Dasheng: LLM-powered Reliability Engineering Assistant for Cloud Overlay Networks

Yining Qi[†,§], Chongrong Fang[‡(✉)], Xiaoqing Sun[§], Tong Wu[†], Jiaxiang Chen[‡], Xiaobo Xue[§], Shize Zhang[§], Xiaonan Shi[§], Yangyang Shi[§], Shuai Yang[§], Xiong Xiao[§], Peng Cheng[†], Jiming Chen[†], and Biao Lyu[§(✉)]

[†]Zhejiang University [‡]Shanghai Jiao Tong University [§]Alibaba Cloud

China

Abstract

Amid mounting incidents of network failures, cloud service providers are placing increasing emphasis on reliability engineering. Leveraging their capabilities in text understanding and reasoning, Large Language Models (LLMs) have been integrated to assist conventional cloud network reliability tools. However, deploying an LLM-powered reliability engineering assistant to cloud overlay networks is significantly challenging due to the scale, complexity, and intricate couplings of cloud overlay network components. In this work, we present Dasheng, an LLM-powered reliability engineering assistant for cloud overlay networks. Dasheng extends the standard assistant paradigm with an iterative, multi-step tool interaction mechanism to fit the characteristics of cloud overlay networks. Dasheng has been deployed on a production cloud for two years. To date, it has been used by 1,450 engineers for approximately 70k invocations. Most Dasheng invocations complete within 3-100 seconds, and reduce case resolution time by up to 97%. In 92% of evaluated cases, Dasheng produces useful outputs that materially assist engineers.

CCS Concepts: • **Networks** → **Cloud computing**; **Network reliability**.

Keywords: Cloud computing, network reliability, LLM

ACM Reference Format:

Yining Qi[†,§], Chongrong Fang[‡(✉)], Xiaoqing Sun[§], Tong Wu[†], Jiaxiang Chen[‡], Xiaobo Xue[§], Shize Zhang[§], Xiaonan Shi[§], Yangyang Shi[§], Shuai Yang[§], Xiong Xiao[§], Peng Cheng[†], Jiming Chen[†], and Biao Lyu[§(✉)]. 2026. Dasheng: LLM-powered Reliability Engineering Assistant for Cloud Overlay Networks. In *Proceedings of The 2026 ACM SIGOPS Annual Technical Conference (Conference ATC '26).* ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Unreliability events [1, 7, 15, 34] in cloud networks emerge frequently and bring huge economic and reputational loss. Hence, maintaining the reliability of cloud overlay networks has always been a top priority and challenging for cloud service providers [13, 18, 51], particularly as their scale and complexity continue to grow. The cloud overlay networks [21, 25] are a logical network layer built on top of the physical infrastructure using virtualization technologies. Analyzing reliability in cloud overlay networks involves far more dimensions [19, 53], including the massive scale and heterogeneity of cloud overlay network components, the coupling relationships among overlay components and with the underlying physical network, and the interactions between tenants sharing segments of the overlay infrastructure (§2.1).

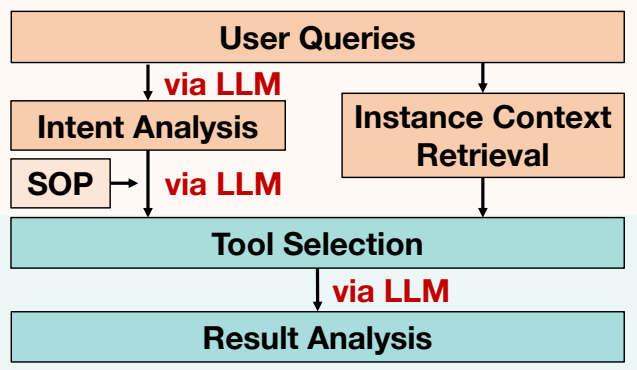


Figure 1. Paradigm of LLM network assistant.

To ensure high reliability, engineers must perform tasks such as *alert triage, change risk assessment, and failure recovery*, collectively referred to as *reliability engineering* (§2.2). For specific network components and architectures, cloud service providers already offer extensive tools for reliability engineering [16, 42, 43, 52], such as network topology discovery, network reachability diagnosis, etc. There are also lots of networking specialists with deep knowledge of particular network segments. However, due to the complexity of cloud overlay networks, there exists no single expert with comprehensive visibility across the entire cloud overlay environment. Even for experts, the ability to correctly identify, effectively apply, and accurately interpret tools for reliability engineering tasks is not guaranteed. The pervasive blind spots pose significant challenges to reliability engineering.

A reliability engineering assistant that automatically leverages existing reliability tools to support cloud overlay reliability tasks is highly desirable. Given task descriptions provided by engineers, such an assistant can automatically interpret the requests, invoke relevant tools, and analyze their outputs. Engineers can thereby complete reliability

tasks without comprehensive knowledge of the entire cloud overlay network. Our prior system iterations (§2.3) have shown that building such an assistant through manual rule engineering alone entails prohibitive effort.

Large Language Models (LLMs) offer a promising alternative for such an assistant. To the best of our knowledge, a paradigm [11, 20, 35, 47, 48] has been established for LLM-powered network assistants that integrate multiple tools for reliability engineering. As illustrated in Fig. 1, upon receiving user queries, the assistant analyzes the queries into two parts: user intent and instance context. User intent refers to the high-level goal of users while the instance context denotes the target network instance set involved in the task, which may change according to the task's progression. Using user intent and instance context, the relevant tools are called according to the selected Standard Operating Procedures (SOPs). The reliability engineering assistant then analyzes the tool outputs and returns the final results to users.

However, our practice in production networks has shown that simply reusing the paradigm is insufficient to address the challenges encountered in cloud overlay networks. The specific reasons are as follows:

- **Multi-Team-Induced Intent Ambiguity.** The intent expressed in user queries is inherently shaped by the submitter's role. For example, the same packet-loss incident may be reported as instance network sluggishness by an elastic compute service (ECS) engineer, as unreachable real servers (RS) by a server load balancer (SLB) engineer, or as degraded business performance by a tenant. Without knowing the submitter's role, an LLM cannot reliably determine that these descriptions refer to the same fault and invoke the appropriate tools. Our overlay network comprises 64 distinct components, far more than in conventional physical networks. Engineers from different component teams differ substantially in how they understand the network. In addition, cloud overlay networks serve tens of thousands of tenants from diverse industries and backgrounds, who further describe the cloud problems from their own perspectives. Moreover, under operational pressure, engineers tend to submit extremely terse queries, e.g., an instance ID with no further description.
- **Coupling-Complicated Instance Context Retrieval.** In practice, the instance cited in an initial alert or failure report often differs from the actual root-cause instance. For example, a user may report an ECS instance failure, while the fault actually lies in a failure on a related SLB, or report datapath packet loss, while the root cause is an invalid configuration from a network controller. The three types of complex coupling relationships among overlay instances (§2.1) make it especially difficult to identify all instances relevant to the initially reported target. Furthermore, cloud overlay networks are highly elastic. Hence, there are frequent configuration changes which continually alter coupling relationships, making timely identification of the current coupled set even harder.
- **Non-Exhaustive SOPs under Proliferating Task Scenarios.** Existing paradigms often rely on SOPs to select appropriate tools from a vast array of available options. However, the vast variety of task scenarios in cloud overlay networks makes it impractical to exhaustively cover all cases with predefined SOPs. For example, a certain SOP specifies packet-loss detection at a fixed sampling rate, but we often encounter cases where packets matching certain rules are dropped, requiring a different sampling strategy for accurate analysis. Moreover, the cost of updating every existing SOP when introducing new tools is rather high.
- **Massive Anomalies in Tool Outputs.** A single reliability task requires invoking multiple tools, and each tool's output often contains multiple anomaly items. This makes identifying the root cause and solution extremely difficult. Moreover, the complex architecture of cloud networks requires correlating more network instances to fully analyze a reliability task. For example, a single physical line-card failure once triggered 28,247 anomaly items across 4,541 devices and instances within 10 minutes in our cloud network, making root-cause localization extremely difficult.

In this work, we propose Dasheng, an LLM-powered reliability engineering assistant for cloud overlay networks. To address the above overlay-specific challenges, Dasheng extends the paradigm in Fig. 1 with four targeted designs, each mapped to one challenge above: (1) profile-based user intent analysis disambiguates intent using team-aware user profiles; (2) trie-tree-based initial instance context retrieval and dynamic instance context management jointly trace coupled instances across evolving overlay topologies; (3) state-machine-driven progressive tool selection iteratively invokes tools beyond static SOP coverage; (4) expert-experience-based result analysis filters propagated anomalies via a four-tuple expert experience library. These modules operate in an iterative loop: upon receiving a query, Dasheng infers intent and resolves the initial instance type, then repeatedly updates instance context, selects and invokes tools, and analyzes outputs until it delivers a final answer or proceeds to another iteration. This design enables Dasheng to autonomously plan tool-selection paths even when no SOP is available.

Dasheng is deployed on production environment of a large-scale cloud service provider, and integrates 139 reliability tools. Dasheng has been in operation for over two years, and assisted 1,450 engineers in resolving approximately 70k reliability tasks. Most Dasheng invocations complete within 3-100 seconds, and reduce case resolution time by up to 97%. In 92% of evaluated cases, Dasheng produces useful outputs that materially assist engineers. Furthermore, Dasheng has been connected to several key reliability engineering platforms within the cloud service provider, including trouble

ticket systems and fault diagnosis platforms. It automates the analysis of engineers' needs, summarizes human-machine dialogues during troubleshooting, and provides actionable recommendations to engineers.

The paper makes the following contributions:

- We characterize four challenges for deploying LLM-based reliability engineering assistants in cloud overlay networks, grounded in multi-year production experience.
- We design and deploy Dasheng to address these challenges in a large-scale production cloud for over two years.
- We distill operational lessons learned from two years of Dasheng deployment in production.

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

2 Background

2.1 The Complexity of Cloud Overlay Networks

Cloud overlay networks provide enhanced elasticity for network services [12, 31]. Increasing applications are migrating to the cloud. However, with the expanding scale, the architecture of cloud overlay networks has become increasingly complex. This complexity is reflected in the diversity of components and their variant forms, intricate coupling among components, and mutual interference between tenants.

The diversity of components. Within our production cloud, there are 64 distinct components in overlay networks, such as gateways and virtual switches. Even a single component may adopt multiple forms [29, 39] to accommodate different situations. For example, the gateways contain forms of P4 switches, virtual machines (VMs), and containers. In large corporations, different components, or even different variants of the same component, are often managed by separate engineering teams. The focus and expert knowledge of different teams on network reliability vary significantly. Engineers from different component teams therefore differ in their reliability focus and domain expertise.

Intricate coupling among components. Components in cloud overlay networks exhibit intricate coupling. As illustrated in Fig. 2, it can be categorized into three types:

(1) Forwarding coupling. The impact of faults or changes can propagate to other components along the packet forwarding path. As shown in Fig. 2, packets traverse multiple components over their lifecycle. Moreover, multi-path routing is common, making it more difficult to determine the actual path taken [28, 40].

(2) Control coupling. A configuration change in one component may indirectly induce configuration errors in another, resulting in system failures. Cloud overlay networks undergo diverse configuration updates, such as VM scaling and routing table updates. Both misconfigurations and delayed configuration propagation can trigger unreliability events. Moreover, reconfiguration occurs at extremely high frequency: for our whole cloud overlay network, daily reconfiguration operations number in the billions [30].

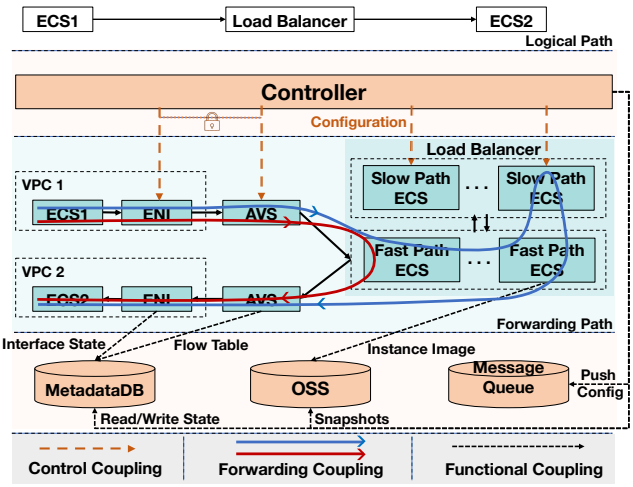


Figure 2. Coupling of cloud overlay network.

(3) Functional coupling. Cloud overlay network components also depend on non-network cloud services, such as storage services. Misconfigurations or failures in these services can likewise be the root cause of overlay network faults. For example, the configuration of Apsara virtual switch (AVS) is stored in cloud storage services [5], and a transient storage disruption can prevent AVS from loading routing state, triggering malfunctions that seem to originate within the overlay network.

Mutual interference between tenants. The high elasticity of cloud overlay networks enables them to serve a large number of tenants [25, 37]. Infrastructure resources, such as storage and bandwidth, are shared across different tenants. A significant portion of network stability issues [52, 53] originates precisely from this resource sharing, stemming from inter-tenant interference. For instance, in practice, we often encounter situations where an abrupt increase in the CPU usage of tenant A can affect the performance of tenant B on the same physical device.

2.2 Reliability Engineering

Reliability engineering in cloud overlay networks is crucial, as reliability lapses can lead directly to substantial economic losses, including service-level agreement (SLA) violations, customer downtime, and reputational damage. SLA in our production cloud is 99.995% [2], which allows for no more than approximately 26 minutes of downtime per year. Reliability engineering can be operationalized through several critical and interdependent tasks, mainly divided into three categories [9, 14, 24]:

- **Alert Triage.** Cloud service providers continuously monitor a vast array of network metrics and generate alerts. Effective triage involves the timely analysis and prioritization of these alerts to distinguish false positives from genuine issues, enabling the preemptive resolution of potential faults before they escalate into larger-scale failures.

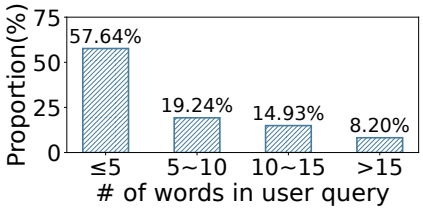


Figure 3. The length distribution of user queries.

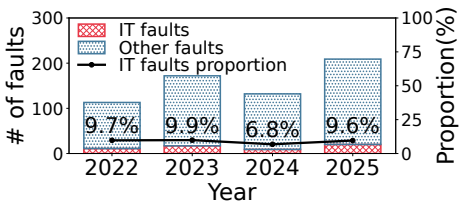


Figure 4. only 10% Faults caused by Initial Target (IT faults).

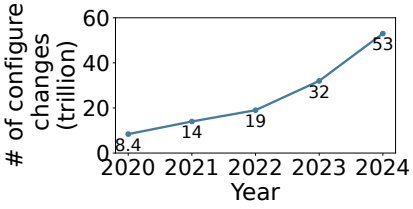


Figure 5. Number of annual configuration changes in our production cloud.

- **Change Risk Assessment.** Based on nine years of empirical data from our production cloud, our analysis shows that nearly 50% of network instability is triggered by changes. Given the high frequency of modifications in cloud overlay networks, a core task is to systematically assess the risk associated with any proposed change before deployment.
- **Failure Recovery.** When a failure occurs despite preventive measures, a rapid response is critical. The focus shifts immediately to executing recovery procedures to restore network performance and service availability. Concurrently, engineers conduct root cause analysis to diagnose the underlying fault. This dual focus on restoration and diagnosis is essential to mitigate impact, prevent recurrence, and avoid further economic loss.

2.3 Generation of Reliability Engineering Assistant

The evolution of cloud overlay network reliability engineering spans two generations.

First Generation. Operators rely on manually written ad-hoc scripts for fault diagnosis, with each scenario requiring a dedicated script authored from scratch. As network scale grows, script coverage becomes severely inadequate.

Second Generation. To address this limitation, we introduce a rule-driven automated diagnosis framework that abstracts scattered scripts into structured rules. Unlike the first generation, where each problem requires a dedicated script, abstract rules organized as decision trees can cover multiple scenarios. This substantially improves diagnostic coverage and automation. However, rules only encode known fault patterns, making the system slow to respond to new failures. Moreover, as the rule set expands, the rule base itself becomes an operational burden.

3 Motivation

Prior to transitioning reliability engineering to LLM-powered approaches, we have accumulated years of operational experience in cloud overlay networks. The complexity of cloud overlay networks makes it infeasible to directly reuse tools from industry peers [8, 11, 20, 35] or adopt solutions proposed in academia [46–48]. We present relevant statistics and representative cases in production cloud overlay networks to illustrate the implementation challenges of LLM-powered reliability engineering assistant.

3.1 User Intent Analysis

The reliability engineering assistant requires users to input queries through an interactive interface. The users of our reliability engineering assistant are primarily operations engineers. During the trial operation of Dasheng, we have observed that users tend to submit highly ambiguous queries. We have collected 120k user queries submitted to the Dasheng assistant during its six-month trial period. As illustrated in Fig. 3, 57.64% of the queries comprise fewer than five words[1]. Many of these short queries contain only a target instance ID and ambiguous directive verbs such as "diagnose" or "check". Even when a query includes a problem description, the description is usually extremely terse, such as "throughput drop" or "network port abnormal".

However, we observe that in cloud overlay networks, identical short descriptions can refer to entirely different problems that require different diagnostic workflows. Conversely, different descriptions can sometimes point to the same underlying issue. For example, when a customer reports widespread request failures with increasing application load balancer (ALB) error logs, different teams would issue totally different queries to Dasheng. The ALB team asks whether the error logs originate from the ALB itself or from the backend RSs while the network load balancer (NLB) team inquires whether an ALB-on-NLB instance exists and whether the NLB status is abnormal. This arises because users of the reliability engineering assistant often come from different teams, with different background knowledge and understanding of the network. Without knowledge of a user's background, LLM cannot reliably infer the true intent from such terse queries alone.

> ***Takeaway 1**: To accurately interpret ambiguous user intent, user profiles based on user role and history could serve as an effective method.*

3.2 Instance Context Maintenance

In reliability engineering tasks, one or a set of target instances, such as faulty instances or change subjects, must be clearly defined in user queries. They are essential inputs for reliability engineering tools. The reliability engineering assistant typically receives string-type input as target instance object, which identifies the initial target instances. To

[1]Word count applies to Chinese characters or English words.

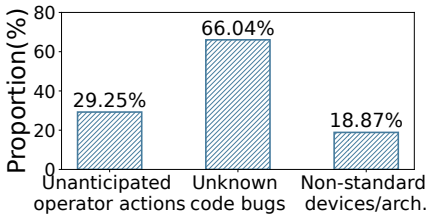


Figure 6. Distribution of SOP coverage gap reasons.

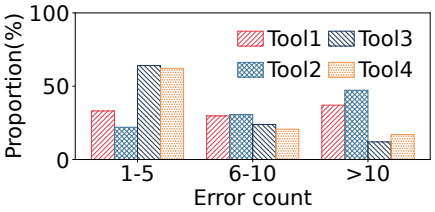


Figure 7. Distribution of anomaly item number for different tools.

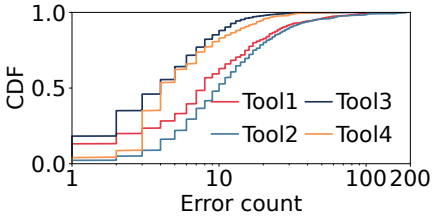


Figure 8. CDF of anomaly item number for different tools.

process the input, the assistant must perform two key tasks: (1) locate the corresponding target instances based on the input strings; (2) identify the associated instances that are coupled with the initial target instance set through any of the coupling relationships described in §2.1.

However, this process has been proven challenging to implement reliably. First, the translation from strings to an instance is inherently complex. User inputs typically consist of instance ID constructed following certain rules or other characterizing fields, such as IP addresses. To map the strings to the corresponding instance, we first need to determine which component it belongs to. Querying the instance tables of all components is not an effective approach, since the query overhead becomes prohibitive. The rule library, manually defined by experts, also proves to be inadequate, suffering from low coverage. In practice, we observe that cloud overlay networks typically undergo several product launches or retirements annually, each introducing or removing hundreds of rules. These changes could sometimes be unsynchronized across different availability zones. Furthermore, cloud overlay networks contain an amount of internal test instances, whose naming rules differ from those of public instances. Given the scale of modern cloud overlay networks, the manual cataloging of such naming rules is practically infeasible.

> ***Takeaway 2**: The naming rule sets associated with instance ID or characterizing fields of cloud overlay network components necessitate a continuous automated process for parsing and updating.*

Furthermore, maintaining only the initial target instances is insufficient for reliability tasks. A significant number of reliability tasks require analyzing not only the initial target instance but also the instances coupled to it. We have statistically analyzed fault records from our production cloud over the past four years, specifically measuring the proportion of incidents directly attributable to the initial target instance. As shown in Fig. 4, we find that only approximately 10% of annual faults are conclusively linked to the initial target instance. Similarly, routine reliability tasks, such as alert triage and change risk assessment, also depend heavily on analyzing the state of coupled instances. However, it is highly challenging for a reliability engineering assistant to identify all coupled nodes based solely on the initial target instances, because of the complex coupling relationships in cloud overlay networks as depicted in §2.1.

Additionally, the dynamism of cloud overlay networks amplifies the difficulty in parsing coupling relationships. As shown in Fig. 5, cloud overlay networks undergo tens of trillions of configuration changes annually, corresponding to an average of hundreds of millions of changes per day. The majority of these configuration changes lead to alterations in network topology and shifts in coupling relationships between instances. Moreover, during the execution of reliability tasks, the coupling relationships within the network may continue to evolve, making it more difficult to identify the most current set of coupled nodes.

> ***Takeaway 3**: To model complex network coupling relationships, a hierarchical topology analyzer capable of dynamic updates in response to real-time network topology changes is required.*

3.3 Tools Selection based on SOPs

To support various reliability engineering tasks, engineers from different cloud network component teams have developed a considerable number of reliability tools. The reliability tools fall into several categories: configuration retrieval, network reachability/packet-loss diagnosis, change-risk identification, and network recovery. Currently, the number of tools available within our production cloud has reached 139, and this figure is expected to continue growing in the future. In conventional procedures, operations engineers rely on a substantial number of SOPs to indicate the precise selection and sequence of reliability tools to be employed. SOPs are documented, step-by-step runbooks for diagnosing and recovering a specific class of problems. For example, for network packet loss, a typical SOP guides engineers to first invoke tool A, then proceed with subsequent steps based on its output. However, we observe widespread SOP failures in practice. Specifically, for failure recovery tasks, our analysis indicates that approximately 70% of fault recovery tasks cannot be effectively resolved by following the existing SOP workflows.

We have categorized the reasons for SOP coverage gaps, as detailed in Fig. 6. Approximately 66.04% of the cases stem from unknown code bugs, which typically escape standardized quality assurance (QA) processes due to their highly

specific or idiosyncratic nature, making predefined SOPs infeasible. Another 29.25% are triggered by unanticipated operator actions, while 18.87% originate from devices or architectures that deviate from standardized designs. These scenarios fall outside the scope of existing SOPs.

> ***Takeaway 4**: In addition to the SOP library, the LLM-based reliability assistant should also be equipped with a high-coverage, accurate mechanism for tool selection to address scenarios where existing SOPs provide no coverage or where designing a definitive SOP is impractical.*

3.4 Tool Results Interpretation

After completing tool invocations, the LLM-powered reliability engineering assistant must analyze the tools' output. However, without sufficient domain knowledge, the LLM has a minimal chance of accurately interpreting the output. These tool outputs are often lengthy and rich in domain-specific knowledge with combinations of distinct anomaly items. Hence, we analyze the number of anomaly items contained in a total of 80k invocations across several major reliability tools in our production cloud over six months. As shown in Fig. 7, it indicates that 35%-78% of tool invocations generate over five anomaly items. In the most extreme instance, a single fault triggered 28,247 anomaly items. The execution of a single reliability task often necessitates invoking several to tens of tools. The anomalies reported by each tool are cumulative, leading to an extensive set for LLM processing. Extensively long lists of anomaly items can overwhelm the LLM's reasoning capacity, thereby compromising its focus and the accuracy of the subsequent reliability analysis.

Based on our operational practice, we observe that anomaly items and their root causes are often interrelated. Typically, a single anomaly item corresponds to one or a few underlying causes. Furthermore, the anomaly items returned from a single reliability tool invocation frequently share the same or highly similar root causes. Translating anomaly items into their corresponding root causes is able to reduce the complexity of tool outputs and improve readability, thereby enabling both LLM and human users to more readily comprehend the results of reliability tasks.

> ***Takeaway 5**: Summarizing root causes for anomaly items prior to reliability assistant analysis could significantly enhance the accuracy of its assessment for the output of reliability tools.*

4 System Design

We present Dasheng, a reliability engineering assistant for cloud overlay networks. The Dasheng framework follows the basic paradigm illustrated in Fig.1, but redesigns the implementation of each module according to the characteristics of cloud overlay networks discussed in §3. As shown in Fig. 9, Dasheng consists of five main modules:

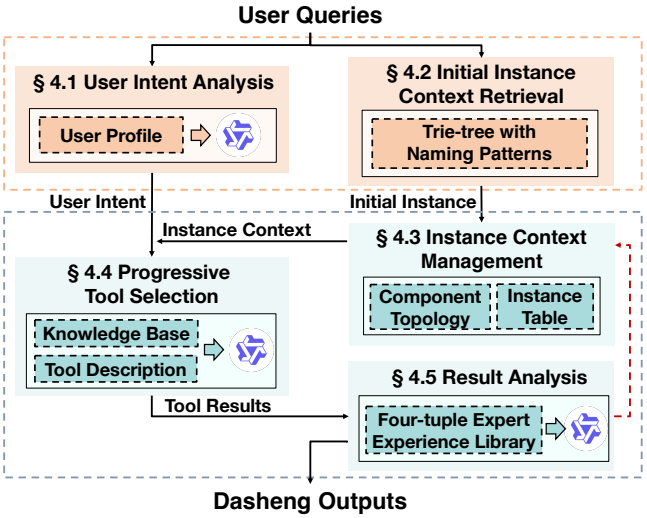


Figure 9. Overview of Dasheng.

- *Profile-based User Intent Analysis* (§4.1) employs a self-reflection classification mechanism that is based on user profiles to address ambiguous user intents (*Takeaway 1*).
- *Trie-tree-based Initial Instance Context Retrieval* (§4.2) incorporates a trie-tree-based component search mechanism that dynamically maintains naming patterns of component instance IDs and efficiently resolves component types from queries (*Takeaway 2*).
- *Dynamic Instance Context Management* (§4.3) decouples stable component topology from dynamic instance topology, enabling efficient retrieval of associated instance context through inter-table indexing *(Takeaway 3)*.
- *State-machine-driven Progressive Tool Selection* (§4.4) adopts a state-machine-driven iterative analysis workflow that progressively selects and invokes reliability tools beyond the limited coverage of static SOPs *(Takeaway 4)*.
- *Expert-experience-based Result Analysis* (§4.5) automatically parses outputs from various reliability tools and provides intent-aware root cause analysis and solutions through intelligent aggregation with a well-defined four-tuple expert experience library *(Takeaway 5)*.

4.1 Profile-based User Intent Analysis

As depicted in §3.1, the ambiguity of user inputs prevents Dasheng from accurately recognizing their intents. To mitigate this issue, Dasheng incorporates user profiles to assist intent recognition. Each user profile consists of two modules: user team affiliation and their historical background, from which a top preferred intent is derived. However, it cannot be assumed that each reliability task request associated with users will always align with their team context or historical background. Therefore, we design a self-reflective intent classification mechanism along with user profiles. The LLM self-assesses whether user-profile information is needed to infer the user's intent accurately.

The LLM analyzes the user queries to derive the user intent, that is, the ultimate goal of the current reliability task, such as diagnosing the root cause of an alert or identifying the source of risk associated with a recent change. Simultaneously, the LLM self-assesses the completeness of the information in the input prompt. If the LLM evaluates the prompt as clear, it directly outputs the inferred user intent to the next module. If the prompt is assessed as ambiguous, the LLM incorporates the user profile. Based on the top preferences recorded in the user profile, the LLM derives the refined intent. This derived intent is then presented to the user for confirmation of its correctness.

In general, Dasheng handles three categories of user intent, corresponding to the three primary reliability tasks described in §2.2. For alert triage tasks, the user typically aims to determine whether an alert is a false positive and, if it is genuine, to identify its root cause and the appropriate recovery procedure. For change risk assessment tasks, the intent is generally to evaluate if a proposed change complies with standardized procedures, represents the optimal solution, and conflicts with other ongoing changes or network activities. For failure recovery tasks, which are the most urgent reliability tasks, the primary objective is to restore network operations as swiftly as possible. The secondary goal is to determine the root cause of the failure.

4.2 Initial Instance Context Retrieval

In reliability tasks submitted to Dasheng, one or a set of target instances must be explicitly specified. In user queries, the target instance is identified by an instance ID or a characterizing field, such as an IP address. If the target instance ID is omitted, Dasheng interactively prompts the user to provide it. User queries, typically presented in textual form, often do not explicitly specify the component type of the target instance. Without the correct component type, the accuracy of subsequent tool selection would be significantly compromised. Therefore, Dasheng records the naming patterns of different component instance IDs and uses the naming pattern set to retrieve the user queries.

As illustrated in §3.2, the naming patterns across different component types exhibit high variability and are subject to numerous exceptions. Hence, Dasheng adopts a trie-tree-based component search scheme to automatically and efficiently identify component types from instance identifiers. As shown in Fig. 10, the trie-tree-based component search scheme operates through two coordinated phases: Trie Tree Maintenance phase and Query Processing phase.

Trie Tree Maintenance. Dasheng dynamically constructs and maintains a trie tree that encodes the regular expression patterns characterizing the instance IDs of cloud overlay network components. During the initialization phase, the trie tree is constructed by processing component instance ID tables to cluster IDs by structural patterns, from which a set of regex rules is derived to generalize each component's identifier format. To guarantee the trie tree's ongoing correctness amid updates of cloud overlay networks, a revalidation routine periodically updates the trie tree. The revalidation routine first samples the latest instance ID tables. The sample size is dynamically calculated based on the current total number of instances, preventing excessive sampling to maintain verification efficiency. Then, the sample instance IDs are concurrently validated against the trie tree during the revalidation routine. Once a regular expression is matched, its timestamp is updated. If a new pattern is detected, a corresponding regular expression is generated and inserted into the trie tree. Regular expressions corresponding to old patterns will be removed if their timestamps have not been updated for consecutive cycles.

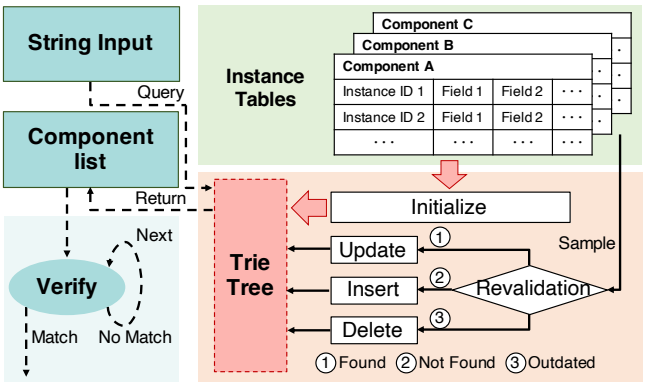


Figure 10. Trie-tree-based component search.

Query Processing. When a new reliability task generates an initial instance set, Dasheng first utilizes LLM to parse the queries and extract the relevant instance strings. Each string is then fed into the trie tree to retrieve the component type matching the longest prefix. The system subsequently searches the instance table corresponding to this primary candidate type for a matching instance. If a match is found, the result is returned to Dasheng. Otherwise, the system iteratively queries the trie tree for the next-most-associated component type and repeats the search in its respective table. This process continues until a matching instance is located or the candidate list is exhausted, ensuring robust and precise component resolution.

4.3 Instance Context Management

As discussed in Takeaway 3, identifying and analyzing the devices and instances associated with the initial target device is a critical task in cloud overlay network environments. However, the highly coupled and dynamic nature of cloud networks makes it extremely challenging to accurately locate these associated devices and instances. In cloud networks, instances represent the runtime entities of specific cloud components. Instance topology dynamically adjusts according to the frequent configuration changes, whereas the component topology remains relatively stable. Hence, we design a dynamic device context management module. The core idea of

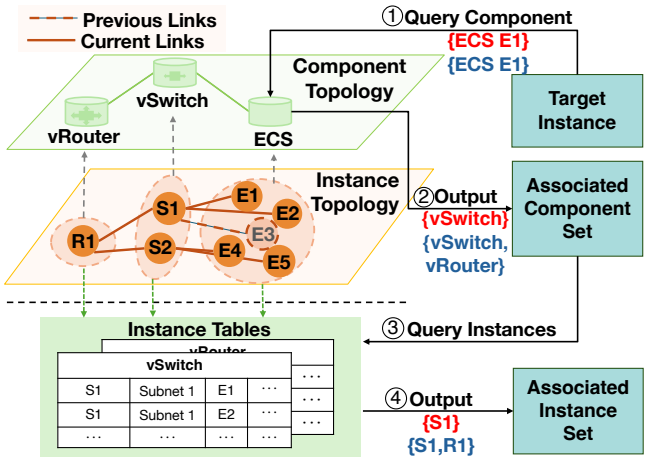


Figure 11. Dynamic instance context management. An example provided for querying instances associated with E1 (red for $depth = 1$, blue for $depth = 2$).

the module is to decouple the cloud network by leveraging the stable topological relationships between components, enabling efficient retrieval of dynamically coupled instances.

As shown in Fig. 11, when Dasheng retrieves the target component type, it first queries the component topology and fetches the set of components associated with the target component. Then, it queries the instance tables of associated components. The instance table is essentially a dimension table, which allows for rapid identification and retrieval of the associated instances using the information provided.

Moreover, the module allows users to specify the depth of the component topology search. The search depth determines the maximum number of hops between an associated component and the initial component. As shown in Fig. 11, a depth of 1 corresponds to components one hop away, while a depth of 2 includes components within two hops. A greater depth enables broader component coverage, which helps ensure more comprehensive fault diagnosis. However, incorporating an excessively large set of components at once may cause Dasheng to rely on associations that do not reflect the most current network topology. After the search finishes, the module aggregates the instance information from all associated components and outputs it in a structured JSON format, providing Dasheng with a comprehensive and real-time instance context.

4.4 Progressive Tool Selection

To address the limited scenario coverage of traditional SOPs, Dasheng incorporates a state-machine-based progressive tool selection mechanism. As illustrated in Fig. 12, Dasheng maintains a task state machine for each reliability task. After completing user intent analysis §4.1 and initial instance context retrieval §4.2, the task finishes its initialization phase and enters an iterative analysis loop. The loop iteratively executes a three-step cycle: instance context update, tool selection, and output analysis.

Dasheng first performs an initial instance context update and tool selection based on the parsed information from initialization and the tool descriptions, then invokes the corresponding tools to obtain their outputs. These tools are encapsulated as MCP tools with standardized inputs and outputs for Dasheng to invoke. Next, Dasheng proceeds to the output analysis state to examine the outputs of the tool. Based on this analysis, Dasheng determines whether to proceed to the next iteration or to deliver the final answer. Details regarding the state transitions of the task state machine are described below. Besides, Dasheng employs a knowledge-enhanced tool selection mechanism to improve the accuracy of tool selection. Implementations of the instance context update and output analysis states are described in §4.3 and §4.5, respectively.

Progressive Task State Update. The task state machine in Dasheng operates in a progressive manner through five distinct states: initialization, tool selection, output analysis, instance context update, and task termination.

Upon successful initialization, which includes user intent parsing and initial instance context retrieval, Dasheng transitions to the instance context update state described in §4.3. Then Dasheng enters the tool selection state to select the appropriate tools. Valid tool selections lead to the output analysis state, where the results from the tools are summarized. If no valid tools are selected, the system transitions to a failure termination and asks for more information from the user, such as additional instance context.

The output analysis state is responsible for determining the next state based on the analysis of the tool output. If the task is deemed complete and the user intent is fully resolved, the system transitions to successful termination. If the tool outputs are deemed insufficient to draw a conclusion and further tool invocations are required, Dasheng returns to the tool selection state for another iteration. If analysis of the current instance context alone is considered inadequate, Dasheng transitions to the instance context update state to dynamically update the instance context, progressively incorporating related instances to ensure analysis accuracy.

The task state machine terminates under one of three conditions: (1) success termination, when the user intent is fully resolved; (2) active abort, which occurs due to user request, exhaustion of the topology traversal depth, or a tool execution timeout; (3) failure termination, which occurs in cases of initialization errors, no valid tools found, invalid outputs, context update failures, or system exceptions. Termination takes precedence over all other states, ensuring that the task halts as soon as a terminal state is reached.

Knowledge-Enhanced Tool Selection. Based on the initial target instances and user intent obtained, Dasheng employs a BM25+ANN dual-tower hybrid retrieval model [44] to search for an appropriate set of tools with the help of

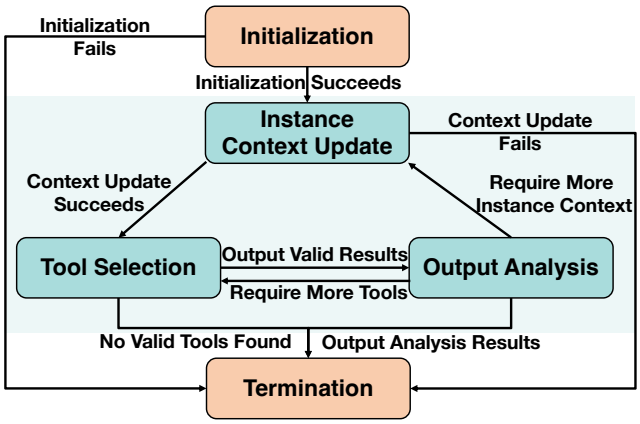


Figure 12. State Machine of Progressive Tool Selection.

knowledge with SOPs. However, as noted in Takeaway 4, the coverage of SOP is limited. In most reliability engineering scenarios, SOP fails to provide a comprehensive solution. This limitation often results in failure to identify the appropriate tools for a given task.

To address this issue, Dasheng provides a dedicated profile for each tool, which includes detailed descriptions of the tool's invocation methods, applicable scenarios, and certain small-sample tricks. These descriptions are integrated into the prompt, enabling the LLM to intelligently select the most suitable toolset for the current task based on information such as target component types. To prevent overwhelming the LLM's attention with excessive information at once, the descriptions are loaded progressively.

4.5 Experience-based Results Analysis

To address the often verbose and highly specialized outputs generated by reliability tools, which typically require expert knowledge to interpret, Dasheng incorporates an expert-experience-based results analysis module. The output of these tools generally consists of one or multiple sets of anomalies. Empirical observation indicates that specific anomalies frequently correlate with distinct root causes. Once a root cause is identified, Dasheng can readily generate recommendations aligned with the user's intent.

Hence, we design a four-tuple expert experience library. Each entry in this library comprises four key attributes: the anomaly item, the observed network phenomenon, the root cause, and the corresponding resolution. Dasheng leverages the library to parse the results from reliability tools and proposes tailored solutions based on the user's intent.

Expert Experience Library Construction. Anomaly items, typically defined during the design of a reliability tool, represent an abnormal judgment on a specific metric or an active probe targeting instances or network links. The observed network phenomenon refers to the perceptible performance variation experienced by tenants. Based on the anomaly item and the observed phenomenon, experts can determine one or several root causes. Correspondingly, once the root cause is identified, experts can propose targeted solutions, such as recovery procedures for failure recovery and alert triage, or revised change proposals for change risk assessment.

For reliability tools designed after Dasheng, engineers can be required to maintain documentation that specifies the root causes and corresponding resolutions for the tool's anomaly items under common network phenomena. However, a large number of reliability tools had already been designed and deployed before the development of Dasheng. Furthermore, as cloud networks evolve, new root causes and solutions may emerge that have not been anticipated during the initial tool design. To address this, Dasheng also maintains an adaptive mechanism for updating the expert library based on historical data. Cloud network operations routinely maintain a historical database of failures, alerts, and changes. Engineers are required to evaluate the outcome of each task. Utilizing the engineer-annotated data, Dasheng periodically reviews and updates the four-tuple expert experience library by adding, modifying, or removing records, thereby continuously expanding and refining its knowledge base over time.

Anomaly Item Aggregation. During each execution cycle, Dasheng invokes multiple reliability tools and collects their returned results, which contain descriptions of various anomaly items. Dasheng first queries the four-tuple expert experience library using the reported anomaly items to retrieve a list of associated root causes and corresponding resolutions. The LLM then processes this list by aggregating closely related root causes and resolutions, ultimately generating a response tailored to the user's intent.

5 Deployment

Dasheng has been deployed in our large-scale production cloud for two years. It has served approximately 1,450 engineers, handling about 70k reliability tasks. Dasheng has been automatically integrated into the trouble ticket platform, actively participating in the workflow of operations engineers to resolve user issues. Furthermore, it is deeply embedded in fault recovery scenarios. Upon the occurrence of incidents, Dasheng automatically notifies the relevant operations engineers based on the severity and affected components, establishes incident response groups, and responds automatically to engineer requests by invoking reliability tools. It also intelligently summarizes tool outputs and the discussion of engineers.

To address the deployment challenges caused by the high complexity of cloud overlay networks, Dasheng adopts the following design in its system architecture:

LLM Selection of Dasheng. Dasheng employs different LLMs across its key stages. Specifically, the user intent analysis module utilizes the qwen-turbo model [4, 50] to ensure rapid computational speed. The tool selection module employs qwen-plus-latest [3, 50], a model that offers a balanced

trade-off between speed and accuracy. For result analysis module, which processes longer inputs with higher token demands, the system uses a separately deployed qwen-30b model [6, 50].

Distributed deployment for multiple available zones: Due to the lack of configuration synchronization across different availability zones in the cloud, we employ a multi-unit distributed architecture. Dasheng maintains separate data structures, such as trie trees and instance tables for different available zones. The distributed data structures ensure both the timelines and accuracy of Initial Instance Context Retrieval (§4.2) and Instance Context Management (§4.3).

Standardized Tool Adapter Layer: To integrate 139 existing reliability tools developed by different teams, we have designed a standardized adapter layer. The layer normalizes the heterogeneous inputs and outputs of different tools into a system-recognizable format. When a new tool is introduced, it can be embedded into Dasheng simply by registering its identifier and description according to the defined schema. Conversely, removing an existing tool requires only the deletion of its corresponding descriptor. This approach enables a modular, plug-and-play management of tools.

Asynchronous Tool Invocation Engine: When processing a single reliability task, Dasheng may involve multiple tool invocations. Sequential calls can lead to prolonged execution times. To address this, the core engine of Dasheng adopts a queue-based asynchronous execution model. It dispatches calls to subsequent tools without waiting for the results of previous ones. The asynchronous mechanism ensures the continuity of reliability tasks even under network fluctuations or slow individual tool execution.

6 Evaluation

We evaluate Dasheng in three aspects: (1) the end-to-end performance of Dasheng using logs collected during its operation in our production cloud, demonstrating its practical deployability (§6.1); (2) ablation studies to quantify the contribution of key modules (§6.2); (3) evaluations on individual modules of Dasheng to demonstrate the effectiveness of its design (§6.3); (4) several representative cases of Dasheng to improve production efficiency, covering different types of reliability tasks such as alert triage, change risk assessment, and failure recovery (§6.4).

6.1 Dasheng Usage in Production

We collect logs from one year of Dasheng's production deployment. The logs include queries submitted by users (e.g., operations and development engineers), Dasheng's analysis outputs, and its execution time. A subset of the logs also includes a human-verified root cause and solution provided by frontline operations engineers.

Calling times of Dasheng. As shown in Fig. 13, we summarize the calling times of Dasheng API. The use frequency of Dasheng has increased steadily over time. The growth is attributed to the integration of Dasheng into multiple platforms, including trouble ticket systems and failure-emergency response systems. Statistics show that Dasheng has been used by 1,450 engineers from 558 teams to date, with approximately 70k invocations in total[2]. The 1,450 engineers include both operations engineers and development engineers from individual cloud network components. The two groups differ substantially in background, and Dasheng benefits both.

Accuracy time of Dasheng. We invite senior engineers to evaluate 809 production cases. Engineers score Dasheng's analysis outputs against the human-verified root cause and solution on a 5-point scale: 5 if the output fully matches the human-verified root cause and solution. The solution it gives is usable without edits; 4 if only minor edits are needed to make the Dasheng's output usable; 3 if the direction is correct, but a little further manual investigation is still required; 2 if the output is only partially related to the correct direction; and 1 if it is entirely incorrect and not helpful. The results are shown in Fig. 14. In 92.0% of cases, Dasheng produces useful outputs that materially assist engineers.

Execution time of Dasheng. As shown in Fig. 15, the execution time of Dasheng is mostly distributed within a range of 3s to 100s. 89.63% of Dasheng calls complete within 100 seconds, demonstrating minute-scale troubleshooting performance of Dasheng. For comparison, we construct reliability tasks from two network components, with 10 cases per component. We ask senior and novice engineers to resolve these cases with and without Dasheng, and measure the resolution time. As shown in Fig. 16, with Dasheng, senior engineers reduce resolution time to 13% (component 1) and 12% (component 2) of that without Dasheng; novice engineers reduce it to 8% (component 1) and 3% (component 2). Dasheng enables novice engineers to match senior-engineer efficiency in a short time.

6.2 Ablation Study

We construct 60 diagnostic scenarios, comprising 20 alert triage, 20 recovery recommendation, and 20 change risk assessment tasks. On four base models (Qwen3.6-Plus, Qwen3.8-Max, Qwen3.8-Flash, and Qwen3.7-Plus), we evaluate the complete Dasheng and four ablated configurations that remove intent analysis (−Intent), context management (−Context), progressive tool selection (−ToolSel), and result analysis (−Result), respectively. Each scenario is executed three times, yielding 180 runs per configuration. We evaluate the end-to-end task completion rate, defined as the fraction of runs

[2]Some of the 1,450 engineers are development engineers who are not full-time reliability engineers, but instead participate on an on-call rotation every few months.

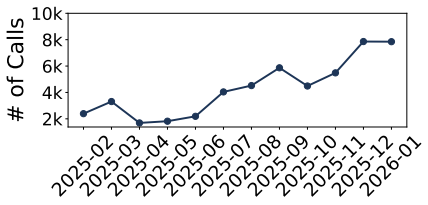


Figure 13. Statistics on calling times the Dasheng API in one year.

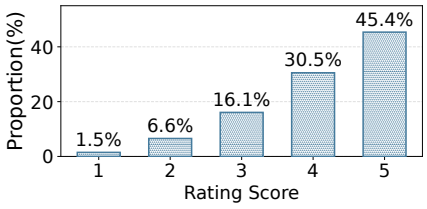


Figure 14. Accuracy scores of Dasheng's end-to-end performance.

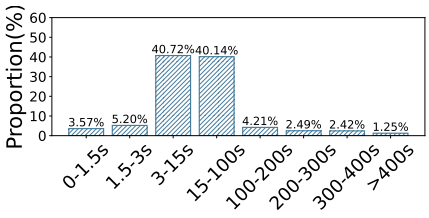


Figure 15. Distribution of Dasheng's end-to-end processing time.

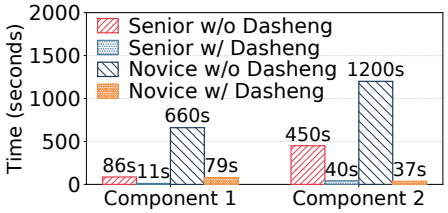


Figure 16. Time comparison between Dasheng and human.

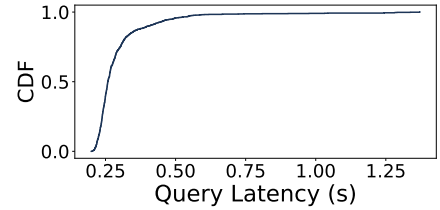


Figure 17. The query latency CDF of Initial Instance Context Retrieval.

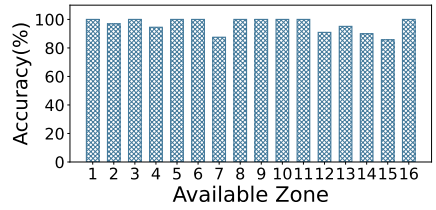


Figure 18. Accuracy of instance context management.

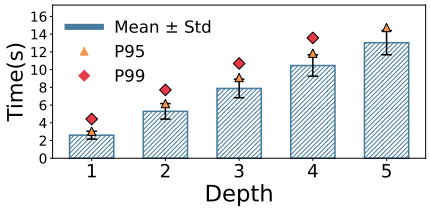


Figure 19. Context retrieval time.

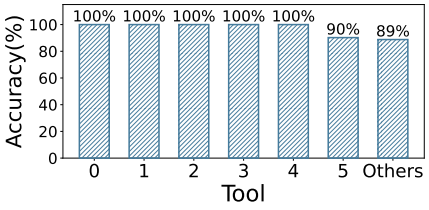


Figure 20. Accuracy of tool selection.

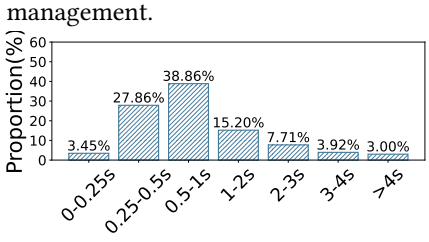


Figure 21. tool selection time.

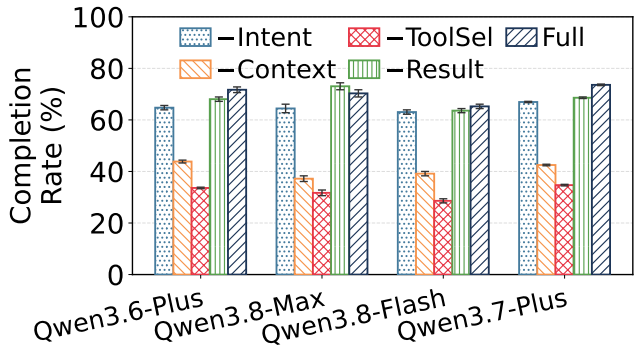


Figure 22. Ablation test of Dasheng.

in which the system output satisfies the predefined key requirements of the scenario. As shown in Fig. 22, all four modules improve the end-to-end task completion rate, with context management and progressive tool selection contributing most.

6.3 Module-level evaluation

Intent analysis module. We randomly sample 500 cases and compare Dasheng's accuracy with and without this module. When intent analysis is enabled, we use different LLM models for this step. As shown in Table 1, adding intent analysis improves accuracy by up to 19.1%.

Initial Instance Context Retrieval. To evaluate the Initial Instance Context Retrieval module, we randomly sample instances from the recent invocation logs of Dasheng and

Table 1. Impact of Intent Analysis Module for Dasheng.

Model	w/o	w/	Improvement
Qwen3-Max	74.9%	91.8%	+16.9%
Qwen3.6-Plus	87.4%	95.1%	+7.7%
Qwen3.6-Max-Preview	80.9%	95.6%	+14.7%
GLM-5.1	75.4%	94.5%	+19.1%

independently query the trie tree maintained by Dasheng to retrieve the corresponding component type for each instance. We filtered out aberrant measurements caused by the already deprovisioned instances. The average query latency was 0.30*s*, with P95 latencies of 0.49*s* and P99 of 0.93*s* respectively. The CDF of the query latency is shown in Fig. 17. We further evaluate the retrieval accuracy across 20 different component types. These instances are analyzed against two models: a rule set manually defined by expert engineers and the trie tree produced by Dasheng. As presented in Table 2, the manual rule set cover 60% of the components. In contrast, the trie tree generated by Dasheng demonstrates significantly higher effectiveness, accurately analyzing nearly every component.

Instance Context Management. Based on the same sampled instance set from the previous section, we further evaluate the performance of the Instance Context Management by querying the knowledge graph to retrieve instance context. The accuracy of instance context management, shown in Fig. 18, exceeds 85% across all available zones, with 9 zones

Type	1	2	Type	1	2	Type	1	2
uid	✗	✓	vm	✗	✓	xgw	✓	✓
vbr	✓	✓	lb	✗	✓	cen	✓	✓
slb	✓	✓	alb	✓	✓	sag	✗	✓
sgw	✗	✓	vpc	✓	✓	br	✗	✓
vswitch	✓	✓	c.s.	✗	✓	vpn	✓	✓
nc	✓	✓	nat	✓	✗	b.p.	✓	✓
ap	✗	✓	eip	✓	✓			

Table 2. Comparison of Rule-based and Dasheng retrieval results across component types (Type 1 denotes the rule-based method, Type 2 denotes Dasheng.)

Table 3. Impact of Expert Experience Library Module.

Model	w/o	w/	Improvement
Qwen3-Max	83.3%	99.2%	+15.9%
Qwen3.6-Plus	85.4%	100.0%	+14.6%
Qwen3.6-Max-Preview	88.5%	100.0%	+11.5%
GLM-5.1	86.6%	100.0%	+13.4%

reaching 100%. The latency for extracting context at different topology depths is measured, with the mean, P95, and P99 values reported in Fig. 19. The results indicate that latency increases almost linearly with tree depth, with an average increase of 2~3 seconds per additional layer.

Tool Selection. To obtain the ground truth for tool selection, we extracted 500 records of Dasheng invocations from historical data. Then, experienced engineers annotate them for the correctness of tool selection. The accuracy of tool selection ranges from 89% to 100% among different tools (Fig. 20) and 97% of tool selection times are under 4*s* (Fig. 21).

Result analysis module. We select 192 cases across 24 scenarios. The two ablations use different case sets because the intermediate steps of some cases cannot be reproduced and are therefore excluded. As shown in Table 3, the result analysis module improves accuracy by up to 15.9%.

6.4 Case Study

Through several real cases from our production cloud that are handled by Dasheng, we analyze the additional benefits Dasheng provides to engineers across three key reliability tasks: alert triage, change risk assessment, and failure recovery.

Case 1: Load-Balancer Alert Triage. Numerous alert mechanisms are deployed in cloud overlay networks. Taking NLB as an example, when an alert is triggered, engineers must not only examine the NLB instance itself, including its Docker configuration and security group settings, but also investigate related components such as ECS, CStar [49], the underlying physical network, etc. With the help of Dasheng, the processing procedure of NLB alert triage case is as follows. First, Dasheng automatically determines that this is an alert triage event and defines the instance context as NLB, the associated context chain as LB-ECS-Docker and the Service Discovery Groups (SDG)—CStar [26]. The diagnostic process initiates by retrieving LB configuration details and invoking a "one-click diagnosis" tool to determine whether the anomaly stems from internal misconfiguration.

Anomaly metrics are analyzed based on health check of 5-tuples. Consecutive failures exceeding 10 suggest client-side Docker networking issues requiring a container restoration, while fewer than 5 indicate transient path jitter. Multi-client failures signal systemic issues, whereas single-client failures point to localized ECS or Docker anomalies. Short-lived connection failures are typically linked to network jitter or slow-path service discovery (SD) faults; long-lived connection failures are predominantly traced to the CStar Protocol Translator (PT) subsystem.

Further root-cause analysis involves a topological drill-down. Inspection of LB Service Discovery Groups (SDGs) determines the scope of the impact: if multiple SDGs are affected, the issue is deemed systemic; conversely, a single anomalous SDG typically indicates an LB SD fault. At the device level, packet loss recorded on the physical or virtual device hosting the LB SD suggests performance degradation, which leads to packet drops.

Additionally, CStar-layer telemetry, which includes metrics and configuration changes for components such as PT and Group, is analyzed. Confirmed anomalies at the CStar layer are generally indicative of platform-side hardware failures or security incidents, requiring escalation to the CStar on-call team.

Finally, the system checks for concurrent Zoonet alerts in the same geographical region. The presence of such alerts strongly suggests underlying physical network impairments, necessitating intervention by the physical network operations support team.

The above procedure is automatically triggered by Dasheng, which autonomously determines the subsequent direction of analysis and the specific tools to be invoked. Novice engineers often struggle to fully comprehend the architecture of each component and the proper application of each tool. In contrast, Dasheng has the ability to use reliability tools integrated into it correctly and analyze inter-component dependencies.

Case 2: Change Risk Assessment. Manual assessment of change risk typically relies on experience or SOPs, which can easily overlook potential conflict risks. Additionally, delayed notifications of SOP updates or other service changes may lead to errors. The manual process of querying information is also time-consuming. In contrast, Dasheng automatically traverses all relevant SOP documents, conflicting components, and related services during each invocation, conducting a comprehensive risk assessment.

When a change request is submitted for a cloud networking component, Dasheng automatically analyzes the CHANGE case and identifies the core components affected by the change. Dasheng automates the assessment of both compliance and business risks. Compliance risks pertain to whether the submission time, execution window, gray-release strategy, and batch intervals conform to Cloud Intelligence's production safety and change management policies. Business risks pertain to potential conflicts arising from concurrent changes (e.g., overlapping CStar and AVS updates on the same ECS fleet) or scheduling during critical customer periods, such as peak traffic in a "Key Customer Protection" campaign. Dasheng automatically extracts key information from the change request, dynamically loads current compliance rules, and invokes the Change Platform and Business Platform APIs to detect concurrent changes and customer conflicts. This enables rapid, accurate, and scalable approval while maintaining real-time alignment with evolving operational standards.

Case 3: Elastic IP (EIP) Throughput Degradation. EIP bandwidth failure represents a classic type of network failure in cloud operations. The conventional recovery procedure can be broken down into several steps, including information querying, fault diagnosis, and service restoration. Even for an experienced engineer, this process typically requires navigating six to seven different web interfaces to gather and correlate information. Even under smooth conditions, the entire workflow still takes approximately twenty minutes.

When Dasheng is employed, however, the same workflow can be completed within three minutes. The engineer needs only to provide the ID of the problematic instance. After ascertaining the user's intent, Dasheng parses the instance type as EIP, analyzes the network topology. Based on the initial network topology, Dasheng identifies the cluster to which the EIP belongs, together with the corresponding ECS and physical device IDs. Leveraging this foundational information, empirical data relevant to EIP throughput degradation (e.g., the corresponding SOPs) are supplied to Dasheng as background data. Dasheng then invokes the relevant tools, analyzes the results, and finally outputs the diagnostic root cause. This procedure significantly shortens fault recovery time and greatly reduces the engineer's manual effort.

7 Lesson Learned

Tenant Application Topology Matters. As discussed in §3.2, accurately resolving cloud overlay network topology is essential for correct execution of reliability tasks. However, our deployment experience shows that the topology of tenants' own applications must also be taken into account. For example, a user once reported performance degradation on a load balancer (LB). Dasheng spent considerable effort diagnosing the corresponding LB instance but found no anomaly. Only after extensive communication did we learn that a backend application behind the LB also depended on our database service, which was experiencing performance degradation caused by a recent change. Without tenant-side topology information, Dasheng cannot effectively diagnose such cases. We currently mitigate this limitation by manually collecting link topology and baseline traffic information for large customers and supplying it to Dasheng. However, this approach does not scale to all tenants. We plan to introduce a tenant-topology association stage in Dasheng that, based on the reliability task description, determines whether tenant topology is required and proactively queries the user, thereby reducing multi-turn interaction overhead. Furthermore, tenant applications may depend on services hosted by other cloud providers, requiring cross-vendor collaboration to complete reliability tasks.

Preventing Repeated Tool-Selection Loops. In some cases, we find that Dasheng can enter a dead loop during progressive tool selection. After exploring multiple infeasible tool-invocation plans, the LLM may forget previously attempted plans and repeat them, leading to excessively long analysis times or even stalling. We mitigate this by recording the tool-invocation plans that Dasheng has already tried.

Proactive Collection of Missing Information. We find that initial queries to Dasheng often lack key information, preventing further diagnosis. Engineers who submit these queries initially often possess only fragments received from tenants. As troubleshooting proceeds, Dasheng progressively requests more information. To supply it, engineers must interact with tenants again, and this back-and-forth greatly extends resolution time. To address this, we add memory for supplemental information in Dasheng and proactively ask for potentially needed information upon receiving a query, thereby reducing analysis time.

Long-Horizon Risk Warnings. In practice, we find that Dasheng often provides long-term risk warnings that engineers had not considered. In manual troubleshooting, engineers tend to resolve the immediate issue with a direct solution, overlooking potential future problems that the solution may introduce. For example, for performance drops caused by rate limiting, a common suggestion is to raise the rate-limit threshold. Dasheng instead warns of the risks of this action and offers alternatives, such as scaling out when traffic exceeds a certain threshold.

Automated Decision-Making in Dasheng. The procedure of Dasheng requires numerous human-machine interactions, such as whether to invoke a particular tool and what specific inputs should be provided. Dasheng may occasionally make errors that a human operator would avoid, for instance, supplying clearly garbled or malformed data as input to a tool. Hence, a secondary verification by engineers is required. Considering the potential risks associated with tool usage, we categorize tools into analytical and operational classes. The former includes tasks such as topology queries

and anomaly analysis, while the latter performs actions that modify the network state, such as initiating active probes or applying configuration changes. To reduce interaction overhead, we currently require engineer confirmation only for executing operational tools. In the future, we will develop more intelligent decision policies. For example, we can design a policy that determines whether human involvement is required based on historical experience.

Attention Deficiency in Dasheng. Dasheng needs to concatenate information as prompts for the LLM. In practice, we observe that generated prompts are often excessively long, leading to attention degradation in the LLM, especially during the result analysis phase. In such cases, the LLM may fail to process the content meaningfully and directly output the raw results from the tools. Therefore, before being processed by the LLM, we first prune tool outputs to remove redundant information.

8 Related Work

Deterministic Verification and Discriminative AIOps. Traditional reliability engineering primarily relies on white-box verification tools or AIOps based on black-box anomaly detection [10, 16, 17, 22, 27, 32, 33, 38, 41, 52, 54]. White-box verification depends on deep understanding and precise modeling of network forwarding structures, which limits its applicability. On the other hand, discriminative AIOps methods leveraging machine learning offer broader applicability. However, in complex cloud overlay networks, their generalization ability is often severely limited when facing unknown issues caused by new architectures or code updates. For example, Rex [32] learns change rules from historical data via machine learning, yet such rules can hardly cover failure scenarios unprecedented in history. Murphy [23] diagnoses faults by reasoning over relationships between entities, while SkyNet [51] filters low-severity alerts during alert flooding. Both could serve as reliability tools orchestrated by Dasheng, yet neither addresses the global, system-level reliability analysis across coupled layers that Dasheng targets.

LLM Adaptation in Reliability Engineering. To overcome the limitations of traditional AIOps, the industry has recently explored various LLM-driven reliability frameworks [45]. These can be broadly categorized into three paradigms.

(1) signal analysis assistants based on context engineering. Google's Gemini Cloud Assist [20] and Azure SRE Agent [36] both employ context engineering to collect massive logs, metrics, and tool outputs for anomaly identification. However, these methods typically aggregate massive raw anomaly items and feed them directly into the LLM. They lack the ability to distill these surface anomalies into underlying root causes. In cloud overlay networks, a single fault often triggers massive heterogeneous anomaly items across multiple layers. Such direct accumulation easily overloads the LLM and causes it to lose analytical focus.

(2) task decomposition and multi-agent frameworks based on SOPs. Meta's *Confucius* [48] decomposes network management tasks based on predefined SOPs and improves the ability to handle complex problems through multi-agent collaboration. However, this approach heavily relies on predefined SOPs. When facing unknown code bugs or non-standard architectures in cloud overlay networks, these rigid predefined orchestrations easily fail and cannot achieve dynamic fault recovery.

(3) conversational diagnostic systems. ByteDance's *VeAIOps* [11] and *NetAssistant* [47] lower the diagnostic barrier through natural language interaction. However, pure text-based interaction loses the structured context of network topologies. Therefore, they struggle to handle complex problems in cloud overlay networks that require cross-layer state correlation.

9 Conclusion

In this paper, we present Dasheng, an LLM-powered reliability engineering assistant tailored to cloud overlay networks. By extending the standard LLM-powered assistant paradigm, Dasheng integrates five core modules: profile-based user intent analysis, trie-tree-based initial instance context retrieval, dynamic instance context management, state-machine-driven progressive tool selection, and expert-experience-based result analysis. To date, Dasheng has integrated 139 reliability tools from diverse teams. Deployed on a large-scale production cloud for two years, Dasheng has served 1,450 engineers, processed approximately 70k reliability tasks and demonstrated robust performance.

10 Acknowledgments

We thank our shepherd and ATC reviewers for their thoughtful feedback. This work is supported in part by the National Key Research and Development Program of China under Grant 2025YFF0514602, and by the National Natural Science Foundation of China under Grants 62293611 and 62473260. This work is supported by Fong Shu Fook Tong Research Innovation Program of Zhejiang University Education Foundation, and also by Alibaba Group through Alibaba Innovative Research Program.

References

[1] Alibaba. 2022. Service Outage in Zone C of the China (Hong Kong) Region. https://www.alibabacloud.com/en/notice/resolved_service_outage_in_zone_c_of_the_china_hong_kong_region_1ae?_p_lc=1.

[2] Alibaba. 2023. Elastic Compute Service (ECS) Service Level Agreement. https://www.alibabacloud.com/help/en/legal/latest/elastic-compute-service-service-level-agreement.

[3] Alibaba. 2025. qwen-plus. https://modelstudio.console.alibabacloud.com/ap-southeast-1/?spm=a2ty_o05.31384571.0.0.5e7d9f6bnMARKc&tab=doc#/doc/?type=model&url=2840914_2&modelId=qwen-plus.

[4] Alibaba. 2025. qwen-turbo. https://modelstudio.console.alibabacloud.com/ap-southeast-1/?spm=a2ty_o05.31384571.0.0.5e7d9f6bnMARKc&tab=doc#/doc/?type=model&url=2840914_2&modelId=qwen-turbo.

[5] Alibaba. 2026. Alibaba OSS. https://www.aliyun.com/product/oss.

[6] Alibaba. 2026. Qwen3-30B-A3B. https://huggingface.co/Qwen/Qwen3-30B-A3B-Thinking-2507.

[7] Ahmed Alquraan, Hatem Takruri, Mohammed Alfatafta, and Samer Al-Kiswany. 2018. An Analysis of Network-Partitioning Failures in Cloud Systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 51–68. https://www.usenix.org/conference/osdi18/presentation/alquraan

[8] AWS. 2025. AWS DevOps Agent. https://docs.aws.amazon.com/devopsagent/latest/userguide/about-aws-devops-agent.html.

[9] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2020. Decaf: Diagnosing and triaging performance issues in large-scale cloud services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM, Seoul, South Korea, 201–210.

[10] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, New York, NY, USA, 155–168.

[11] ByteDance. 2025. VeAIOps. https://volcengine.github.io/veaiops/introduction/overview.

[12] Qiong Chen, Jianmin Qian, Yulin Che, Ziqi Lin, Jianfeng Wang, Jie Zhou, Licheng Song, Yi Liang, Jie Wu, Wei Zheng, et al. 2024. Yuanrong: A production general-purpose serverless system for distributed applications in the cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference*. Association for Computing Machinery, Sydney, NSW, Australia, 843–859.

[13] Xin Chen, Charng-Da Lu, and Karthik Pattabiraman. 2014. Failure analysis of jobs in compute clouds: A google cluster case study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, Naples, Italy, 167–177.

[14] 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*. Association for Computing Machinery, Athens, Greece, 674–688.

[15] Cloudflare. 2025. Cloudflare outage on November 18, 2025. https://blog.cloudflare.com/18-november-2025-outage.

[16] Chongrong Fang, Haoyu Liu, Mao Miao, Jie Ye, Lei Wang, Wansheng Zhang, Daxiang Kang, Biao Lyv, Peng Cheng, and Jiming Chen. 2020. Vtrace: Automatic diagnostic system for persistent packet loss in cloud-scale overlay network. 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*. ACM, Virtual Event, USA, 31–43.

[17] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, USA, 469–483.

[18] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*. Association for Computing Machinery, San Francisco, CA, USA, 126–141.

[19] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference*. ACM, Toronto, ON, Canada, 350–361.

[20] Google. 2025. Gemini Cloud Assist. https://cloud.google.com/blog/products/management-tools/gemini-cloud-assist-investigations-performs-root-cause-analysis.

[21] Albert Greenberg, Gisli Hjalmtysson, David A Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. 2005. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review* 35, 5 (2005), 41–54.

[22] 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)*. USENIX Association, Seattle, WA, USA, 71–85.

[23] 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.

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

[25] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, et al. 2014. Network virtualization in multi-tenant datacenters. In *11th USENIX symposium on networked systems design and implementation (NSDI 14)*. USENIX Association, Seattle, WA, USA, 203–216.

[26] Haonan Li, Tian Pan, Jin Ke, Baohai Hu, Changgang Zheng, Enge Song, Zhi Xu, Ye Yang, Bowen Yang, Donglin Lai, et al. 2026. {CStar} Gateway: Augmenting Public Cloud Infrastructure for Heterogeneous Network Function Virtualization. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. 1501–1515.

[27] Yilei Lin, Ting He, Shiqiang Wang, Kevin Chan, and Stephen Pasteris. 2020. Looking glass of NFV: Inferring the structure and state of NFV network from external observations. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1477–1490.

[28] Kefei Liu, Jiao Zhang, Zhuo Jiang, Xuan Zhang, Shixian Guo, Yangyang Bai, Yongbin Dong, Zhang Zhang, Xiang Shi, Lei Wang, et al. 2024. Hostmesh: Monitor and Diagnose Networks in Rail-optimized RoCE Clusters. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*. ACM, Sydney, Australia, 122–128.

[29] Jianyuan Lu, Shunmin Zhu, Jun Liang, Yuxiang Lin, Tian Pan, Yisong Qiao, Yang Song, Wenqiang Su, Yixin Xie, Yanqiang Li, et al. 2025. Albatross: A Containerized Cloud Gateway Platform with FPGA-accelerated Packet-level Load Balancing. In *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, Coimbra, Portugal, 71–84.

[30] Biao Lyu, Enge Song, Tian Pan, Jianyuan Lu, Shize Zhang, Xiaoqing Sun, Lei Gao, Chenxiao Wang, Han Xiao, Yong Pan, et al. 2024. POSEIDON: A Consolidated Virtual Network Controller that Manages Millions of Tenants via Config Tree. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, USA, 1083–1099.

[31] Sajee Mathew and J Varia. 2014. Overview of amazon web services. *Amazon Whitepapers* 105, 1 (2014), 22.

[32] 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.

[33] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI '19)*. ijcai.org, Macao, China, 4739–4745. doi:10.24963/ijcai.2019/658

[34] Microsoft. 2025. Azure status history. https://azure.status.microsoft/en-us/status/history/?trackingId=YKYN-BWZ.

[35] Microsoft. 2025. Microsoft SRE agent. https://learn.microsoft.com/en-us/azure/sre-agent/.

[36] Microsoft. 2026. Azure SRE Agent. https://azure.microsoft.com/en-us/products/sre-agent.

[37] Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. 2011. Netlord: a scalable multi-tenant network architecture for virtualized datacenters. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 62–73.

[38] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. E2: A framework for NFV applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, Monterey, CA, USA, 121–136.

[39] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, et al. 2021. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. ACM, Virtual Event, USA, 194–206.

[40] 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)*. USENIX Association, Boston, MA, USA, 595–612.

[41] 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*. ACM, New York, NY, USA, 420–437.

[42] Xiaoqing Sun, Xionglie Wei, Xing Li, Ju Zhang, Bowen Yang, Yi Wang, Ye Yang, Yu Qi, Le Yu, Chenhao Jia, et al. 2025. ZooRoute: Enhancing Cloud-Scale Network Reliability via Overlay Proactive Rerouting. In *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, Coimbra, Portugal, 1251–1253.

[43] 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)*. USENIX Association, Boston, MA, USA, 599–614.

[44] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Joaquin Vanschoren and Sai-Kit Yeung (Eds.). Curran Associates, Inc., Red Hook, NY, USA, 1–10. https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html

[45] Changjie Wang, Mariano Scazzariello, Alireza Farshin, Simone Ferlin, Dejan Kostić, and Marco Chiesa. 2024. Netconfeval: Can llms facilitate network configuration? *Proceedings of the ACM on Networking* 2, CoNEXT2 (2024), 1–25.

[46] Chenxu Wang, Xumiao Zhang, Runwei Lu, Xianshang Lin, Xuan Zeng, Xinlei Zhang, Zhe An, Gongwei Wu, Jiaqi Gao, Chen Tian, et al. 2025. Towards LLM-Based Failure Localization in Production-Scale Networks. In *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, Coimbra, Portugal, 496–511.

[47] 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)*. USENIX Association, Santa Clara, CA, USA, 2011–2024.

[48] Zhaodong Wang, Samuel Lin, Guanqing Yan, Soudeh Ghorbani, Minlan Yu, Jiawei Zhou, Nathan Hu, Lopa Baruah, Sam Peters, Srikanth Kamath, et al. 2025. Intent-driven network management with multi-agent llms: The confucius framework. In *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, Coimbra, Portugal, 347–362.

[49] Tingting Xu, Bengbeng Xue, Yang Song, Xiaomin Wu, Xiaoxin Peng, Yilong Lyu, Xiaoliang Wang, Chen Tian, Baoliu Ye, Camtu Nguyen, et al. 2024. CyberStar: Simple, Elastic and Cost-Effective Network Functions Management in Cloud Network at Scale. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, USA, 227–246.

[50] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[51] Bo Yang, Huanwu Hu, Yifan Li, Yunguang Li, Xiangyu Tang, Bingchuan Tian, Gongwei Wu, Jianfeng Xu, Xumiao Zhang, Feng Chen, et al. 2025. SkyNet: Analyzing Alert Flooding from Severe Network Failures in Large Cloud Infrastructures. In *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, Coimbra, Portugal, 512–526.

[52] Shunmin Zhu, Jianyuan Lu, Biao Lyu, Tian Pan, Chenhao Jia, Xin Cheng, Daxiang Kang, Yilong Lv, Fukun Yang, Xiaobo Xue, et al. 2022. Zoonet: a proactive telemetry system for large-scale cloud networks. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. ACM, Roma, Italy, 321–336.

[53] Shunmin Zhu, Jianyuan Lu, Biao Lyu, Tian Pan, Shize Zhang, Xiaoqing Sun, Chenhao Jia, Xin Cheng, Daxiang Kang, Yilong Lv, et al. 2024. Proactive Telemetry in Large-Scale Multi-Tenant Cloud Overlay Networks. *IEEE/ACM Transactions on Networking* 32, 4 (2024), 3002–3017.

[54] 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*. ACM, London, United Kingdom, 479–491.

Received 10 June 2026; revised 12 March 2009; accepted 5 June 2009