Towards LLM-Based Failure Localization in Production-Scale Networks

Chenxu Wang[1,2], Xumiao Zhang[2], Runwei Lu[2,3], Xianshang Lin[2], Xuan Zeng[2], Xinlei Zhang[2], Zhe An[2], Gongwei Wu[2], Jiaqi Gao[2], Chen Tian[1], Guihai Chen[1], Guyue Liu[4], Yuhong Liao[2], Tao Lin[2], Dennis Cai[2], Ennan Zhai[2]

[1]Nanjing University [2]Alibaba Cloud [3]New York University Shanghai [4]Peking University

Abstract

Root causing and failure localization are critical to maintain reliability in cloud network operations. When an incident is reported, network operators must review massive volumes of monitoring data and identify the root cause (*i.e.,* error device) as fast as possible, making it extremely challenging even for experienced operators. Large language models (LLMs) have shown great potential in text understanding and reasoning. In this paper, we present BiAn, an LLM-based framework designed to assist operators in efficient incident investigation. BiAn processes monitoring data and generates error device rankings with detailed explanations. To date, BiAn has been deployed in our network infrastructure for 10 months and it has successfully assisted operators in identifying error devices more quickly, reducing *time to root causing* by 20.5% (55.2% for high-risk incidents). Extensive performance evaluations based on 17 months of real cases further demonstrate that BiAn achieves accurate and fast failure localization. It improves accuracy by 9.2% compared to the baseline approach.

CCS Concepts

• **Networks** → **Data center networks**; **Network management**; **Network monitoring**; **Network reliability**.

Keywords

Data Center Networks, Network Troubleshooting, Incident Management, Root Cause Analysis, Large Language Model, AIOps

ACM Reference Format:

Chenxu Wang[1,2], Xumiao Zhang[2], Runwei Lu[2,3], Xianshang Lin[2], Xuan Zeng[2], Xinlei Zhang[2],, Zhe An[2], Gongwei Wu[2], Jiaqi Gao[2], Chen Tian[1], Guihai Chen[1], Guyue Liu[4], Yuhong Liao[2], Tao Lin[2], Dennis Cai[2], Ennan Zhai[2]. 2025. Towards LLM-Based Failure Localization in Production-Scale Networks. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3718958.3750505

1 Introduction

As one of the largest public cloud providers, Alibaba Cloud serves millions of customers all over the world 24×7. We build and operate a global network infrastructure consisting of a wide area network (WAN) and tens of interconnected data centers. Network incidents are, nevertheless, inevitable given the scale of our operations and can significantly impact reliability, potentially leading to revenue loss. To ensure efficient and effective network incident management, we have developed a comprehensive monitoring system, benefiting from recent advances in network telemetry and monitoring [7, 22, 41, 62, 90]. The system initially attempts to self-heal using predefined rules and automated solutions based on operators' years of experience. However, not all incidents are straightforward enough to be directly addressed—complex cases still require manual investigation, in particular for identifying root causes and locating failures. Mitigation actions are taken afterward. Incident investigation is the most time-consuming and cognitively demanding in the workflow.

When an incident occurs, operators carefully review massive logs from diverse monitoring tools, understand interdependencies between anomalous events as well as affected devices, analyze the root cause, and finally identify the faulty device (see Figure 3 for a detailed case). Due to time constraints, oftentimes they need to finish the investigation before thoroughly reviewing all available logs. The urgency of an incident increases as time goes by. Consequently, the efficiency and accuracy are often bottlenecked by operators' limited capacity to process large amounts of information. In addition, as our network infrastructure evolves, it becomes increasingly challenging for operators to perform fast and accurate failure localization. Therefore, we aim to design a tool to assist in the reasoning process of incident investigation.

Large language models (LLMs) have demonstrated superior capabilities in efficiently reading and reasoning over large-scale data, particularly in handling complex tasks. Leveraging LLMs to understand textual data from monitoring tools and assist operators in failure localization presents a promising direction. LLMs surpass traditional automated methods in logical reasoning and generalizability for unseen incidents. Previous research on applying LLMs in network operations [5, 16, 33, 65, 81] either is limited to coarse-grained analysis, still requiring operators to identify root causes; or utilizes only partial information (*e.g.,* incident summaries), which oversimplifies failure localization (§2.3).

We present BiAn[1], a practical and comprehensive system for accurate and fast root causing and failure localization in Alibaba Cloud's network infrastructure. BiAn employs LLM agents powered by carefully crafted prompts to understand network monitoring logs, extract key information, reason through incidents, and assist operators by providing error device rankings with explanations. BiAn achieves its goal through the following design aspects:

[1]In ancient Chinese mythology, Bi An is a divine beast known for its strong sense of justice and ability to distinguish right from wrong. We named our system BiAn to reflect its role in intelligently analyzing root causes and identifying error devices in complex network environments.

SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/2025/09
https://doi.org/10.1145/3718958.3750505

Hierarchical reasoning for large data scale (§4.1). Facing the logs generated by various monitors for different affected devices, not only is it impractical for human operators to analyze based on all the data, but also LLMs struggle to capture the most relevant information at once. To tackle this, BiAn approaches the localization task through three separate steps: (1) It first summarizes monitor alerts into simplified alert reports. (2) Then, it groups these reports by device and conducts device-level anomaly analysis following standard operating procedures (SOPs) compiled from years of incident handling experience by operators. (3) Finally, BiAn combines all single-device analysis results to derive failure scores for candidate devices based on their suspicion levels.

Multi-pipeline integration to tackle complexity (§4.2). In large-scale networks with countless devices interconnected, an incident caused by one device can spread across a number of logically nearby devices, all of which may present anomalous behaviors at different times. To address the *spatial* and *temporal* complexity, we include two additional dimensions of data, network topology and event timeline, into the original processing pipeline. By integrating these three pipelines, BiAn performs more comprehensive reasoning on incidents and generates scores with more informed explanations.

Continuous updating to cope with network evolution (§4.3). As our network components and configurations rapidly evolve, operators need to learn from historical incident handling and keep their operating procedures and knowledge up-to-date. The same applies to BiAn. We propose to continuously "train" the prompts for reasoning tasks. Specifically, we use LLMs' capabilities of generation, reflection, and summarization to extract useful knowledge from historical incidents. We then enrich the task prompts with this knowledge digest to enhance BiAn's reasoning performance.

Optimizations for practical consideration (§4.4). To make BiAn practically useful in assisting operators with failure localization, it needs to be fast (real-time) and resource-efficient. We utilize smaller LLMs to conduct simpler tasks, such as monitor alert summary and single-device analysis, and apply fine-tuning to incorporate domain-specific knowledge. In this way, we can achieve both improved localization accuracy and reduced reasoning latency. We also devise an early stop mechanism to prevent redundant processing when BiAn has already obtained high confidence after the initial stage of processing. Additionally, we enable parallel execution for the same level of agents to run simultaneously.

BiAn has been deployed in our global network infrastructure for almost a year (10 months) and has proven its ability to *assist* operators in network incident investigation. We present A/B tests, feedback from operators, and representative real cases, to demonstrate its effectiveness (§5). Besides, with real incident data collected over the past 17 months, we conduct extensive offline experiments to comprehensively evaluate the performance of BiAn across various aspects (§6). We discuss several lessons learned through the design, deployment, and evaluation of BiAn (§7).

Ethics. *This work does not raise any ethical issues.*

2 Background and Motivation

Alibaba Cloud operates a global network infrastructure that supports a wide range of cloud services, including computing, storage, and AI training/inference. By January 2025, this infrastructure spans 87 data centers across 29 geographical regions in North America, Europe, Asia, and Oceania. A private WAN serves as the backbone, interconnecting all these data center networks. Our network operations center (NOC) acts as the guardian for all network traffic. As incidents in the physical network can significantly impact overall reliability, which is our top priority, network incident management (IM) is a key focus for the team.

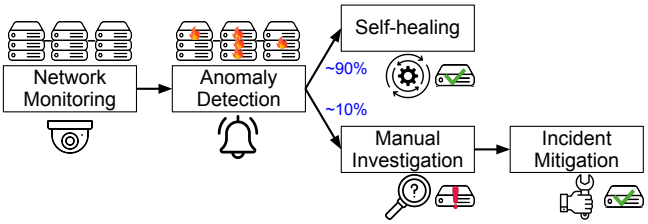


Figure 1: Network incident management workflow.

2.1 Incident Management Workflow

Figure 1 illustrates our operators' IM workflow, consisting of several steps: network monitoring, anomaly detection, self-healing, incident investigation, and incident mitigation. Over the years, we have developed a sophisticated network monitoring system equipped with a variety of tools, including foundational diagnostic utilities like `ping` and `traceroute`, statistical traffic analysis techniques, and more advanced solutions adapted from state-of-the-art efforts [7, 22, 41, 62, 90]. They process raw monitoring data and generate alerts for anomalous behaviors. When anomalies are detected, the system first mitigates them via predefined rules and automated solutions, which is called self-healing. However, in a large-scale network, there are always issues that are too complex to be directly addressed by the system or that are previously unseen. The system reports them as "*incidents*" and on-call operators get involved. In a typical week, 2377 anomalies can be self-healed while 202 are handled by our operators (*i.e.,* incidents).

2.2 Failure Localization and Limitations

When complex incidents bypass self-healed, operators step in to further investigate. For effective mitigation, they need to find the root cause of the incident and locate the failure (*i.e.,* error device). In this case, the system is more like providing preliminary "clues" or "recommendations", rather than definitive localization results. Operators continue with outputs from the upstream monitoring and diagnosis systems.

Why failure localization is hard for the operators? The manual investigation is an iterative reasoning process. Operators first review the incident description generated by the system. Then, they pick a few monitoring data types and retrieve the corresponding alerts. Initial analyses rarely yield clear conclusions. Thus, they continue to obtain and analyze additional types of data until they have sufficient confidence in identifying the root cause and error device. Based on our experience, this process is challenging for our operators due to the following reasons:

① **Excessive alerts**: An incident can trigger anomalies of varying degrees across several to hundreds of devices in the network, each potentially generating hundreds of alerts. The total size of relevant monitor alerts for an incident can exceed 1 GB (8k log entries), with an average of 26.4 MB, as shown in Figure 2.

② **Complex root causing**: Under time pressure, it is impractical for operators to analyze based on all available information. The

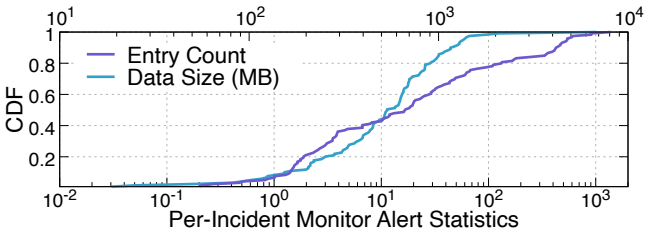


Figure 2: Large volume of monitoring data.

root causes of many failures are complex, and there is no fixed rule for the investigation. It often takes more than 10 minutes to identify a root cause, exceeding our service-level agreements (SLAs). The longest cases may take over half an hour.

③ **High incident load**: As mentioned, operators need to handle a large number of incidents (*e.g.,* 202 incidents in a week). As a result, the efficiency and accuracy of failure localization become heavily dependent on the expertise and experience of operators.

Taking the incident in Figure 3 as an example. The monitoring system detected an alert of significant packet loss in a data center, which resulted in multiple incidents being reported within 4 minutes, though merged later. These include flapping between core routers $B2 - A1$ and $B3 - A1$ (L2, L3 in the Response Log), an unreachable router $B2$ (L6), and an ICMP-related incident (L8). Tens of high-risk alerts appeared over time. Operators first suspected service updates (L12), which were ruled out after checking update histories (L14). Next, they shifted focus to traffic drops. After investigation, the root cause was identified as $B2$ (L17). To confirm $B2$ was not affected by neighboring devices like $B3$, active probing was conducted (L18). 22 minutes after the first alert, they finished the mitigation, by isolating $B2$ (L20). Despite immediate attention, operators had to review alerts of affected devices, engage in extensive discussions, and go through trial and error. Faced with the complication of massive alerts, it is challenging for them to pinpoint the root cause methodically.

A major difficulty in this process is the conflict between the massive amount of information generated by the monitoring system and the limited capacity of operators to consume it. As the network scales up, the challenges ahead only grow.

Our goal. We aim to develop a system that facilitates fast, accurate root causing and failure localization for our large-scale network infrastructure (including WAN and data center networks). Note that we are not replacing any existing monitors or data analytic algorithms. Instead, we intend this system as an assistant to our operators during incident investigation. The IM workflow shown in Figure 1 remains the same.

2.3 Opportunities with LLMs

Recently, large language models (LLMs) such as GPT [3], Claude [2], Llama [64], Gemini [63], and Qwen [10], trained with large text corpora, have demonstrated superior capabilities in text reading and reasoning. The outputs of upstream monitors and data processing tools are mainly textual, and are designed for human operators to interpret. Besides, unlike humans, LLMs are not prone to mental stress and can operate continuously and efficiently. These present a promising direction to assist in the reasoning process.

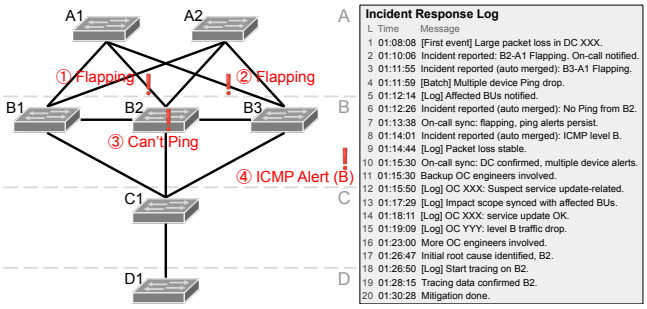


Figure 3: A motivating example (partial topology).

One might ask: *why not rely solely on traditional automated methods (e.g., heuristic-based, statistical)?* While such methods are valuable and already help mitigate countless anomalies, they are often based on specific metrics or historical patterns, limiting their ability to generalize to previously unseen cases. Extending those tools to handle new cases is expensive, and achieving 100% coverage is unrealistic. By contrast, similar to human operators, LLMs excel at logical reasoning and possess greater generalization capabilities. Thus, LLMs can serve as a bridge, complementing existing tools by assisting operators in incidents that require a holistic analysis of available information to identify root causes confidently. We present a detailed comparison between LLM-based and traditional methods later in §6.1.

We are not the first to explore the use of LLMs in networking [5, 16, 33, 56, 65, 69, 72, 81, 84, 86]. However, prior works on physical network operations are limited in several ways. Some focus on monitoring or only provide coarse-grained root cause analysis: MonitorAssistant [81] generates guidance-oriented anomaly reports. Oasis [33] assesses the impact scope of outages and produces summaries. RCACopilot [16] outputs the root cause category with explanations after triggering active probing. For the incident in Figure 3, these tools can only return a summary listing the range of affected devices with suspect information rather than directly pointing out the specific culprit, $B2$. The operators still have much work to do. NetAssistant [65] recognizes diagnosis intent and follow predefined workflows to process queries, leaving the investigation task to network users/operators. Ahmed *et al.* [5] study GPT's capabilities in producing root causes and mitigation steps from incident titles and digests. Compared to using detailed logs, the lack of deep visibility into incidents cannot lead to fully informed decisions. In Figure 3, this type of solution cannot pinpoint $B2$ based on auto-generated, low-quality incident digests. We thus decide to build an LLM-powered root causing and failure localization system to meet our goal.

2.4 Challenges

Building such a system poses the following challenges:

Large volume and diversity of data. As mentioned in §2.2, an incident can affect many network devices, and the monitoring system generates logs of dozens of types. Hence, the system must efficiently extract relevant information necessary for accurate localization from this sea of data.

Complex device and event relationships. Affected devices may span different parts of the network and trigger events at different times. The spatial (*i.e.,* topological) and temporal interdependencies

among devices and events complicate the process of tracing the original error device.

Need for fast investigation. In production environments, it is crucial to detect and fix incidents as quickly as possible. Unsuccessful incident investigation, whether due to slow responses or incorrect diagnoses, can lead to significant revenue loss and harm the provider's reputation. For example, major cloud service outages can cost companies $1 million per hour as reported [1].

Rapid evolution of network infrastructures. Modern network infrastructures are highly dynamic, with frequent updates to components and topologies. This changing environment makes it challenging to maintain consistent performance over time and calls for an adaptable solution that continuously learns from changes in network configurations as well as past incident investigations to stay effective.

Result explainability and consistency. The main purpose of building this automated system is to "assist" operators rather than to fully replace them. Therefore, its outputs need to come with explanations so that operators can easily understand and trust. Besides, the inherent randomness of LLMs should be minimized to prevent unpredictable changes in outputs that can disrupt operators' decision-making process.

3 Overview

We propose BiAn, a practical LLM-powered root causing and failure localization system for cloud network operations. Acting as an assistant in incident investigation, BiAn uses LLMs to understand monitor alerts, reason about root causes, and identify error devices. This approach mirrors the thinking and handling processes of network operators during the investigation. Figure 4 shows the architecture of BiAn.

When an incident is reported, BiAn takes as input monitor alerts for candidate devices and predicts the probabilities of each device being the actual error device, utilizing its internal knowledge about network incidents and reasoning capability. The candidates are the most suspicious devices, selected by the upstream automated monitoring system.[2] Since the alert data may contain redundancy and noise due to the logging formatting, we first preprocess it to remove empty items (*e.g.,* keys with no values) and unused fields. Then, LLM agents that we build following the standard operating procedures (SOPs) established by operators, analyze and reason about the incident. The output of BiAn is a ranked list of candidate devices with explanations for operators to review. Compared to a single answer, this ranked list adds fault tolerance: operators can investigate secondary devices when the top-ranked is incorrect (see §6.1 for more discussion). We reiterate that BiAn is not designed to replace human operators, but to significantly reduce distractions from irrelevant information. Operators make the final decisions for incident mitigation.

Originally, operators would have to quickly go through system-generated incident summaries and extensive monitoring data. Although the monitoring system provides key information for localization, it hides in the flood of alerts. Due to limited information processing capacity, operators usually spend considerable time filtering out irrelevant alerts. The traditional workflow requires operators to carefully read vast amounts of data, logically reason step by step to identify the error device, and make the final decision. With BiAn, the search space is greatly reduced, and the entire process is transformed to reviewing BiAn's output (*i.e.,* device ranks and explanations), validating the results by selectively checking relevant data sources, and making more informed decisions, faster (see Figure 8a for detailed comparisons).

[2]The system has a default sorting mechanism that considers the volume and severity of alerts. The number of candidates is empirically set (*e.g.,* 6).

4 Design

This section introduces the key components of BiAn and describes how it addresses each challenge outlined in §2.4.

4.1 Hierarchical Reasoning

The current incident investigation (§2.2) is primarily driven by human operators. During an incident, the monitoring system generate thousands of alerts, making it impossible for operators to review everything in time. While LLMs are more capable of reading efficiently, they too face limitations: LLMs have token limits as they cannot absorb too much information at a time; and they can be distracted by irrelevant information. Our approach is to process different types of alerts for different devices separately and breaks down the investigation process into structured steps: monitor alert summary, single-device anomaly analysis, and joint scoring.

Monitor alert summary. We first utilize LLMs to process incident-related alert data produced by 11 upstream monitoring tools, as listed in Figure 5 (left). These tools are developed for various purposes and have been refined over the years of operation, to comprehensively monitor network health and performance. When certain thresholds for indicators such as climbs, slumps, spikes, or dips in time series data are met, the corresponding monitor generates a log item. For instance, the "Device Ping Log" tool tracks dropped `Ping` packets between devices and records details such as device name, dropped `Ping` count, timestamps, and device role.

In BiAn, each type of alert is processed by a dedicated LLM agent which is carefully prompted to extract key information and summarize anomalous behaviors. For each device, BiAn generates 11 highly condensed summaries and forwards them for subsequent analysis. This approach allows us to easily extend BiAn for other monitoring tools by adding new LLM agents. The prompt template consists of five parts: role definition, input field description, summary guidelines, alert example, and expected summary, and response format. Additional details, including the prompt templates and an input example of monitor alert summary agents, are provided in Appendices A and B.

Single-device anomaly analysis. Given the summarized alerts, we should answer an important question: *what anomaly can we derive from the data?*

> **Key Experience.** Our operators have defined 7 scenarios of anomalies (Figure 5 right), based on their 10-year operational experience. These scenarios can cover all anomalies that an error device may present. Each anomaly analysis relies on a specific combination of alert types (left).

More importantly, operators' experience also shows that such a small number of scenarios can cover 100% of incidents observable through alerts, which significantly simplifies our design. These scenarios represent low-level symptoms commonly found on devices in large-scale networks and are not tied to specific architectures or

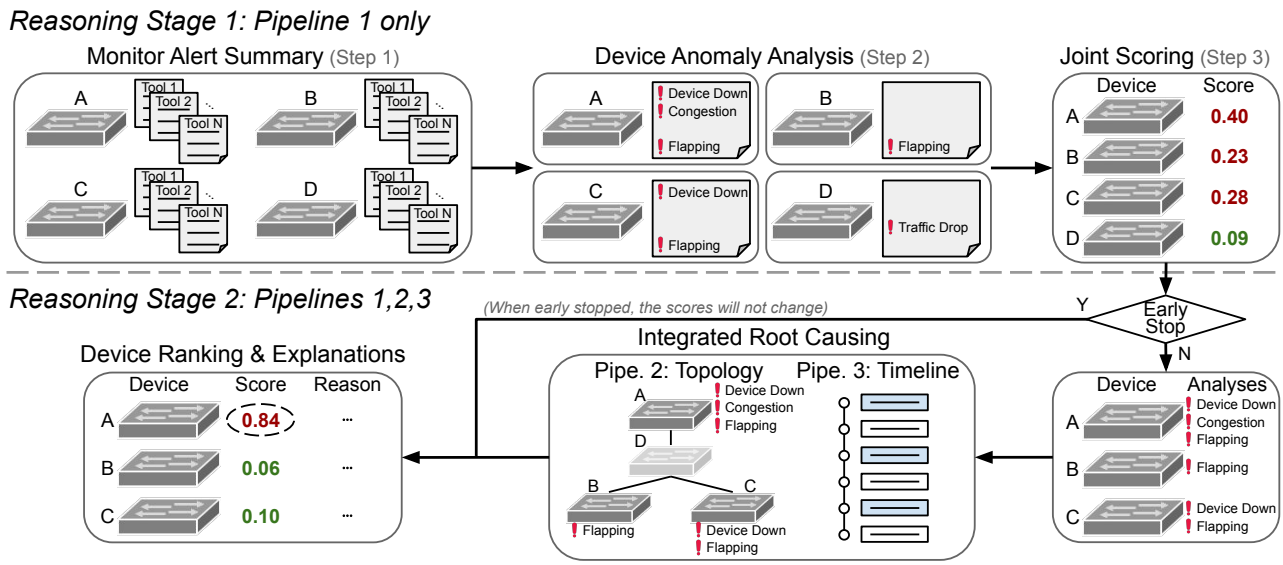


Figure 4: System architecture of BiAn: The two-stage reasoning process first assigns initial scores to all candidates based on anomaly analysis. In Stage 2, which incorporates more comprehensive data for a narrowed list of devices, the error device stands out with more pronounced score differences.

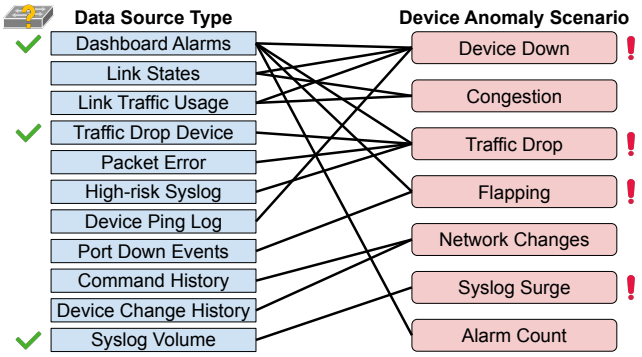


Figure 5: Mapping of data source types and device anomaly scenarios. An example of analysis is provided.

device vendors. Some issues such as silent packet drops may not generate alerts and are thus out of scope.

Driven by the above insights, we employ 7 LLM agents (see prompt template in Appendix A), each responsible for analyzing single devices for one of the anomaly scenarios according to the mapping in Figure 5. For example, the Flapping analysis uses "Dashboard Alarms" and "Port Down Events" summaries to determine whether a device has a flapping anomaly. An anomaly is not the root cause of the incident, and one device can exhibit multiple anomalous behaviors.

Joint scoring. In the last step, BiAn consolidates the anomaly analysis reports from all suspect devices to reason about the root cause of the incident and locate the error device. For the joint reasoning, operators set severity levels for different anomaly scenarios. An LLM agent is tasked to evaluate the compiled reports and generate failure scores for all suspect devices, reflecting the likelihood that each device is the error device. The scores add up to 1. The highest-scoring device is marked as the error device. This step is based on another key insight: The error device typically exhibits more severe and more numerous behaviors compared to others, which may also have anomalies found, but only less severe or partial.

4.2 Three-pipeline Integration

Given the complex spatial and temporal relationships between devices and anomaly events (§2.2), sometimes the system cannot confidently identify the error device using the above SOP-based pipeline (**Pipeline 1**), which mainly focuses on single-device anomalies and their severity levels. Besides, the process demands accurate results. To address such challenges, BiAn incorporates additional information through two more pipelines, *network topology* and *event timeline*, to create a two-stage reasoning framework.

Pipeline 2: Network topology. As network devices are interconnected, anomalies can propagate: anomalies on one device may cause anomalous behaviors on its neighbors. For example, if Device *A* is a parent node of Devices *B* and *C*, and anomalies are detected on all three devices, *A* is likely the source of the incident. To account for this, BiAn extracts topology-related information from logs. It identifies a smaller sub-topology that includes all suspect devices. This approach allows BiAn to focus on the most relevant topological information during analysis. We create an agent that processes JSON-formatted topology data using the following algorithm:

> ❶ Find the shortest paths between every two devices.
> ❷ For any pair of suspect devices, for instance, A and B, remove the shortest paths between A and B if a suspect device C exists on their shortest path. The A-B relationship can be represented via A-C and B-C instead.
> ❸ Construct a topology using the shortest paths left.
> ❹ Simplify the topology by aggregating non-suspect network devices in the same device group into a single node.

Pipeline 3: Event timeline. Different anomalies on different devices occur at different times. The underlying insight is that devices with anomalies reported earlier have higher suspicion. BiAn takes into account the timeline data when performing the final reasoning.

Reasoning Stage 2 - Integrated Root Causing

You are an experienced network operations engineer specializing in investigating complex incidents in large-scale networks. Your task is to analyze reports of suspect devices, identify the most possible error device, and provide explanations.

Anomaly Analysis Reports:
<Device A>: {Analysis 1}: {Analysis content}, {Analysis 2}: {Analysis content}, …
…

Network Topology:
[<Device A>, <Device B>], [<Device A>, <Device D>, <Device C>], …
…

Global Timeline:
{Timestamp 1}: {Event 1}, {Timestamp 2}: {Event 2}, {Timestamp 3}: {Event 3}, …
…

Principles for Investigation (SOPs):
1. Anomaly analysis
2. Differences between devices
3. Alarm count comparison
4. Timeline and Topology

Figure 6: Prompt template for integrated root causing.

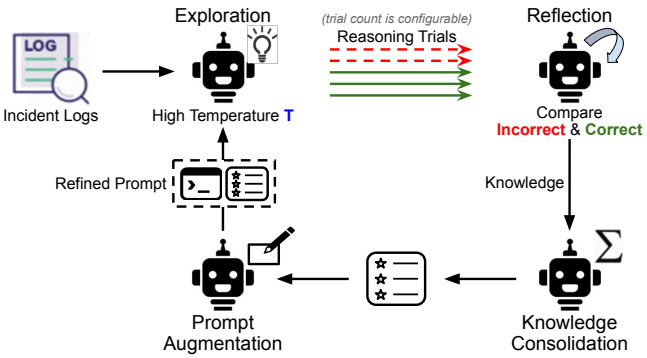


Figure 7: Continuous prompt updating mechanism.

Specifically, the timeline is a list of events from all devices in the order of start times.

Integrated root causing. BiAn does not execute all three pipelines at the beginning. After running Pipeline 1 on all devices and obtaining initial scores, BiAn filters out devices with low suspicion and starts the second reasoning stage on the remaining devices. Specifically, borrowing from LLMs, we define a *Top-p* concept to retain the devices whose cumulative scores, after applying softmax and ranking in descending order, reach a threshold (p). This ensures the remaining device data is kept relevant and focuses on the most suspicious devices. Then, with an enhanced final reasoning LLM agent, BiAn performs integrated root causing and generates failure scores with data from all three pipelines: device anomaly reports, topological relationships, and temporal event data. The prompt for this agent, as presented in Figure 6, specifies the task objective, combines information from three pipelines, and lists principles for root cause analysis. By incorporating data from multiple aspects, this approach addresses the limitations of reasoning on a single pipeline, improving the overall localization performance. Also, when Pipelines 2 and 3 provide stronger evidence for alternate causes, BiAn can adjust its predictions from Pipeline 1 accordingly.

Furthermore, to tackle the randomness of LLMs (for creativity by design), we consider averaging results from repeated executions. Specifically, we introduce *Rank of Ranks*: (1) We run the integrated reasoning step N times; (2) For each run, we rank the devices by failure scores; (3) We calculate the average rank of each device across N trials. Then, the highest-ranked device by average ranks is considered the error device. This approach smooths out the random results output by the LLM while mitigating inaccuracy in absolute scoring. For example, in three trials, the failure scores for two devices are {0.6, 0.3}, {0.5, 0.4}, and {0.2, 0.7}, respectively. The outlier in the third output may result from LLM's randomness. We can easily tell that the first device is more suspicious. However, if we simply average their failure scores, the worse-performing one will be the second device, while the results of *Rank of Ranks* will stay with our intuition. N is configured empirically, and we set it to 3. We show in §6.4 that increasing N beyond 3 yields diminishing performance improvements. Unless the model fundamentally misinterprets a case, repeated reasoning converges eventually.

4.3 Continuous Prompt Updating

As our networks and investigation procedures evolve rapidly, BiAn is designed to adapt. For alert summary and anomaly analysis, our modular design allows for flexible addition of new monitors or anomaly scenarios (or removal of stale ones). For the scoring step (Step 3), a more sophisticated mechanism is needed to enable LLM agents to update continuously. Manually reviewing and revising task prompts is straightforward but labor-intensive, which calls for automated training. However, challenges arise in constructing a training dataset. Data augmentation cannot be used here as we aim for explainable outputs; while operators can label error devices in post-incident reviews, scoring and explanations are subjective and require detailed reasoning. Thus, fine-tuning or retrieval augmented generation (RAG) [39] with historical data is impractical.

To address this, we propose to train LLM's prompts instead of its parameters. This involves two phases: (1) extracting knowledge from past incidents using LLM's generation, reflection, and summarization capabilities, and (2) integrating the knowledge into prompts to enhance reasoning. The algorithm in Figure 7 runs as an iterative loop with four parts: exploration, reflection, knowledge consolidation, and prompt augmentation. This requires designing and building additional LLM agents.

Exploration. Initially, the scoring agent performs 5 reasoning attempts for each incident with the current task prompt. We evaluate the accuracy and filter out cases with perfect accuracy (*i.e.,* 100%). For the remaining cases, reasoning is repeated with a higher temperature T to encourage diversity, thus increasing the likelihood of obtaining both correct and incorrect results. We then prompt the LLM using zero-shot chain-of-thought (CoT) to generate intermediate reasoning steps along with the final scores.

Reflection. We build an LLM agent (Agent R) to analyze those diverse reasoning trials and extract useful knowledge for root causing. By contrasting correct and incorrect trials, R identifies key factors influencing the results and proposes actionable knowledge that can effectively triggers (or avoid) such factors leading to correct (or incorrect) trials.

Knowledge consolidation. To ensure coherence, the knowledge extracted from all incidents is summarized by a different LLM agent (Agent C). C counts the frequency of similar knowledge, and ranks them accordingly. We retain the six most frequently occurring pieces of knowledge, as these are more likely to have broad applicability across various incidents.

Prompt augmentation. Lastly, we integrate the consolidated knowledge into the task prompt to improve reasoning performance. A buffer array stores knowledge from previous iterations. Newly extracted knowledge is added and merged with prior knowledge, with similar pieces combined and their frequencies updated. Another

LLM uses the old prompt and the knowledge buffer to generate a "refined prompt". This ensures controlled adjustments and produces logically consistent prompts. The new prompt resembles the original prompt structure, appended with newly learned knowledge.

We perform prompt updates every few months, aligned with major hardware and software upgrades, to ensure that the reasoning strategies evolve accordingly.

4.4 System Optimizations

We introduce some optimizations for practical consideration.

Fine-tuning. Typically, smaller, more specialized models run faster. We fine-tune smaller models with domain-specific information for simpler tasks, including monitor alert summary and single-device anomaly analysis in Pipeline 1, rather than relying on large, general-purpose models. For the alert summary step, we randomly generate alerts based on the value (numerical or categorical) distributions of our real-world data.[3] Since manually labeling synthetic data is prohibitively labor-intensive, we use GPT-4 to produce ground truth, as it was the most powerful general-purpose model available at the time. For the anomaly analysis step, we randomly combine synthetic alert summaries as inputs, and again, obtain outputs with GPT-4. We develop rule-based algorithms to verify the correctness of the ground truth. A subset of data is further verified through small-scale fidelity tests with operators. Then, the validated data are used to train smaller models for each reasoning task. We provide more fine-tuning insights in Appendix C.

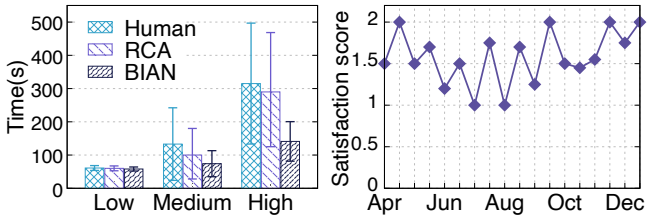
Early stop. There are cases where the error device is relatively obvious, so BiAn only proceeds with further reasoning when it has insufficient confidence. We calculate *entropy* of the failure scores from the first stage to determine the confidence. If the entropy is below a threshold, BiAn stops earlier and directly outputs the device rank. Otherwise, higher entropy indicates greater uncertainty, which triggers the second stage involving more information. This strategy not only saves compute resources for easy cases, but also reduces average response time.

Parallel execution. BiAn executes all agents within the same step concurrently to maximize efficiency, as there are no interdependencies among them. For instance, the alert summary agents and anomaly analysis agents in Pipeline 1 run in parallel. In the second stage, the multiple runs required for calculating *Rank of Ranks* are also performed simultaneously once data from all three pipelines are ready.

5 Real-World Deployment

BiAn has been deployed in Alibaba Cloud's global network infrastructure for ten months. It has successfully helped operators find root causes and corresponding error devices more efficiently. This section evaluates BiAn's effectiveness through A/B testing and explainability analysis (§5.1), presents three real cases to show in detail how BiAn assists our on-call operators in identifying error devices (§5.2), and shares broader operational insights gained from deploying BiAn (§5.3).

Our NOC team consists of tens of engineers with varying levels of expertise. They become on-call operators at a rolling basis and hold regular post-incident review meetings to share experiences and align incident practices. In addition to day-to-day incident response, the team is also actively involved in designing, developing, and continuously improving the incident management workflow.



(a) TTR of solo and assisted incident investigations. **(b) Operators' satisfaction with BiAn's explanations.**

Figure 8: Comparison of time and satisfaction scores.

5.1 Tests during Deployment

We continuously examine the effectiveness of BiAn and collect feedback from our operators during deployment.

A/B tests. To demonstrate the operational value provided by BiAn, we compare incident investigations performed by operators with BiAn versus those done fully manually. We also add a comparison with a variant of RCACopilot [16] which only outputs coarser-grained root cause categories of incidents (see discussion in §2.3). Over the past ten months, while there are on-call operators ready for any escalated incidents, we have specially asked them to set up two shadow operator roles to perform the root causing and failure localization tasks with BiAn and RCACopilot, respectively. Since it is impractical for a single operator to perform the task with and without assistance simultaneously on the same incident, we assigned operators of comparable expertise to each approach at best effort to ensure fairness.[4] We record the *time-to-root-causing* (TTR), defined by our operators as the duration between investigation initiation and the identification of the root cause and error device. TTR complements the renowned metric, time-to-mitigation (TTM), by isolating the investigation phase. Typically, once the root cause is pinpointed, it takes them a relatively fixed duration (around 2 minutes) to proceed to mitigation tools and act. We categorize the incidents into high-, medium-, and low-risk, based on factors such as incident scale (*i.e.*, number of devices affected), recovery difficulty, and impact on SLAs. Figure 8a plots the TTR results across risk levels. BiAn reduces TTR by 20.5% on average, with a notable 55.2% reduction for high-risk incidents since LLM inference time does not inflate with the risk level. While RCACopilot provides root cause categories, the operators still need to revisit monitoring data to validate predictions and pinpoint error devices as the output does not explain the reasoning (*e.g.*, which device is the root cause, how it becomes the root cause). This process becomes increasingly challenging with greater incident complexity, resulting in larger TTR differences at higher risk levels.

Explainability. As an assistant to operators, BiAn must provide clear and useful explanations for its outputs to gain their trust and provide insights into its reasoning. We asked the operators to review the explanations provided for the incidents during post-incident reviews. We set a scoring scale of 0 (not helpful), 1 (somewhat

[3] Unlike §4.3, data augmentation is feasible here because these tasks are simpler, capturing key information for logs and determining the existence of anomalies, rather than providing detailed explanations for device faults.

[4] All operators are qualified for on-call duty and take turns working with both systems. However, we acknowledge that absolute fairness is hard to achieve and individual variability may still introduce minor noise.

helpful), and 2 (very helpful). To minimize bias, we have requested them to provide feedback strictly and independently, and assured that negative ratings were equally valuable. The average scores for high-risk incidents, calculated every 15 days, are shown in Figure 8b. Encouragingly, the feedback predominantly rated the explanations as 1 or 2, with only a few as 0. While it may seem surprising that unhelpful explanations are rare, this reflects the goal of BiAn: to assist human-led investigations. Even in cases where the top-1 device is incorrect, the explanations often highlight useful context or intermediate reasoning, which supports the operator's thinking process. As a result, operators may benefit from BiAn's output more or less and tend not to rate it as completely unhelpful. The fluctuation of scores can be attributed to operators raising their expectations as we iteratively improve BiAn over time.

5.2 Case Study

We discuss three representative incidents to showcase how our approach effectively addresses difficult incidents in practice, particularly how topology and timeline information helps. For confidentiality, we use letters A–C to represent network devices at different levels, with A being the upper stream. Devices $A1$ and $A2$ are on the same level.

Incident 1. We revisit the example introduced in §2.2. It occurred before the deployment of BiAn, so we replay the incident and walk through how BiAn resolves it. Initially, five different alerts were triggered on $B2$ and four on its peer $B3$. Other neighbors (*e.g., Ai* and *Ci*) were affected at varying degrees. Upon feeding the alerts, BiAn outputs $B2$ as the most suspicious device with four anomalies found, and $B3$ as the second, having three anomalies. BiAn successfully detects all anomalies and locates the error device in 28 seconds. In contrast, operators—despite having access to the same data—struggled to pinpoint the exact alerts all at once and locate the root cause immediately. After considerable analysis, they eventually suspected $B2$ or $B3$ might have failed. They confirmed $B2$ as the error device and completed the isolation in 22 minutes. BiAn could have significantly reduced the TTR.

Incident 2. This case aims to highlight how the integration of topology helps. In a sub-network of our WAN, $B1$ is connected to $A1$ and $A2$, with $A2$ also connected to $B2$. After Stage 1, BiAn assigned equal failure scores to $A1$ and $A2$ due to comparable anomalies, both having traffic drop and flapping; other devices have far fewer anomalies. However, in Stage 2, the topology pipeline revealed that $A2$ is directly connected to $B1$ as a parent node. As a result, BiAn flagged $A2$ as the error device with a much higher score. The root cause was a *line card failure* in $A2$. When this incident occurred, on-call operators were notified. With BiAn, they conducted additional measurements for $A2$ and confirmed it was indeed faulty. Thanks to BiAn, the TTR was only 3 minutes.

Incident 3. Accurate localization can sometimes be particularly challenging without timeline data. In an incident, three devices were diagnosed with one or more anomalies: $A1$ experienced disconnection and network changes; $A2$ and $B1$ showed traffic drop. Both disconnection and traffic drop are classified as priority-1 anomalies, so BiAn initially assigned similar scores to all of them, making localization difficult. However, the timeline provides important context: multiple alerts were reported for $A1$ and $A2$, while $A1$'s alerts occurred earlier. This greatly increased $A1$'s suspicion. BiAn thus determined $A1$ as the error device, due to *disconnection*. Upon reviewing BiAn's output and checking monitoring data, the operators ruled out the other devices. The TTR was 1.5 minutes.

5.3 Operational Experience

Selection of candidate devices. In our design (§3), a fixed number of six candidate devices are selected by the upstream monitoring system based on the number of associated alerts, which are then fed into BiAn. We have experimented with different numbers and the choice of "6" strikes an effective balance between processing overhead and coverage. It already makes sure the error device is included over 98% of the time, but not always. We have been actively refining our pre-selection technique: We are developing a dynamic selection mechanism that considers the top suspicious device(s) in various monitoring tools and narrows down the selection to around six devices. Since different individual tools often focus on specific aspects of network behavior, their top outputs—though not always directly relevant—can offer valuable complementary perspectives. We have since seen measurable improvements in coverage (to 99%!).

Iterative design process. BiAn did not take its current form overnight. It has gone through many rounds of iteration and optimization, driven by continuous testing and feedback from operators, before and during deployment. This process, however, comes with challenges. For example, our company imposes retention periods for operational data unless explicitly extracted. The three-pipeline design (§4.2) was not featured in early versions of BiAn. When we came up with the integration of topology and timeline information, we realized that it was not retained in earlier logs, forcing us to delay the improvement until sufficient new data had accumulated. Besides, some data required extra preprocessing to meet the design requirements. In general, we find that adapting datasets to support new features is often a bottleneck in evolving the design, as it incurs non-trivial overhead due to format mismatches or missing context. As a result, the full design and development cycle of BiAn extended well beyond the 10-month deployment duration that we claimed.

Onboarding operators to BiAn. Although we collaborate closely with operators in building BiAn, it is a different yet important job to teach them to use it efficiently. To facilitate a seamless transition, we integrated BiAn into the NOC's on-call platform. The user interface and output format (ranked devices and explanations) align with the system that they are familiar with. Newly onboarded operators have reported a comfortable experience with BiAn, with most requiring minimal training to get started. We share more details on our interaction with operators in Appendix D.

6 Performance Evaluation

We conduct a series of experiments to understand the performance of BiAn, corresponding to the technical challenges mentioned in §2.4. We first demonstrate that BiAn can accurately locate error devices in various scenarios (§6.1). Next, we measure system latency to show that BiAn enables fast investigation (§6.2). We conduct ablation studies to evaluate how each component contributes to the overall performance (§6.3) and show the benefit of the *Rank of Ranks* in reducing output randomness (§6.4). We run microbenchmarks to examine the impact of fine-tuning on monitor alert summary and single-device anomaly analysis in Pipeline 1, as well as the effectiveness of continuous prompt updating (§6.5). Then, we discuss the affordability of deploying BiAn in terms of training and inference

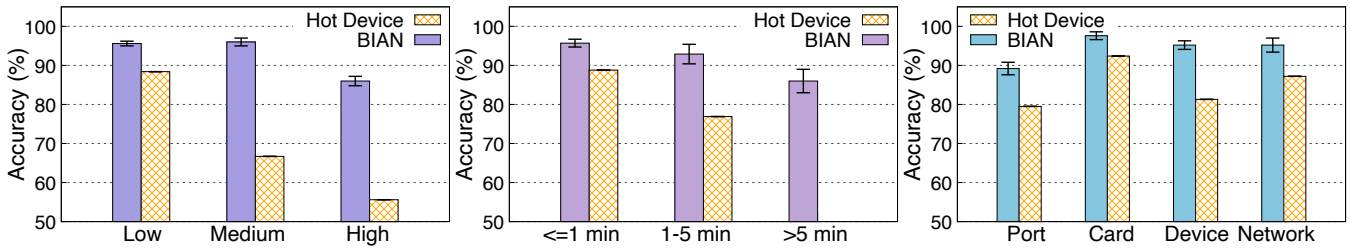


Figure 9: Localization accuracy under different incident categorizations (risk level, difficulty, incident type).

costs (§6.6). Finally, we replace the default model, Qwen [10], with other top-performing LLMs and evaluate (§6.7).

Experiment setup. We install BiAn on a server equipped with an Intel Xeon Platinum 8269CY CPU clocked at 2.50 GHz, 32 GB RAM, and 1 TB SSD for both online deployment and performance evaluation purposes. We select 357 non-trivial cases from real incidents that occurred in our network infrastructure over the past 17 months[5] for evaluation.

Models. For performance evaluation, we use Qwen2.5 in BiAn, with the fine-tuned `Qwen2.5-7B-Instruct` handling the first two steps of Pipeline 1 and `Qwen2.5-72B-Instruct` for all other tasks. For real-world deployment (§5), BiAn always leverages the latest Qwen models available at the time.

6.1 Accuracy

Accuracy is defined as whether the system picks the actual error device as top-1 (see more details in Appendix E). We compare BiAn with our baseline, *Hot Device*, one of our in-use automated failure localization tools. A hot device is the device having the most associated alerts. Similar to 007 [7] which focuses on pinpointing problems in TCP flows, this is a "democratic" approach that assumes the error device, being the source of the incident, exhibits various anomalies while its neighboring devices only show partial or less severe symptoms. Over 357 cases, BiAn achieves 95.5% accuracy averaged over 10 runs, outperforming the baseline's 86.3%. When *Hot Device* cannot tell the differences between several top devices and produces tied top-1s, its accuracy drops to 70.5%, whereas BiAn still maintains high accuracy at 97.1%, highlighting BiAn's advantage of comprehensively analyzing diverse alerts.

We also consider top-2 and top-3 accuracies, where the error device falls into the top 2 or 3 most suspicious devices. The results are 98.6% and 99.3%, respectively (*Hot Device* yields 88.9% and 93.0%). These results are encouraging and already satisfy our operators (§5.1), as (1) explanations from BiAn can significantly accelerate the investigation even if the top-1 device is incorrect, and (2) operators often still review several top-ranked devices for validation before making final decisions.

We further group the incidents in different ways and present the results of BiAn and *Hot Device* in Figure 9. BiAn always wins. Firstly, on the left, we categorize the incidents into three risk levels (see §5.1). We can find that the accuracy remains similar between the first two groups, indicating that BiAn is relatively equally helpful to incidents at low and medium risk levels. For high-risk incidents, accuracy drops below 90%. This decline is due to the high number of concurrent incidents, which increases noise and complicates the extraction of key information. The accuracy of *Hot Device* drops more sharply. In the middle, the incidents are categorized by resolution time (*i.e.,* time-to-root-causing) calculated during post-incident reviews. Shorter times indicate that the cases are easier to address. As shown, easy cases ($t \leq 1$ *min*) have the highest accuracy of 95.7% and medium cases ($1 < t \leq 5$ *min*) approach 92.9%. For harder cases ($t > 5$ *min*), accuracy goes down, likely due to their complexity, which also makes them considerably time-consuming for human operators. Despite this, BiAn's fast resolution capability (discussed next in §6.2) ensures that even if top-1 is not the actual error device, the trial-and-error phase (note, in most cases, mitigation solutions applied to wrong targets are still tolerable) is still shorter compared to manually inspecting extensive logs and aimlessly looking for the error device. *Hot Device* fails to report the actual error devices in all hard cases. On the right, we categorize failures based on their physical root cause types: port, line card, device, and network disconnection. BiAn performs best in resolving line card-related failures. This is because such failures typically affect only the hosting device, which means its neighboring devices are less affected and thus may have fewer anomalies. Other error types receive comparable results.

While we have compared BiAn with *Hot Device* which is adapted from 007 [7], we emphasize that BiAn serves a role different from that of existing failure localization tools in the IM workflow (§2.1). It is not intended to replace them, but rather to complement by assisting operators when self-healing fails.

6.2 Latency

BiAn delivers real-time assistance in root causing and failure localization, which can greatly reduce the total time required for investigation and mitigation. We break down the latency of each system component in Figure 10, together with the end-to-end latency. Overall, the entire reasoning process completes within 30 seconds. When an incident occurs, the automated system notifies on-call operators to start investigation and loads relevant information, which typically takes a minute. This means that BiAn's output is ready by the time operators get online. Looking at the breakdown, the reasoning task of Stage 2 takes relatively longer than other components because it needs to incorporate the analysis results from all devices for generating the final output. Ideally, the end-to-end latency matches the sum of individual component latencies as the workflow is designed for concurrent execution of agents within the same step. However, factors such as compute resources for LLM inference can affect the latency. With our evaluation setup, the end-to-end latency is only 15% greater than the sum, thanks to our high degree of parallelism.

Figure 11 shows the latency reduction for alert summary and anomaly analysis achieved by applying fine-tuned smaller models.

[5]We have made every effort to gather more incidents while ensuring data quality. Each of the selected cases has a ground truth error device, which has been discussed and confirmed by operators during post-incident reviews.

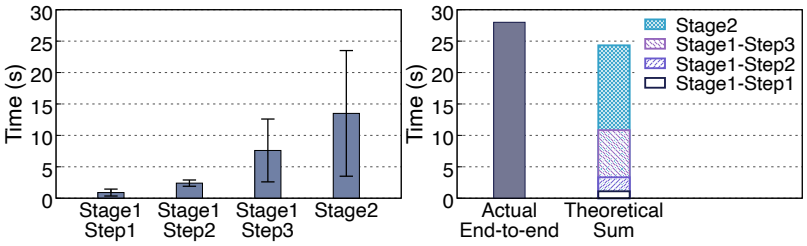


Figure 10: End-to-end latency breakdown.

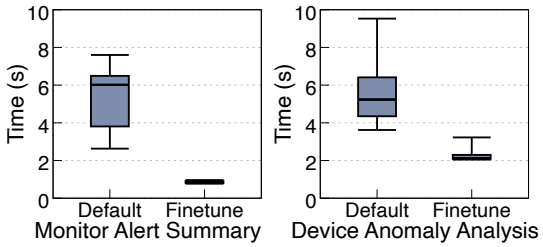


Figure 11: Latency for fine-tuned components.

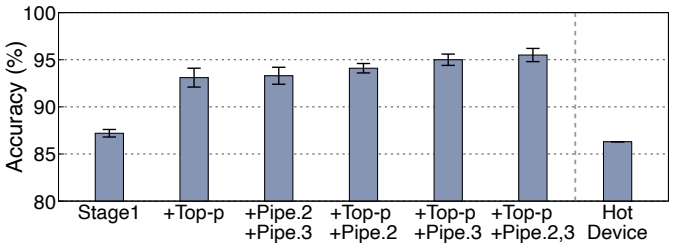


Figure 12: Accuracy at different design phases.

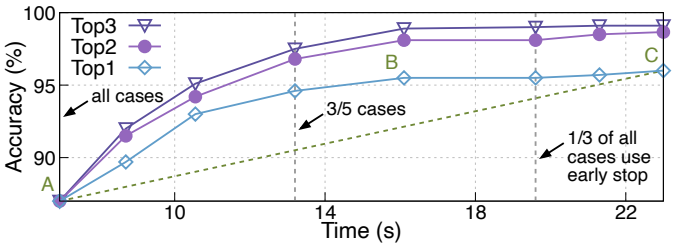


Figure 13: Performance trade-off in the early stop mechanism.

We can observe significant savings. Together with the results in §6.5, we aim to demonstrate that fine-tuning smaller models on less complex tasks with domain-specific knowledge can reduce resource usage without compromising performance—in fact, improving accuracy in our case.

6.3 Ablation Study

We next evaluate the contributions of different components.

Progressive design. We progressively add various reasoning steps to the original SOP-based pipeline (Pipeline 1). Figure 12 presents the results for different design phases. Running only Pipeline 1 achieves an accuracy of 87.2%, which already surpasses the performance of *Hot Device*. With the *Top-p* filter, a second run of Pipeline 1 on the top candidates increases accuracy by 5.9%. Further incorporating the topology or the timeline pipelines brings the accuracy to higher (4th and 5th bars). Finally, the accuracy reaches 95.5% with all components enabled (also see §6.1). Note, running the three pipelines without the *Top-p* filter (3rd bar) results in lower accuracy compared to the final version, because including data of all devices for all pipelines significantly inflates the input token count, which dilute the model's attention.

Early stop. After the first reasoning stage, BiAn uses the entropy of failure scores to determine whether to proceed to Stage 2 (§4.4). The entropy threshold affects the accuracy-latency trade-off. As depicted in Figure 13, when the threshold is higher, more cases stop early, leading to lower latency but less satisfactory accuracy. At zero threshold, BiAn consider all cases uncertain and send them to Stage 2, achieving the highest accuracy at the cost of increased latency. Notably, these curves illustrate a convex relationship: early adjustments yield substantial accuracy gains with minimal time investment (dashed line *AC* represents the accuracy gain per unit time). With an *appropriate* threshold, we can intercept simpler cases at Stage 1 to save time, while reserving Stage 2 for more challenging cases to achieve accurate localization. We consider point *B* as an optimal trade-off and use this threshold (0.75) in deployment. This reduces the overall processing time to 70.0% with only a 0.5% accuracy loss compared to always running Stage 2.

6.4 Result Stability

We introduced *Rank of Ranks* (§4.2) to mitigate randomness in the LLM's output. We compare results from single-run final scoring (Stage 2) versus running it three and five times, then calculating the average ranks to determine the error device. As shown in Figure 14, while average accuracy remains comparable across all settings, the *Rank of Ranks* approach noticeably reduces the variance in accuracy. Increasing the number of rounds from 3 to 5 does not bring significant difference. Note that this design was inspired by the self-consistency technique [67] which proves to be simple yet effective in deductive reasoning where conclusions can be deterministically derived from observations. In our inductive failure localization task, however, results depend on analyzing noisy data, making accuracy improvements less pronounced.

6.5 Microbenchmarks

We conduct additional experiments on specific components.

Monitor alert summary. An alert summary, if not empty, consists of key elements such as type, start/end timestamps, alert digest, and type-specific details. We manually check if the summaries capture critical information in the digest. Results aggregated from four representative alert types are plotted in Figure 15. Across all summaries regardless of whether the device is the error device, fine-tuned models achieve an average accuracy of 98.7%, significantly outperforming the default model. Such tasks, though simpler, may need domain expertise for effective and accurate summarization.

Single-device anomaly analysis. The next step involves determining if a device exhibits a specific anomaly. Figure 15 (right) shows that our fine-tuned models achieve 98.6% accuracy, beating the default model by over 6%.

Prompt updating. We evaluate the prompt updating algorithm using five-fold cross-validation, splitting our incident dataset into 80% training and 20% testing. We conduct multiple iterations of the algorithm and measure accuracy after each iteration, averaging the results over five runs. After three iterations, training accuracy increases from 88.7% to 90.0%, and test accuracy rises from 81.7% to 85.9%. The marginal gain is largely due to the high baseline

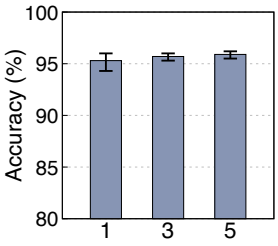


Figure 14: Impact of *Rank of Ranks* rounds.

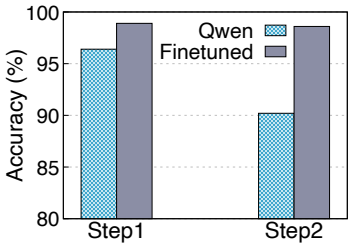


Figure 15: Benefits of fine-tuning on Steps 1 and 2.

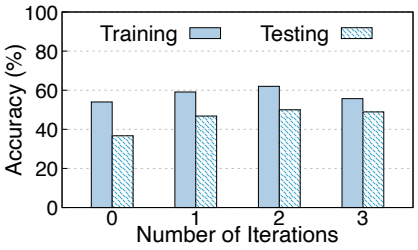


Figure 16: Accuracy with number of iterations in prompt updating.

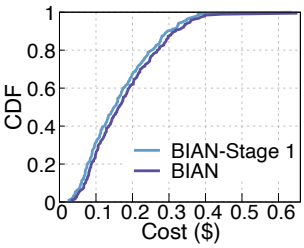


Figure 17: Training and inference costs.

Table 1: Performance of different models.

Model Name	Size	Top 1	Top 2	Top 3
Qwen2.5	72B	95.5	98.6	99.3
Llama-3.1	405B	95.7	98.7	99.3
GPT-4o	-	95.2	98.1	99.3
Claude-3.5-Sonnet	-	93.9	97.4	98.5
Gemini-1.5-Pro	-	93.2	97.9	98.7

performance, as over 90% of incidents already achieve 100% accuracy. We exclude them and repeat the experiment whose results are presented in Figure 16. After three iterations, accuracy improves from 54.0% to 62.0% for the training set, and from 36.7% to 50.0% for testing. Beyond that, excessive prompt length hinders LLM's comprehension, causing performance degradation as the LLM begins to ignore most of the augmented instructions.

6.6 Training and Inference Costs

The primary costs of deploying BiAn in our network infrastructure come from LLM usage: (1) fine-tuning specialized LLMs for alert summary and device anomaly analysis, and (2) online inference for all LLM agents in BiAn during live investigations. For fine-tuning, we record the actual costs incurred by our training jobs, which is $121.63 on average. For inference, we calculate the cost for each incident based on Qwen's input/output token pricing. Input tokens include both prompts and upstream data from the monitors or the previous step in BiAn. Figure 17 shows the CDF of per-incident costs for two settings: running only the SOP-based Pipeline 1 and running the full BiAn. The average cost is only $0.17 and $0.18, respectively, demonstrating BiAn's cost efficiency at scale. We note that, Pipeline 1 alone already beats *Hot Device* in accuracy, and an additional $0.01 per incident brings further gains (see §6.1). Such improvements are meaningful in large-scale cloud environments, where better incident management translates into significant savings in terms of service availability and quality.

6.7 Running with Other LLMs

Since BiAn was originally designed for internal use, we initially deployed it with Qwen [10], our open-source language model series. To validate the generalizability of our approach, we extend our experiments to include several state-of-the-art LLMs. For fairness, we did not specially optimize prompts for different models. As summarized in Table 1, BiAn consistently achieves high top-1 to top-3 accuracies on all models. The results suggest that our design is not only effective with Qwen but also works well across other leading models. This cross-model evaluation highlights the robustness of our proposed framework.

7 Lessons and Discussion

We share our lessons learned in designing and deploying BiAn, and discuss some open questions.

API latency fluctuation. BiAn relies on Qwen's online services for inference. It issues an API call for each reasoning task. As incidents can happen at any time, BiAn is triggered at different times. We have noticed variations in end-to-end latency. We run the inference tasks at different times over 24 hours (details in Appendix F) and find that, average latency remains stable throughout the day (around 5 seconds) but individual calls can experience significant fluctuations (up to 16 seconds). This can impact the latency-sensitive tasks. Similar issues are also observed in other LLM services.

Explainability and trust. BiAn provides detailed explanations along with failure scores, for two main reasons: (1) they build operators' trust, especially during early deployment; and (2) they facilitate feedback that helps refine prompts, and even framework design. Experiments show that accuracy is not affected by whether we ask the LLM agent to provide explanations, but including explanations reduces score entropy (*e.g.,* {0.5, 0.3, 0.2}→{0.7, 0.2, 0.1}), indicating greater confidence. Moreover, it is hard to know in advance whether correct rankings come with wrong explanations, or whether useful explanations are hidden by incorrect rankings. Future work can explore how to validate the consistency between scores and explanations.

Prompt engineering. We have spent considerable effort on improving the LLM prompts in BiAn and gained some insights. First, managing prompt length is crucial. We have observed a decline in performance as prompt length increases, calling for precise instructions to minimize distractions from less important information. Similar challenges arise in prompt updating (§4.3), where prompt length increases with iterations. The performance decline due to growing prompts will eventually outweigh the benefits brought by augmented knowledge. This indicates a limitation of "training" LLM prompts versus LLM parameters. Second, prompt obedience can be inconsistent. LLM agents sometimes fail to follow our instructions, such as skipping analysis steps or producing overly verbose content. We need to carefully tune the prompts to clarify roles and align agent behavior with the intended tasks. On the other hand, we want agents to explore different reasoning paths on complex cases (*e.g.,* devices with similar symptoms, multiple root causes). It has been non-trivial to make the trade-off between control and flexibility. We have applied various techniques, including few-shot prompting (in-context learning) [12, 71], chain-of-thought prompting [70], and self-consistency [67].

Multi-agent systems. Multi-agent orchestration [15, 29, 40, 73] is another direction that deserves further study. In the early design stage, we explored enabling LLM agents to freely plan analysis tasks, including alert processing and device scoring. However, this setup quickly ran into practical challenges: without clear task boundaries and guidelines, the agents often entered loops of repeated actions, failed to terminate properly, or accumulated excessive context, leading to performance degradation. Later, we transitioned to a structured, hierarchical framework with predefined roles, informed by the thinking and handling processes of human operators. The roles are scoped and incorporate data from different dimensions (*e.g.,* alerts, topologies, timelines). Despite the improvements in controllability, this static workflow has limited flexibility—except that we can define more roles (*e.g.,* for new alert types) under the framework. In the future, to handle increasingly complex incidents as network infrastructure evolves, the agents need to have more adaptive behaviors, such as retrying failed analysis, skipping irrelevant steps, or dynamically invoking other agents. Designing multi-agent systems [13] raises new challenges in prompt engineering, agent coordination, and model tuning.

Limitations. While BiAn has greatly improved incident investigation, certain limitations remain: (1) BiAn falls short in multi-device failures. It assumes a single error device per incident. Although rare, there are cases where multiple devices contribute equally. For instance, network updates involving configuration changes on two devices may result in errors on both. Nevertheless, the accompanying scores and explanations often reveal that the top two are closely ranked, enabling operators to identify both as faulty, quickly. (2) BiAn relies on the effectiveness of upstream monitors. If monitors generate invalid or ambiguous data that can hardly differentiate responsibilities among devices, it may struggle to provide accurate results. Notably, this also challenges human operators, who often report uncertainty or need more time. BiAn shines at efficiently understanding and reasoning through large amounts of data. (3) BiAn does not yet support link-based incidents. It currently focuses on device-based incidents which accounts for the majority of cases. Link-based incidents follow different investigation processes and SOPs. We are extending BiAn to handle both device- and link-based incidents effectively.

8 Related Work

Network telemetry and measurements. Operators rely on measurements to assess network health. Numerous studies aim to indicate fault-related information through flow-level or packet-level data [24, 25, 35, 36, 51, 58, 59, 79, 82, 88, 90]. Some utilize such measurements for preliminary identification and routing of faults [8, 18, 19, 44, 45, 49]. While helpful in narrowing down a range of devices, they cannot pinpoint error devices. Other works optimize telemetry systems themselves in terms of overhead, reliability, and availability [4, 20, 21, 23, 30–32, 41, 46, 47, 54, 61, 77, 80, 87, 89], without directly addressing failure localization. NetPilot [74], CorrOpt [91], and SWARM [52] optimize incident mitigation strategies after failures are located.

Network failure localization. Various works target diagnosis and failure localization. Some use statistical techniques to locate link failures [28, 37], while others analyze packet drops [7, 53, 62]. Several studies propose rule-based troubleshooting systems for IPTV/IP networks used by ISPs [6, 48, 76]. NetSonar [83] builds a gray box tester for network tomography. NetPoirot [8] tells if bottlenecks reside in the network. NetNORAD [38] and Pingmesh [22] locate failures at a network level (*e.g.,* spine, ToR switch). They can serve as monitoring data sources for operators and BiAn, for example, by generating alerts or recommending orders of fixes. Some works target data center network failures and need hardware support [26, 42, 66]. INT [11] and provenance-based solutions [14, 75] are resource-intensive. Packet marking is intrusive and fields may not be available [55]. These are hard to be deployed in heterogeneous infrastructures like ours.

Troubleshooting at upper layers. Efforts are also made in detecting software and service issues in cloud environments. Some approaches identify root causes leveraging dependencies between entities [9, 17, 27, 34]. For example, Murphy [27] uses a Markov Random Field to model dependencies for performance diagnosis. These works primarily conduct analysis on performance metrics, whereas BiAn operates on textual alerts generated by monitoring and diagnosis systems. Others address purely application-layer bugs unrelated to physical devices [43, 50, 57, 60, 68, 85], and thus cannot assist physical network teams in locating device faults.

Utilizing LLMs in the network domain. Besides the works mentioned earlier in §2.3, there are other attempts that apply LLMs in systems and networking. In cloud software incident management, Roy *et al.* [56] evaluate the ReAct framework [78] in incident analysis. Zhang *et al.* [86] use in-context learning to address challenges in fine-tuning general-purpose LLMs. RCAgent [69] is an agent capable of tool use for analyzing cloud job anomalies. LM-PACE [84] estimates output confidence for black-box root causing from incident summaries. NetLLM [72] modifies LLM architectures for network-specific applications. They have provided valuable insights for us in building BiAn.

9 Conclusion

We present BiAn, a practical framework that leverages LLMs to improve root cause analysis and failure localization in network incident management. By decoupling the complex reasoning process across vast volumes of monitoring data, BiAn—through its hierarchical, multi-pipeline design and other system enhancements—significantly boosts the efficiency of localization while maintaining explainability. Its successful deployment in Alibaba Cloud's production network demonstrates both the feasibility and impact of applying LLM-based approaches in network operations at scale. We are actively working to further advance BiAn, and we hope our experience can spur more research in AI for networking.

Acknowledgments

We acknowledge all teams within Alibaba Cloud that contributed to the success of BiAn, including the Network Automation, Network Operations, and Network Systems, to name a few. We thank our shepherd, Brighten Godfrey, and the anonymous reviewers for their insightful comments. This work was supported by the National Natural Science Foundation of China under Grant No. 62325205, the Beijing Municipal Science and Technology Project No. Z241100004224023, the National Natural Science Fund for the Excellent Young Scientists Fund Program (Overseas), and Alibaba Cloud through the Alibaba Research Intern Program. Ennan Zhai, Chen Tian, and Guyue Liu are the corresponding authors.

References

[1] 2023. IT outages cost companies up to $1M per hour: report. https://www.ciodive.com/news/IT-outage-cost-report-new-relic/696359/.

[2] 2024. Anthropic - Introducing Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet.

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[4] Anup Agarwal, Zaoxing Liu, and Srinivasan Seshan. 2022. {HeteroSketch}: Coordinating network-wide monitoring in heterogeneous and dynamic networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 719–741.

[5] 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)*. IEEE, 1737–1749.

[6] MAHIMKAR Ajay. 2008. Troubleshooting Chronic Conditions in Large IP Networks. *ACM CoNEXT, December 2008, Madrid, Spain* (2008).

[7] 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.

[8] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. 2016. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 440–453.

[9] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A Maltz, and Ming Zhang. 2007. Towards highly reliable enterprise network services via inference of multi-level dependencies. *ACM SIGCOMM Computer Communication Review* 37, 4 (2007), 13–24.

[10] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[11] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 662–680.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[13] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657* (2025).

[14] Ang Chen, Yang Wu, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. 2016. The good, the bad, and the differences: Better network diagnostics with differential provenance. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 115–128.

[15] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2023. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848* (2023).

[16] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2024. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 674–688.

[17] 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.

[18] Jiaqi Gao, Nofel Yaseen, Robert MacDavid, Felipe Vieira Frujeri, Vincent Liu, Ricardo Bianchini, Ramaswamy Aditya, Xiaohang Wang, Henry Lee, David Maltz, et al. 2020. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 253–269.

[19] Kaihui Gao, Chen Sun, Shuai Wang, Dan Li, Yu Zhou, Hongqiang Harry Liu, Lingjun Zhu, and Ming Zhang. 2022. Buffer-based end-to-end request event monitoring in the cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 829–843.

[20] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. {SIMON}: A simple and scalable method for sensing, inference and measurement in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 549–564.

[21] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. 2017. Dapper: Data plane performance diagnosis of tcp. In *Proceedings of the Symposium on SDN Research*. 61–74.

[22] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 139–152.

[23] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 357–371.

[24] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 71–85.

[25] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. 2018. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research*. 1–7.

[26] Vipul Harsh, Tong Meng, Kapil Agrawal, and Philip Brighten Godfrey. 2023. Flock: Accurate network fault localization at scale. *Proceedings of the ACM on Networking* 1, CoNEXT1 (2023), 1–22.

[27] 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.

[28] Herodotos Herodotou, Bolin Ding, Shobana Balakrishnan, Geoff Outhred, and Percy Fitter. 2014. Scalable near real-time failure localization of data center networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1689–1698.

[29] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations*.

[30] Qun Huang, Patrick PC Lee, and Yungang Bao. 2018. Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 576–590.

[31] Qun Huang, Siyuan Sheng, Xiang Chen, Yungang Bao, Rui Zhang, Yanwei Xu, and Gong Zhang. 2021. Toward {Nearly-Zero-Error} sketching via compressive sensing. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 1027–1044.

[32] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 404–421.

[33] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, et al. 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.

[34] Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. 2009. Detailed diagnosis in enterprise networks. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 243–254.

[35] Pravein Govindan Kannan, Nishant Budhdev, Raj Joshi, and Mun Choon Chan. 2021. Debugging transient faults in data centers using synchronized network-wide packet histories. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 253–268.

[36] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. 2007. Detection and localization of network black holes. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2180–2188.

[37] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. 2009. Fault localization via risk modeling. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (2009), 396–409.

[38] Petr Lapukhov and Aijay Adams. 2016. NetNORAD: Troubleshooting networks via end-to-end probing. https://engineering.fb.com/2016/02/18/core-infra/netnorad-troubleshooting-networks-via-end-to-end-probing/.

[39] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[40] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems* 36 (2023), 51991–52008.

[41] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. {FlowRadar}: A better {NetFlow} for data centers. In *13th USENIX symposium on networked*

systems design and implementation (NSDI 16). 311–324.

[42] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. LossRadar: Fast detection of lost packets in data center networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. 481–495.

[43] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, et al. 2020. Gandalf: An intelligent,{End-To-End} analytics service for safe deployment in {Large-Scale} cloud infrastructure. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 389–402.

[44] Kefei Liu, Zhuo Jiang, Jiao Zhang, Shixian Guo, Xuan Zhang, Yangyang Bai, Yongbin Dong, Feng Luo, Zhang Zhang, Lei Wang, et al. 2024. R-Pingmesh: A Service-Aware RoCE Network Monitoring and Diagnostic System. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 554–567.

[45] Kefei Liu, Zhuo Jiang, Jiao Zhang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. 2023. Hostping: Diagnosing intra-host network bottlenecks in {RDMA} servers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 15–29.

[46] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*. 334–350.

[47] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 101–114.

[48] Ajay Anil Mahimkar, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Qi Zhao. 2009. Towards automated performance diagnosis in a large IPTV network. *ACM SIGCOMM Computer Communication Review* 39, 4 (2009), 231–242.

[49] Ajay Anil Mahimkar, Han Hee Song, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Joanne Emmons. 2010. Detecting the performance impact of upgrades in large operational networks. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 303–314.

[50] Sonu Mehta, Ranjita Bhagwan, Rahul Kumar, Chetan Bansal, Chandra Maddila, Balasubramanyan Ashok, Sumit Asthana, Christian Bird, and Aditya Kumar. 2020. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 435–448.

[51] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 129–143.

[52] 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)*. 335–357.

[53] Yanghua Peng, Ji Yang, Chuan Wu, Chuanxiong Guo, Chengchen Hu, and Zongpeng Li. 2017. {deTector}: a topology-aware monitoring system for data center networks. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 55–68.

[54] Tongqing Qiu, Zihui Ge, Dan Pei, Jia Wang, and Jun Xu. 2010. What happened in my network: mining network events from router syslogs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 472–484.

[55] 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)*. 595–612.

[56] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring llm-based agents for root cause analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 208–219.

[57] Junxian Shen, Han Zhang, Yang Xiang, Xingang Shi, Xinrui Li, Yunxi Shen, Zijian Zhang, Yongxiang Wu, Xia Yin, Jilong Wang, et al. 2023. Network-centric distributed tracing with deepflow: Troubleshooting your microservices in zero code. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 420–437.

[58] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. 2017. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*. 164–176.

[59] Haifeng Sun, Jiaheng Li, Jintao He, Jie Gui, and Qun Huang. 2023. OmniWindow: A general and efficient window mechanism framework for network telemetry. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 867–880.

[60] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2016. Simplifying datacenter network debugging with {PathDump}. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 233–248.

[61] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2018. Distributed network monitoring and debugging with {SwitchPointer}. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 453–456.

[62] 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.

[63] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[64] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[65] 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.

[66] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, Ang Chen, and TS Eugene Ng. 2022. Closed-loop network performance monitoring and diagnosis with {SpiderMon}. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 267–285.

[67] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[68] Zhe Wang, Huanwu Hu, Linghe Kong, Xinlei Kang, Qiao Xiang, Jingxuan Li, Yang Lu, Zhuo Song, Peihao Yang, Jiejian Wu, et al. 2024. Diagnosing Application-network Anomalies for Millions of {IPs} in Production Clouds. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 885–899.

[69] 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.

[70] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[71] Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. 2023. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846* (2023).

[72] 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.

[73] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

[74] 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.

[75] 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.

[76] He Yan, Lee Breslau, Zihui Ge, Dan Massey, Dan Pei, and Jennifer Yates. 2010. G-rca: a generic root cause analysis platform for service quality management in large ip networks. In *Proceedings of the 6th International COnference*. 1–12.

[77] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 561–575.

[78] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[79] Minlan Yu, Albert Greenberg, Dave Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. 2011. Profiling network performance for multi-tier data center applications. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*.

[80] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software {Defined}{Traffic} Measurement with {OpenSketch}. In *10th USENIX symposium on networked systems design and implementation (NSDI 13)*. 29–42.

[81] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, et al. 2024. MonitorAssistant: Simplifying Cloud Service Monitoring via Large Language Models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 38–49.

[82] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Automatic test packet generation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 241–252.

[83] Hongyi Zeng, Ratul Mahajan, Nick McKeown, George Varghese, Lihua Yuan, and Ming Zhang. 2015. *Measuring and Troubleshooting Large Operational Multipath Networks with Gray Box Testing*. Technical Report MSR-TR-2015-55.

[84] 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.

[85] Qiao Zhang, Guo Yu, Chuanxiong Guo, Yingnong Dang, Nick Swanson, Xinsheng Yang, Randolph Yao, Murali Chintalapati, Arvind Krishnamurthy, and Thomas Anderson. 2018. Deepview: Virtual disk failure diagnosis and pattern detection for azure. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 519–532.

[86] 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.

[87] Hao Zheng, Chengyuan Huang, Xiangyu Han, Jiaqi Zheng, Xiaoliang Wang, Chen Tian, Wanchun Dou, and Guihai Chen. 2024. μMon: Empowering Microsecond-level Network Monitoring with Wavelets. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 274–290.

[88] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. 2020. Flow event telemetry on programmable data plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 76–89.

[89] Yang Zhou, Ying Zhang, Minlan Yu, Guangyu Wang, Dexter Cao, Eric Sung, and Starsky Wong. 2022. Evolvable network telemetry at facebook. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 961–975.

[90] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491.

[91] 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. As supplementary materials, we provide additional details, including prompt templates, monitoring data examples, details regarding fine-tuning, our interaction with operators, clarifications on the accuracy metric, explanations on some terms, and an experiment on inference API calls. While the core paper is self-contained, these materials offer more information for interested readers.

A Prompt Templates

Reasoning Stage 1 - **Monitor Alert Summary**

You are an experienced network operations engineer specializing in investigating complex incidents in large-scale networks. Your task is to analyze monitoring alerts, extract key information, and generate a data summary.

Field Description:
Device Name: <Device A>
Event Details: …
…

Summary Guidelines:
1. Ensure the summary is concise and clear.
2. Include key information such as XXX, XXX, …
3. Extract the relevant details without omission.

Example #1:
Input: <Monitoring log>
Summary: …

Response Format:
Device Name: <Device A>
Digest: …

Figure 18: Prompt template for monitor alert summary.

We apply various prompt engineering techniques when designing prompts for the LLM agents in BiAn. Figure 18 shows the prompt template for monitor alert summary, Step 1 in Pipeline 1. The template contains five parts: (1) Role definition: specifying the agent's role; (2) Field description: describing the fields in the log to be analyzed; (3) Summary guidelines: providing guidelines on how to summarize; (4) One-shot example: providing an example of the data along with the expected summary; (5) Response format: highlighting the format of the response (*i.e.,* alert summary). We also share the template for single-device anomaly analysis (Pipeline 1 Step 2) in Figure 19. Similarly, we first define the role of the agent and describe the expected input (*e.g.,* alert summary format). Then, we specify how to perform an analysis. One-shot example helps the agent understand its role with in-context learning. At the end, response format is provided.

Reasoning Stage 1 - **Single-device Anomaly Analysis**

You are an experienced network operations engineer specializing in investigating complex incidents in large-scale networks. Your task is to perform a comprehensive analysis of all relevant monitoring data summaries, and determine whether there is an anomaly of XXX in the current device, step by step.

Data Summaries:
<Summary 1>: {Summary content}, {Summary 2}: {Summary content}, …
…

Analysis SOPs:
Step 1: <Description>
Step 2: <Description>
…

Example #1:
Input: <Data summaries>
Summary: <Step-by-step analysis>

Response Format:
Step 1: <Analysis>, Step 2: <Analysis>, …
Conclusion: …

Figure 19: Prompt template for device anomaly analysis.

B Input Examples

The following example is an alert produced by the monitoring system. As shown, the events have undergone aggregation, resulting in a semi-structured format that contains mostly human-readable textual fields. This format incorporates preliminary classifications and statistics, eliminating the need for operators and BiAn to directly process raw time series data, numerical values, or charts.

```
{
  "brief": "inter-pod circuit|CITY61-R1-VM-##-G1_CITY61-VM-GATE-
       G2_GATE EGRESS|2/32 ANOMALY: BGP-down-Alert",
  "circuitGroupName": "CITY61-R1-VM-##-G1_CITY61-VM-GATE-G2_GATE
       EGRESS",
  "clusterNameA": "CITY61-R1-VM-##-G1",
  "clusterNameB": "CITY61-VM-GATE-G2",
  "deviceNameA": "R1-VM-##-G1-1.CITY61",
  "deviceNameB": "GATE-VM-4.CITY61",
  "eventSource": "SYSLOG EVENT",
  "faultScenario": "LINK FLAPPING",
  "gmtCreate": "2024-09-20 16:03:11",
  "gmtModified": "2024-09-20 16:03:12",
  "level": "2",
  "scanObjs": "R1-VM-##-G1-1.CITY61#100GE1/0/5,R1-VM-##-G1-2.CITY61
       #100GE1/0/5,GATE-VM-4.CITY61#HU1/12",
  "type": "LINK_GROUP",
  "typeName": "[V2|P2][low freq] backbone link flapping"
},
{
...
},
...
```

C Fine-tuning

Training strategy. In practice, we find that combining data generated for multiple different tasks into a single training set does not diminish the accuracy of the fine-tuned model across individual tasks. In fact, this approach often leads to modest improvements in

performance. This finding is significant as it reduces the resource consumption for training and deploying models. In our current deployment, we maintain only one fine-tuned model each for Step 1 and Step 2.

Hyper knobs. We have conducted a thorough exploration of the fine-tuning hyperparameters. We observe that richer training data generally yields better performance of fine-tuned models while larger batch sizes accelerate the training process. For most tasks, a dataset of 2,000 examples per task is sufficient to achieve near-optimal performance, comparable to models trained on 5,000 examples. Interestingly, we also find that reducing the batch size enhances accuracy, with the best results achieved using a batch size of 1.

D Interaction with Operators

Questionnaire. To evaluate operators' satisfaction with BiAn, we collected feedback focused on the clarity and usefulness of the explanations provided by BiAn. For medium- and high-risk incidents, they also assessed if critical information was included. As mentioned in §5.1, we used a 3-point scale: 0 (not helpful), 1 (somewhat helpful), and 2 (very helpful). Experienced operators participated in reviewing the outputs of BiAn. To gain more insights into the system's performance in real-world operations, we also include the following questions in the questionnaire:

❶ Which types of incidents are handled inadequately?
❷ What problems should we prioritize to address next?
❸ Was this case trivial to resolve?
❹ Whether trivial or not, was the explanation helpful?
❺ If BiAn fails in this case, was the explanation for the top-1 (incorrect) device reasonable? Any logical fallacies?

Result display. Operators provide feedback on how to improve output formatting based on their typical reading patterns. Their suggestions include adding bold titles at the top of push notifications, using collapsible views for detailed explanations, and embedding original monitor alerts. We have incorporated these recommendations iteratively into BiAn to enhance result readability. In addition to LLM-based reasoning, BiAn also integrates relevant references to better support operators in the decision-making process, including links to tools for deeper analysis and data visualization (*e.g.,* traffic statistics, packet loss rates). The resources enable operators to make more informed decisions.

Learning curve. A common barrier for our operators to adopting new tools is a steep learning curve or deviation from their existing practices. To minimize this, BiAn's core design aligns with internal operational guidelines, ensuring acceptance by both experienced and newly onboarded operators. As we maintain close collaboration with the NOC team throughput the development process, most operators become proficient with BiAn after just a few real-world cases.

E Accuracy

Table 2: Confusion matrix for error device localizaion.

	Root Cause (Positive)	Non-Root Cause (Negative)
True	Y	$(D-1)Y$
False	$X-Y$	$(D-1)(X-Y)$

In the context of error device localization, we frame the problem as a special case of classification, where each sample (*i.e.,* incident) contains exactly one positive instance, the true error device. We illustrate this with Table 2. Assume we have X incidents and BiAn successfully identifies the error device in Y of them. Without loss of generality, let the initial number of candidate devices be D, determined by upstream monitoring and filtering tools. The accuracy is calculated as:

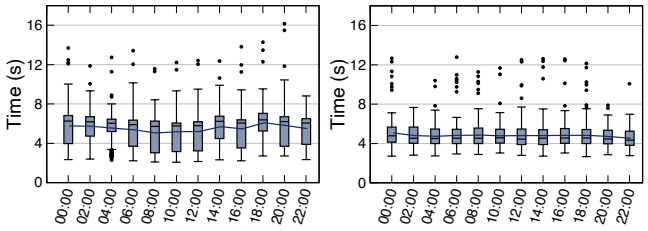
$$\frac{TP+TN}{D\times X}=\frac{Y+(D-1)Y}{DX}=\frac{Y}{X}, \tag{1}$$

which reduces to the fraction of cases in which the system correctly identifies the error device. This also allows us to derive other metrics. For example, the false positive rate is:

$$\frac{FP}{D\times X}=\frac{X-Y}{DX}=\frac{1}{D}(\frac{X-Y}{X})=\frac{1}{D}(1-\frac{Y}{X})=\frac{1}{D}(1-Accuracy). \tag{2}$$

Similarly, we can get the true positive rate, true negative rate, and false negative rate if needed.

F Inference API Calls



(a) Step 1: Alert summary. **(b) Step 2: Anomaly Analysis.**

Figure 20: Inference API call latency.

In §7, we observed fluctuating latency when invoking Qwen's inference APIs. This perceived latency consists of network latency, task scheduling latency, and inference latency. The former two may be related to the current background load on the Qwen platform. Inference latency itself can be influenced by factors such as inference framework, acceleration methods, available resources for inference, batching states of the inference task, hardware stragglers, and dynamic scaling of underlying infrastructure components. To further investigate the fluctuating latency, we run an experiment: we repeatedly run inference tasks for the alert summary and device anomaly analysis steps over a 24-hour period, 100 times per hour. As shown in Figure 20, while the Qwen API can provide a stable overall latency on average, the fluctuation in latency for individual calls remains substantial, potentially impacting latency-sensitive inference tasks. For stable latency, we can consider adopting a locally deployed, dedicated resource approach (see Figure 11 for a comparison between local fine-tuned inference and Qwen API invocation).