Shenglin Zhang, Bowen Hao, Chenyu Zhao, Yongqian Sun, Zhi Zhang, Ronghua Sun, Haihua Li, Wei Song, Xiaolong Chen, Jingbo Miao, and Dan Pei. 2025. FlowXpert: Expertizing Troubleshooting Workflow Orchestration with Knowledge Base and Multi-Agent Coevolution. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*. 4839–4850.

[33] 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. 2023. LogKG: Log Failure Diagnosis Through Knowledge Graph. *IEEE Transactions on Services Computing* 16, 5 (2023).

[34] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. 2024. ART: A Unified Unsupervised Framework for Incident Management in Microservice Systems. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1183–1194.

[35] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 599–614.

[36] Chenxu Wang, Xumiao Zhang, Runwei Lu, Xianshang Lin, Xuan Zeng, Xinlei Zhang, Zhe An, Gongwei Wu, Jiaqi Gao, Chen Tian, Guihai Chen, Guyue Liu, Yuhong Liao, Tao Lin, Dennis Cai, and Ennan Zhai. 2025. Towards LLM-Based Failure Localization in Production-Scale Networks. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 496–511.

[37] Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin, Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu, Wei Zhou, Yongbin Dong, et al. 2024. {NetAssistant}: Dialogue based network diagnosis in data center networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2011–2024.

[38] Zhe Wang, Huanwu Hu, Linghe Kong, Xinlei Kang, Qiao Xiang, Jingxuan Li, Yang Lu, Zhuo Song, Peihao Yang, Jiejian Wu, Yong Yang, Tao Ma, Zheng Liu, Xianlong Zeng, Dennis Cai, and Guihai Chen. 2024. Diagnosing Application-network Anomalies for Millions of IPs in Production Clouds. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 885–899.

[39] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024. RCAgent: Cloud Root Cause Analysis by Autonomous Agents with Tool-Augmented Large Language Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4966–4974.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 35. 24824–24837.

[41] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. 2024. NetLLM: Adapting Large Language Models for Networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 661–678.

[42] Xin Wu, Daniel Turner, Chao-Chih Chen, David A. Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. 2012. NetPilot: automating datacenter network failure mitigation. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 419–430.

[43] Yang Wu, Ang Chen, and Linh Thi Xuan Phan. 2019. Zeno: Diagnosing Performance Problems with Temporal Provenance. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 395–420.

[44] Bo Yang, Huanwu Hu, Yifan Li, Yunguang Li, Xiangyu Tang, Bingchuan Tian, Gongwei Wu, Jianfeng Xu, Xumiao Zhang, Feng Chen, Cheng Wang, Ennan Zhai, Yuhong Liao, Dennis Cai, and Tao Lin. 2025. SkyNet: Analyzing Alert Flooding from Severe Network Failures in Large Cloud Infrastructures. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 512–526.

[45] Dylan Zhang, Xuchao Zhang, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. LM-PACE: Confidence Estimation by Large Language Models for Effective Root Causing of Cloud Incidents. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 388–398.

[46] Shenglin Zhang, Jun Zhu, Bowen Hao, Yongqian Sun, Xiaohui Nie, Jingwen Zhu, Xilin Liu, Xiaoqian Li, Yuchi Ma, and Dan Pei. 2024. Fault Diagnosis for Test Alarms in Microservices through Multi-source Data. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 115–125.

[47] Tianzhu Zhang, Han Qiu, Gabriele Castellano, Myriana Rifai, Chung Shue Chen, and Fabio Pianese. 2023. System log parsing: A survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 8 (2023), 8596–8614.

[48] Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. 2024. 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*. 266–277.

[49] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[50] Yongxin Zhao, Shenglin Zhang, Yuxin Sun, Wenwei Gu, Yongqian Sun, Luping Wang, Li Shi, Cheng Huang, Guodong Yang, Liping Zhang, et al. [n. d.]. When LLMs Listen to Experts: Accurate Failure Diagnosis in Operating Systems. ([n. d.]).

[51] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. 2017. Understanding and Mitigating Packet Corruption in Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 362–375.

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A Details of Graph-based Similarity Refinement

This appendix provides the formal mathematical details for the graph-based similarity refinement process described in Step 2 of §3.3.2. Our goal is to iteratively refine node similarity by balancing static content similarity with relational consistency derived from the graph structure, formalizing the intuition that *node equivalence requires both semantic similarity and relational consistency.*

A.1 Graph Representation and Notation

For each incident partition $P_m \in \mathcal{P}$, we construct a bipartite graph $B_m = (V_{\text{rc}} \cup V_{\text{ev}}, E)$, where V_{rc} is the set of root cause nodes and V_{ev} is the set of evidence nodes. An edge $(n, rc) \in E$ exists if the evidence node n and root cause node rc co-occur within the same reasoning chain. We use the following notation, with all similarity scores normalized to $[0, 1]$.

• **Neighbors $\mathcal{N}(u)$:** The set of nodes connected to node u. For any $rc \in V_{\text{rc}}$, its neighbors $\mathcal{N}(rc) \subseteq V_{\text{ev}}$, and vice versa.

• **Content similarity matrices ($\mathbf{S}_{\text{rc,cont}}, \mathbf{S}_{\text{ev,cont}}$):** These static matrices store similarities based purely on node content, corresponding to the $S_{\text{cont}}(u, v)$ term in the main text. For a root cause pair (rc_i, rc_j), $(\mathbf{S}_{\text{rc,cont}})_{ij}$ is their summary's semantic similarity. For an evidence pair (n_k, n_l), $(\mathbf{S}_{\text{ev,cont}})_{kl}$ is the minimum of their symptom description similarity and log-based lexical similarity.

• **Cross-type similarity matrix ($\mathbf{S}_{\text{ev,rc}}$):** A static matrix where $(\mathbf{S}_{\text{ev,rc}})_{ki} = \text{sim}(n_k, rc_i)$ captures the content-based semantic linkage *between* an evidence node n_k and a root cause node rc_i.

A.2 Iterative Refinement Algorithm

The iterative updates presented here are the matrix formulation of Eq. (2). At each step t, we update the root cause similarity matrix $\mathbf{S}_{\text{rc}}^{(t)}$ and evidence similarity matrix $\mathbf{S}_{\text{ev}}^{(t)}$ as follows:

$$\mathbf{S}_{\text{rc}}^{(t+1)} = (\mathbf{1} - \mathbf{W}_{\text{rc}}) \odot \mathbf{S}_{\text{rc,cont}} + \mathbf{W}_{\text{rc}} \odot \mathbf{S}_{\text{rc,prop}}^{(t)}, \tag{5}$$

$$\mathbf{S}_{\text{ev}}^{(t+1)} = (\mathbf{1} - \mathbf{W}_{\text{ev}}) \odot \mathbf{S}_{\text{ev,cont}} + \mathbf{W}_{\text{ev}} \odot \mathbf{S}_{\text{ev,prop}}^{(t)}, \tag{6}$$

where $\odot$ denotes the element-wise product. The matrix components $\mathbf{W}$ and $\mathbf{S}_{\text{prop}}$ directly correspond to the scalar terms w_{uv} and $S_{\text{prop}}^{(t)}(u, v)$ from the main text, and are detailed below.

A.2.1 The Propagated Similarity Matrix. The propagated similarity matrix $\mathbf{S}_{\text{prop}}^{(t)}$ contains the scores $S_{\text{prop}}^{(t)}(u, v)$ that capture relational consistency. Its elements are computed as a weighted average of neighbor similarities from the previous step. For a root cause pair (rc_i, rc_j), the element $(\mathbf{S}_{\text{rc,prop}}^{(t)})_{ij}$ is calculated as:

$$(\mathbf{S}_{\text{rc,prop}}^{(t)})_{ij} = \begin{cases} \hat{S}_{ij}, & |\mathcal{N}(rc_i)|\,|\mathcal{N}(rc_j)| > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

$$\hat{S}_{ij} = \frac{\sum_{n_k \in \mathcal{N}(rc_i)} \sum_{n_l \in \mathcal{N}(rc_j)} \omega(rc_i, rc_j, n_k, n_l)\, (\mathbf{S}_{\text{ev}}^{(t)})_{kl}}{\sum_{n_k \in \mathcal{N}(rc_i)} \sum_{n_l \in \mathcal{N}(rc_j)} \omega(rc_i, rc_j, n_k, n_l)}, \tag{8}$$

where the relational consistency weight ω is:

$$\omega(rc_i, rc_j, n_k, n_l) = \sqrt{(\mathrm{S}_{\mathrm{ev,rc}})_{ki} \cdot (\mathrm{S}_{\mathrm{ev,rc}})_{lj}}. \tag{9}$$

The matrix $\mathrm{S}^{(t)}_{\mathrm{ev,prop}}$ is defined symmetrically.

Intuition. The weight ω quantifies how semantically aligned a neighbor pair (n_k, n_l) is with the target pair (rc_i, rc_j). By using this weight, the propagation mechanism prioritizes similarity propagation along paths that are both structurally and semantically coherent, effectively filtering out noise from irrelevant neighbor pairs.

A.2.2 The Adaptive Weight Matrix. The adaptive weight matrix $\mathbf{W}$ contains the scalar weights w_{uv} from Eq. (2). For a root cause pair (rc_i, rc_j), the element $(\mathbf{W}_{\mathrm{rc}})_{ij}$, corresponding to w_{rc_i,rc_j}, is:

$$(\mathbf{W}_{\mathrm{rc}})_{ij} = \min\left(\sqrt{\bar{\sigma}(rc_i)\,\bar{\sigma}(rc_j)},\ \lambda_{\mathrm{rc}}\right), \tag{10}$$

where λ_{rc} is a clipping hyperparameter and $\bar{\sigma}(u)$ quantifies the node's **specificity**:

$$\bar{\sigma}(rc_i) = \begin{cases} \dfrac{1}{|\mathcal{N}(rc_i)|} \displaystyle\sum_{n_k \in \mathcal{N}(rc_i)} (\mathrm{S}_{\mathrm{ev,rc}})_{ki}, & |\mathcal{N}(rc_i)| > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

The matrix $\mathbf{W}_{\mathrm{ev}}$ is defined analogously for evidence node pairs.

Intuition. A node's specificity $\bar{\sigma}$ measures its average semantic connectivity to its neighbors. The geometric mean in Eq. (10) ensures that relational influence is amplified only when *both* nodes are specific, steering the refinement process with the most informative diagnostic signals.

B Derivation and Estimation of the Confidence Score

This appendix provides the formal probabilistic derivation for the confidence score introduced in §3.4. The goal is to define how we rank root-cause candidates. We begin by modeling the confidence for a single *reasoning trace* (§B.1) and then describe how to aggregate scores from multiple traces (§B.2). Finally, we describe how the model parameters, namely the reliability of the underlying static chain and the LLM-inferred confidence in the dynamic evidence, are estimated (§B.3, §B.4).

B.1 Confidence Model for a Single Reasoning Chain

For each validated reasoning trace $T_i \in \mathcal{T}_{\mathrm{val}}$, we compute its confidence score, $\gamma(T_i)$. Each trace T_i is a dynamic instantiation of an underlying static reasoning chain, which we denote as C_i. The trace T_i contains the set of supporting log segments $\boldsymbol{\ell}^{(i)} = (\ell^{(i)}_1, \ldots, \ell^{(i)}_{k_i})$ collected from the current incident, used to validate the evidence nodes of C_i. This score represents the probability that the root cause proposed by the underlying chain, $rc(C_i)$, is the ground-truth root cause (rc_g), given the evidence collected in the trace:

$$\gamma(T_i) = P(rc_g = rc(C_i) \mid \boldsymbol{\ell}^{(i)}) \tag{12}$$

To make this computation tractable, we introduce two latent variables that explicitly separate the properties of the trace from its underlying chain:

• **Evidence correctness ($\mathbf{H}^{(i)}$)**: We formally define a binary vector as $\mathbf{H}^{(i)} = (H^{(i)}_1, \ldots, H^{(i)}_{k_i})$. $H^{(i)}_j = 1$ if the j-th evidence node of chain C_i is factually correct for the current incident (*i.e.,* within trace T_i), and 0 otherwise.

• **Chain reliability ($Z^{(i)}$)**: A binary variable. $Z^{(i)} = 1$ if the static reasoning chain C_i represents a logically valid inference path, and 0 otherwise. This captures the intrinsic quality of the chain's logic, independent of the current incident.

Using the law of total probability, we can expand Eq. (12) by marginalizing over all possible states of $\mathbf{H}^{(i)}$ and $Z^{(i)}$:

$$\gamma(T_i) = \sum_{\mathbf{h} \in \{0,1\}^{k_i}} \sum_{z \in \{0,1\}} P(rc_g = rc(C_i), \mathbf{H}^{(i)} = \mathbf{h}, Z^{(i)} = z \mid \boldsymbol{\ell}^{(i)}) \tag{13}$$

By applying the chain rule of probability, we can factor this expression:

$$\begin{aligned} \gamma(T_i) = \sum_{\mathbf{h}, z} & P(rc_g = rc(C_i) \mid \mathbf{H}^{(i)} = \mathbf{h}, Z^{(i)} = z, \boldsymbol{\ell}^{(i)}) \\ & \cdot P(Z^{(i)} = z \mid \mathbf{H}^{(i)} = \mathbf{h}, \boldsymbol{\ell}^{(i)}) \cdot P(\mathbf{H}^{(i)} = \mathbf{h} \mid \boldsymbol{\ell}^{(i)}) \end{aligned} \tag{14}$$

This expression can be simplified by introducing two conditional independence assumptions:

• **Assumption 1**. The final conclusion's correctness depends on the validity of the reasoning steps ($Z^{(i)}$) and the truthfulness of the evidence ($\mathbf{H}^{(i)}$), rather than on the raw logs ($\boldsymbol{\ell}^{(i)}$) themselves. Therefore, $P(rc_g | \mathbf{H}^{(i)}, Z^{(i)}, \boldsymbol{\ell}^{(i)}) = P(rc_g | \mathbf{H}^{(i)}, Z^{(i)})$.

• **Assumption 2**. A reasoning chain's intrinsic reliability ($Z^{(i)}$) is a historical property and does not depend on the specific logs of a new case instance, given the evidence correctness pattern $\mathbf{H}^{(i)}$. Thus, $P(Z^{(i)} | \mathbf{H}^{(i)}, \boldsymbol{\ell}^{(i)}) = P(Z^{(i)} | \mathbf{H}^{(i)})$.

Applying these assumptions, the equation simplifies to:

$$\begin{aligned} \gamma(T_i) = \sum_{\mathbf{h}, z} & P(rc_g = rc(C_i) \mid \mathbf{H}^{(i)} = \mathbf{h}, Z^{(i)} = z) \\ & \cdot P(Z^{(i)} = z \mid \mathbf{H}^{(i)} = \mathbf{h}) \cdot P(\mathbf{H}^{(i)} = \mathbf{h} \mid \boldsymbol{\ell}^{(i)}) \end{aligned} \tag{15}$$

This formulation is further reduced based on the following modeling assumptions:

• **Assumption 3:** A reasoning trace T_i yields the correct root cause if and only if (1) all its constituent evidence is true ($\mathbf{H}^{(i)} = \mathbf{1}$) and (2) its underlying reasoning chain C_i is logically reliable ($Z^{(i)} = 1$). This is a conservative assumption implying that any single weak link—either a piece of false evidence or a flawed logical step—invalidates the entire chain.

$$P(rc_g = rc(C_i) \mid \mathbf{H}^{(i)} = \mathbf{h}, Z^{(i)} = z) = \begin{cases} 1 & \text{if } \mathbf{h} = \mathbf{1} \text{ and } z = 1 \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

• **Assumption 4:** The correctness of each evidence node $H^{(i)}_j$ in the trace is conditionally independent of other evidence nodes, given its corresponding log segment $\ell^{(i)}_j$. This allows us to factor

the evidence probability:

$$P(\mathbf{H}^{(i)} = \mathbf{h} \mid \boldsymbol{\ell}^{(i)}) = \prod_{j=1}^{k_i} P(H_j^{(i)} = h_j \mid \ell_j^{(i)}) \tag{17}$$

Under Assumption 3, the term $P(rc_g = rc(C_i)|\cdots)$ is non-zero only when all evidence holds ($\mathbf{h} = \mathbf{1}$) and the chain is reliable ($z = 1$). Consequently, the summation in Eq. (15) collapses to a single non-zero term for any validated trace $T_i \in \mathcal{T}_{\text{val}}$:

$$\gamma(T_i) = \underbrace{P(Z^{(i)} = 1 \mid \mathbf{H}^{(i)} = \mathbf{1})}_{\text{Chain Reliability}} \cdot \underbrace{\prod_{j=1}^{k_i} P(H_j^{(i)} = 1 \mid \ell_j^{(i)})}_{\text{LLM-inferred Evidence Confidence}} . \tag{18}$$

Eq. (18) provides the probabilistic formulation underlying the simplified confidence score in Eq. (4). Specifically:

- The **Chain reliability** term, $\text{Rel}(C_i)$, in Eq. (4) corresponds to $P(Z^{(i)} = 1|\mathbf{H}^{(i)} = \mathbf{1})$, which captures the estimated reliability of the static chain C_i.
- The **LLM-inferred evidence confidence**, $\text{Conf}(n_j^{(i)}|\ell_j^{(i)})$, in Eq. (4) corresponds to $P(H_j^{(i)} = 1|\ell_j^{(i)})$, which quantifies confidence that the evidence node $n_j^{(i)}$ is supported by the observed logs $\ell_j^{(i)}$ within the dynamic trace T_i.

This design ensures our scoring scheme prioritizes conclusions from reliable reasoning chains that are strongly supported by LLM-validated evidence from the current reasoning trace.

B.2 Aggregating Confidence from Multiple Traces

A root cause candidate, rc_a, may be suggested by multiple reasoning traces, denoted by the set $\mathcal{T}_a = \{T_i \in \mathcal{T}_{\text{val}} \mid rc(C_i) = rc_a\}$. We aggregate their individual scores into a consolidated score, $\Gamma(rc_a)$, which represents the probability that *at least one* of these traces is correct. To aggregate their scores, we assume the traces are independent events for tractability, and calculate the probability of at least one chain being correct (*i.e.,* one minus the probability that all of them fail):

$$\Gamma(rc_a) = 1 - \prod_{T_i \in \mathcal{T}_a} (1 - \gamma(T_i)) \tag{19}$$

B.3 Estimation of Reasoning Chain Reliability

The reasoning chain reliability term, $P(Z^{(i)} = 1 \mid \mathbf{H}^{(i)} = \mathbf{1})$, is estimated from historical data for each static reasoning chain C_i. To ensure robustness against sparse data, we use Bayesian smoothing. This approach allows us to assign a reasonable baseline reliability to all chains, which then converges towards an empirically justified value as the chain is applied more frequently across historical incidents. Our model is based on the assumption that *chains that have appeared more frequently in the past are inherently more reliable.* The reliability is estimated as:

$$\text{Rel}(C_i) = P(Z^{(i)} = 1 \mid \mathbf{H}^{(i)} = \mathbf{1}) = \frac{A \cdot p_0 + m^{(i)}}{A + m^{(i)}} \tag{20}$$

where $m^{(i)}$ is the number of times chain C_i has appeared in historical diagnostic cases. We set the prior belief $p_0 = 0.5$ and the smoothing factor $A = 1$, a form of Laplace smoothing. This assigns a neutral 0.5 reliability to new chains and ensures the score rapidly increases with each historical use. Consequently, well-established chains are trusted more, while new ones are not unfairly penalized.

B.4 Estimation of Evidence Verification Confidence

The term $P(H_j^{(i)} = 1 \mid \ell_j^{(i)})$ in Eq. (18) represents the confidence that the hypothesis of an evidence node $n_j^{(i)}$ (from chain C_i) is factually supported by the log segment $\ell_j^{(i)}$ (within trace T_i). We estimate this by quantifying the LLM's intrinsic certainty based on the output probability distributions over candidate tokens at each generation step [9].

First, we compute a raw certainty score, $\zeta_{\text{raw}}(n_j^{(i)})$, by averaging a token-level certainty metric across the entire response:

$$\zeta_{\text{raw}}(n_j^{(i)}) = \frac{1}{L_j} \sum_{t=1}^{L_j} \left(\ln(k) + \sum_{w \in \mathcal{V}_t} P_t(w) \ln P_t(w) \right) \tag{21}$$

where L_j is the length in tokens of the response, $\mathcal{V}_t$ is the set of top-k candidate tokens at step t, and $P_t(w)$ is the model's predicted probability for token $w \in \mathcal{V}_t$. This score is higher when the model's token predictions are less uniform (*i.e.,* more "certain").

We normalize the score ζ_{raw} into $[0, 1]$ by dividing it by its theoretical maximum. This serves as our final confidence estimate:

$$P(H_j^{(i)} = 1 \mid \ell_j^{(i)}) = \frac{\zeta_{\text{raw}}(n_j^{(i)})}{\ln(k)} \tag{22}$$

C Experiment Results For Traditional ML-based RCA

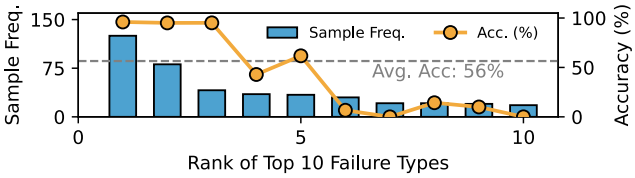


Figure 26: A standard BERT classifier performs poorly under the long-tail failure distribution, achieving 96% accuracy on the most frequent type but under 10% for less frequent types within the top-10, yielding an overall accuracy of only 56%.

D Problem of Conventional Log-Parsing Methods

Oversimplified log preprocessing: log-based anomaly detection methods [7, 11, 33, 34, 49] lack semantic understanding for complex cross-event correlation. They rely on log parsing [6, 15], which abstracts dynamic variables into wildcards (*e.g.,* potentially replacing device states or specific modules), causing significant information loss.

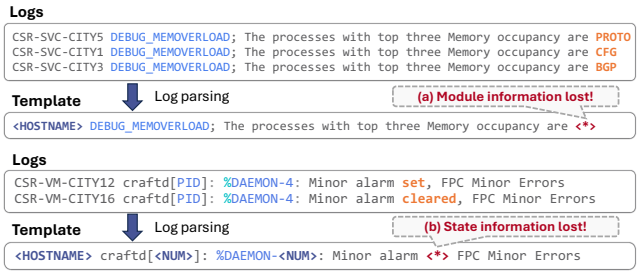


Figure 27: Conventional log-parsing methods often discard key tokens during preprocessing of new inference cases, such as (a) module identifiers and (b) state keywords, hindering fine-grained RCA in subsequent RAG-based LLM inference.

E Additional Deployment Lessons

Log-analysis-free root cause identification with KG. We realized that with the support of knowledge graph, we can efficiently identify the root cause even without log analysis for some cases. This happens when all the historical cases with problem descriptions similar to the new case have the same trivial (low severity) root cause and the number of historical cases is large. In this case, we can conclude that the new fault has the same root cause without further investigating the associated logs. This helps to eliminate the time and effort spent on collecting logs by our operational engineers.

Ensuring Reliability in Automated RCA. Erroneous root cause identification remains a major risk in automated RCA system, as incorrect diagnoses can even prolong service disruptions. This implies the need for trustworthiness in diagnostic results. Our design emphasizes trustworthiness by adopting a multi-step reasoning framework that enforces rigorous validation at each step before drawing conclusions. This reduces the risk of unsupported conclusions from partial evidence, often seen in holistic single-step reasoning. During the initial deployment, our emphasis on trustworthiness also helped establish trust with operators. However, as shown in Figure 14, this strict validation hurts accuracy and limit the number of failures that AIDA can effectively handle. Future work can explore ways to improve precision without compromising recall.