Aloha: Localizing Batch Failures in Large-scale Cloud Systems via Contrast Analysis and Human-in-the-Loop Agent

Shenglin Zhang
Nankai University
Tianjin, China

Yujia Wu
Nankai University
Tianjin, China

Jinghuan Ren
Nankai University
Tianjin, China

Yongqian Sun*
Wenwei Gu
Nankai University
Tianjin, China

Chaoyun Zhang
Microsoft
Beijing, China

Liqun Li
Qingwei Lin
Microsoft
Beijing, China

Dongmei Zhang
Microsoft
Beijing, China

Saravan Rajmohan
Chetan Bansal
Microsoft
Redmond, USA

Minghua Ma
Microsoft
Redmond, USA

Abstract

Large-scale cloud systems underpin modern computing, hosting diverse components to deliver critical services worldwide. A single fault—such as an outage or misconfiguration—can simultaneously disrupt thousands of users. Such large-scale faults, referred to as batch failures, are characterized by many affected instances across the same subject within a short time window, typically stemming from a shared root cause. Handling these failures efficiently requires anomaly localization, but existing approaches offer insufficient support to engineers, making the process time-consuming and cognitively demanding. To address this, we propose **Aloha**, a human-in-the-loop agent framework for anomaly localization based on contrast analysis. Aloha operationalizes the entire batch failure handling pipeline, providing scenario- and data-aware guidance along with interpretable root-cause patterns for engineers. Pilots on real-world batch failure cases in Microsoft's cloud show that Aloha streamlines data handling, supports contrast-based anomaly localization, and makes the process more practical and accessible, offering a promising step toward human-centered, scalable failure management in large-scale cloud systems.

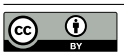
CCS Concepts

• **Software and its engineering → Software maintenance tools**.

Keywords

Cloud Service Monitoring, Software Reliability, Anomaly Localization, Large Language Models

* Yongqian Sun is the corresponding author.



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3805241

ACM Reference Format:
Shenglin Zhang, Yujia Wu, Jinghuan Ren, Yongqian Sun*, Wenwei Gu, Chaoyun Zhang, Liqun Li, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, Chetan Bansal, and Minghua Ma. 2026. Aloha: Localizing Batch Failures in Large-scale Cloud Systems via Contrast Analysis and Human-in-the-Loop Agent. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3803437.3805241

1 Introduction

Large-scale cloud systems have become the backbone of modern computing, hosting diverse hardware and software components to deliver critical services worldwide. With millions of users depending on these infrastructures, a single fault can potentially affect thousands of customers simultaneously. For example, an infrastructure outage or configuration error in cloud systems may cause widespread service disruption [5, 26, 39].

We refer to such large-scale faults as batch failures [20], which are characterized by three properties: (1) a large number of failure instances, (2) failures occurring across different instances of the same subject, and (3) occurrences within a relatively short time window. The consequences of batch failures in cloud systems are severe, as they often escalate into service incidents that cause large-scale disruption and performance degradation. Figure 1 illustrates a representative case of this phenomenon.

As shown in the figure, a defining feature of batch failures is that their propagation usually originates from a common root cause, such as a misconfiguration or an infrastructure fault in cloud systems. [20, 52] Consequently, effective handling hinges on the ability to localize anomalies, i.e., to narrow down failure instances to the responsible components, configurations, or execution contexts.

In practice, recent work such as CONAN [20] demonstrates the promise of contrast-based diagnosis for anomaly localization in batch failures. However, when an engineer attempts to adopt such approaches in real troubleshooting workflows, several practical barriers emerge. We distill these barriers into three key challenges.

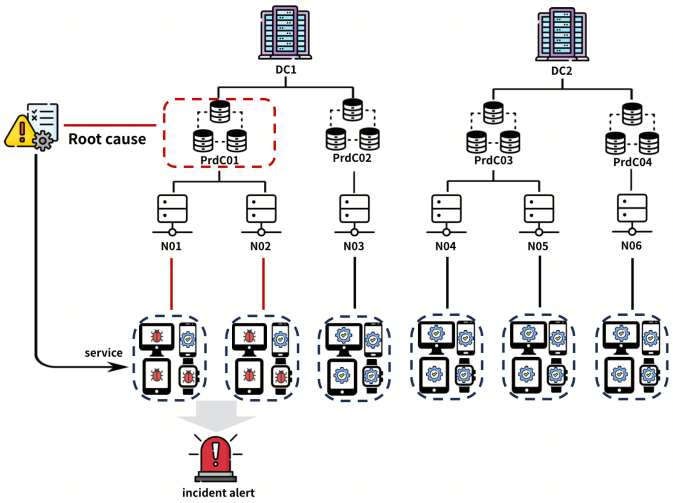


Figure 1: Example of a batch failure in a cloud system. Two datacenters host multiple clusters (PrdC01–PrdC04), each containing several nodes that serve end users. A failure in PrdC01 propagates to its subordinate nodes (Node01–Node02), leading to large-scale service disruption and an aggregated incident alert—a typical example of how batch failures emerge in cloud environments.

Challenge 1: Scenario Ambiguity.

When an engineer encounters an incident alert in a cloud system, the first task is to determine whether the observed failures constitute a batch case and whether the available data satisfies the conditions required for contrast-based anomaly localization. Raw data in cloud systems often comes in heterogeneous formats and with diverse semantics (e.g., logs [29, 54], traces [45, 46], metrics [22, 42], structured reports [4, 23, 31]), making this judgment non-trivial.

This difficulty is compounded by limitations in existing research. Many studies focus on algorithms for specific scenarios [31, 33, 35]—for instance, Castelluccio et al. [4] proposed grouping crash reports to identify common root causes. Sun et al. [40] targets online video service metrics to localize anomalous multi-dimensional derived measures. CONAN [20] generalizes the problem by unifying batch failure diagnosis across diverse scenarios through contrast pattern extraction. While this generality increases methodological applicability, it also imposes a substantial cognitive burden on engineers, who must interpret scenario definitions and map raw data to the framework before any meaningful anomaly localization can proceed.

Challenge 2: Unreliable Data Quality.

For batch failure cases, contrast-analysis–based anomaly localization algorithms require representing failure instances and their features in a structured, table-like format in order to perform root-cause search. When engineers handle tabularized data derived from raw incident context, even small inconsistencies—such as weak label contrast, mis-specified attributes, or hidden missing values—can hinder anomaly localization, making the diagnosis unreliable or, in some cases, infeasible.

However, existing methods largely assume that the input data is already well-prepared and meaningful [20, 40, 52], offering little guidance for engineers to handle real-world inconsistencies. As a result, engineers often face a significant cognitive burden, needing to validate, clean, and interpret the data before anomaly localization can proceed.

Challenge 3: Maladapted Strategies.

Batch failure cases with well-prepared tabularized data still require careful consideration of anomaly localization strategies. Engineers need to determine the appropriate objective function to guide the root-cause search, which evaluates how well different candidate explanations account for the observed failures, and configure algorithm parameters—such as iteration limits, pattern sizes, and search scopes. Unsuitable parameters and configuration settings for the anomaly localization algorithm can lead to inefficient searches and suboptimal root cause identification.

While existing methods largely focus on designing algorithms for contrast-based localization, with objective functions and search strategies tailored to specific scenarios [22, 40, 42, 52], they provide little guidance on parameter tuning or strategy selection, leaving engineers to manually explore configurations. This not only slows down the diagnostic process but also increases the likelihood of errors or incomplete localization.

Our Solution. To tackle these challenges, we propose **Aloha**, an anomaly localization framework based on contrast analysis and human-in-the-loop agent for handling batch failures, unifying the end-to-end diagnostic workflow and guiding engineers toward actionable anomaly localization. It integrates three phases: **scenario check** distinguishes batch failures from isolated cases using the Criterion-driven Applicability Protocol; **data validation** organizes and verifies failure data with the Semantic-and-Executable Validation Toolkit; and **anomaly localization** identifies root causes through pattern search guided by RAG-based Strategy Selection. Together, these phases enable efficient, accurate, and user-friendly handling of batch failure cases, with human-in-the-loop support reducing engineer cognitive load and guiding the engineer through recognition and anomaly localization.

Contributions. The contributions of our work are summarized as follows:

- **Originality:** To the best of our knowledge, this is the first work to identify and address the practical usability gap of contrast-analysis–based anomaly localization for batch failures. Our agent-assisted workflow reduces the cognitive burden on engineers and makes recognition and localization of batch failures practically achievable.
- **Approach:** we proposed Aloha, a human-in-the-loop agent for anomaly localization based on contrast analysis. Aloha operationalizes the entire pipeline of batch failure handling, providing scenario- and data-aware guidance as well as interpretable root-cause patterns for engineers.
- **Practicality:** We piloted Aloha on 127 real-world batch failure cases in Microsoft cloud systems, demonstrating its utility and usability by successfully including the true root cause in the top-5 recommendations for 93.7% of cases and reducing engineers' time per case from 10 hours to 0.5 hours.

Table 1: Representative real-world batch failure scenarios

Incident Scenario	Data Source	Batch Scope	Contextual Attributes
API failures spike [20, 35, 52]	Request logs	Failed API requests clustered by shared attribute values within a short time window	API name, cluster, datacenter, node, software version
Software crash surge [4, 28, 31, 45]	Crash reports	Manifested as cluster group of multiple simultaneous crash instances	Software version, operating system, hardware configuration, stack trace, runtime
Service-level video stalls [21, 22, 40, 42]	KPI metrics	User sessions aggregated within a short time window that jointly contribute to the anomalous service-level metric	CDN provider, bitrate, device
Multi-server failure surge [12, 29, 54]	Console logs	A batch of anomalous servers exhibiting similar failure symptoms within the same incident time window	Server ID, node ID, software version, configuration files, runtime environment
Virtual machine(VM) crashes due to virtual hard disks(VHD) access loss [20, 50]	VM, storage account, and network topology	VMs sharing remote VHD dependencies that fail within a short time window across clusters	VM ID, compute cluster, storage account, VHD, network path, hypervisor

2 Practical Batch Failure Diagnosis

Batch failures are a common yet challenging class of incidents in large-scale production systems. To design an anomaly localization framework for batch failures that is truly usable in practice, it is essential to first understand what batch failures look like in real production, how engineers currently diagnose them, and what capabilities an effective framework should provide.

In this section, grounded in our experience with real production incidents, we empirically analyze practical batch failure diagnosis by addressing the following questions:

Q1. **What characteristics do batch failures show in practice?**
Q2. **How are batch failures handled by engineers in practice?**
Q3. **What capabilities are required for a practical batch failure diagnosis framework?**

2.1 Real-world Scenarios of Batch Failures

Batch failure occurs when multiple instances of the same subject experience failures within a short time window, forming a set of related anomalies. Unlike single-point failures, which affect only one instance, or complete service outages, which impact all users, batch failures typically involve a portion of instances that share an underlying issue. This pattern naturally suggests that analyzing the failures in relation to one another–grouping similar instances and comparing different groups–can provide valuable insight into the problem.

Given these characteristics, the nature of batch failures becomes more tangible when considering concrete examples from production systems, as summarized in Table 1. The table presents representative scenarios and highlights the associated data sources, batch scopes–that is, how a batch of failed instances manifests in each scenario–and the contextual attributes associated with each failure instance, which describe the conditions under which it occurred.

Despite the diversity of these scenarios, they all exhibit the properties introduced in the Introduction: (1) a large number of failure instances, (2) failures occurring across multiple instances of the same subject or system component, and (3) occurrences within a relatively short time window. Observing these properties helps clarify the scope of the problem by indicating whether a given scenario can be considered a batch failure, providing a concrete basis for defining the problem before any analysis or method is applied.

2.2 Practical Batch Failure Diagnosis Process

Recent research demonstrates the promise of contrast-based diagnosis for localizing anomalies in batch failures. At Microsoft, CONAN [20] implements this approach as a flexible system that can be applied across diverse scenarios, offering a general solution compared to prior scenario-specific methods [4, 40, 52, 54]. The system has been deployed as both a Python library and a cloud-native application in Azure. Despite this accessible and scalable implementation, we found that engineers often exhibit low willingness to adopt the tool in operational practice.

Before analyzing the factors that hinder engineers in practice, it is important to review the foundational concepts of contrast-based diagnosis for batch failures. These include the principle of contrast analysis in the context of batch failures, as well as notions of pattern search and scoring that underpin the anomaly localization process. The following sections provide a closer look at these concepts, starting with contrast analysis and then pattern search.

2.2.1 Contrast Analysis. Contrast analysis is a method for identifying meaningful differences between groups, studied in both statistics and data mining [2, 7, 11]. It generally assumes that the groups under comparison are well-defined and comparable, and that the observed differences reflect systematic effects rather than random noise. In practice, this often takes the form of a target group, which represents the focal condition of interest, contrasted against one or more background groups that provide the necessary baseline for meaningful comparison [20]. Such requirements are naturally met in batch failures, which involve a large number of instances from the same subject occurring within a specific time window, where the anomalous subset forms the target group and the remaining instances provide the background for comparison in contrast analysis.

In anomaly localization, contrast analysis helps identify differences that distinguish one group of interest from other contrasting groups. For instance, Castelluccio et al. [4] and Qian et al. [31] applied contrast set mining to crash reports, comparing one crash type against others to extract distinguishing features. Similarly, Ren et al. [33] captured relational contrasts between fine-grained runtime events to locate performance anomalies and their root causes.

In our framework, contrast analysis underlies the anomaly localization process, while all preceding checks of the scenario and data ensure that the input data satisfies its requirements—most importantly, the presence of clearly defined target and background groups—so that the algorithm can meaningfully identify potential causes of batch failures.

2.2.2 Pattern Search. Building on contrast analysis, which highlights meaningful differences between target and background groups, anomaly localization can be seen as the task of identifying which features or feature combinations exhibit these contrasts. This identification process is inherently a search: it seeks patterns that satisfy specific contrast-based conditions. Following Li et al. [20], contrast pattern extraction can thus be formulated as a search problem, where an objective function is defined and then maximized to guide the discovery of relevant anomalies.

In this search framework, a pattern corresponds to a combination of attribute-value pairs (AVPs) that collectively distinguish the target group from the background. In the context of batch failures, each attribute is chosen for its potential to help localize anomalies, such as environmental factors, system configurations, or runtime metrics, making the AVP combination a concrete representation of candidate causes for the observed failure. Using combinations of multiple features allows the search to capture interactions that single-feature tests cannot, providing interpretable and actionable insights for anomaly localization.

To formalize the discussion, we summarize the key concepts used throughout our framework:

- **Case**: A single batch failure scenario, encompassing a set of instances that together define the event.
- **Instance**: An individual entity within a batch failure. All instances in a case belong to the same subject, representing the same type of failure (e.g., API requests in an API failure batch or crash reports in a software crash batch). Instances can be part of the target group (exhibiting the failure) or part of background groups used for contrast, which may represent normal operation or other alternative failure types.
- **Attribute-Value Pair (AVP)**: A feature and its observed value for an instance. AVPs capture properties relevant for localizing anomalies, such as environmental conditions, system configurations, or runtime metrics. Multiple attributes may have hierarchical or relational constraints, for example forming chains like Node → Cluster → Datacenter, as observed in Figure 1.
- **Pattern**: A combination of AVPs identified through the search process that distinguishes the target group from the background. In anomaly localization, patterns correspond to candidate root causes of the batch failure.
- **Objective Function**: A function defined over candidate patterns that quantifies their degree of contrast between the target and background groups. The search process aims to maximize this function to identify the most relevant patterns.

Building on the concepts above, CONAN [20] implements the anomaly localization process through a meta-heuristic search that iteratively modifies candidate patterns by adding or removing AVPs and evaluates them with the objective function. The search maintains a fixed-size list of the best-scoring patterns, guiding the discovery of candidate root causes in batch failure scenarios.

Despite this well-defined process, operational use of CONAN [20] by engineers is far from straightforward. To better understand the practical challenges, we collected feedback from engineers across multiple batch failure cases. These discussions revealed that manually handling a case–inspecting the scenario and preparing data–along with applying contrast-based anomaly localization algorithms, including selecting or writing the objective function, configuring parameters, and consulting prior incident records, typically required about one week and involved numerous steps that consumed substantial cognitive effort. Such findings highlight that, although existing methods provide a systematic approach, they impose a substantial usability gap for engineers in real-world settings.

2.3 Requirements for Practical Batch Failure Diagnosis

The analysis above reveals that the primary challenge in practical batch failure diagnosis does not lie in the lack of effective algorithms, but in the substantial usability gap between formal diagnosis frameworks and engineers' operational workflows. To bridge this gap, a practical solution is expected to support engineers throughout the end-to-end reasoning and execution process.

Batch failure diagnosis is characterized by high scenario diversity, heterogeneous data formats, and flexible problem formulations. As a result, many key decisions—such as determining whether an incident constitutes a batch failure, mapping raw operational data to contrast-based representations, and selecting or adapting scoring functions—require contextual understanding and cannot be fully automated in a reliable manner.

This motivates the design of an assistant framework that combines a degree of autonomous reasoning with a human-in-the-loop interaction model. Rather than replacing engineers or eliminating human judgment, the goal is to support engineers by surfacing interpretations, offering structured guidance, and enabling validation at critical steps, without introducing additional operational burden. Such a balance is essential in production environments, where diagnosis errors are costly and domain expertise remains indispensable.

Against this background, a practical batch failure diagnosis framework should provide the following capabilities:

Requirement understanding. The framework should be able to interpret engineers' high-level diagnostic intent, including the incident description, available data sources, and operational constraints. Rather than requiring users to manually translate a case into formal inputs, the system should assist in identifying the subject, batch scope, and candidate contextual attributes relevant to diagnosis.

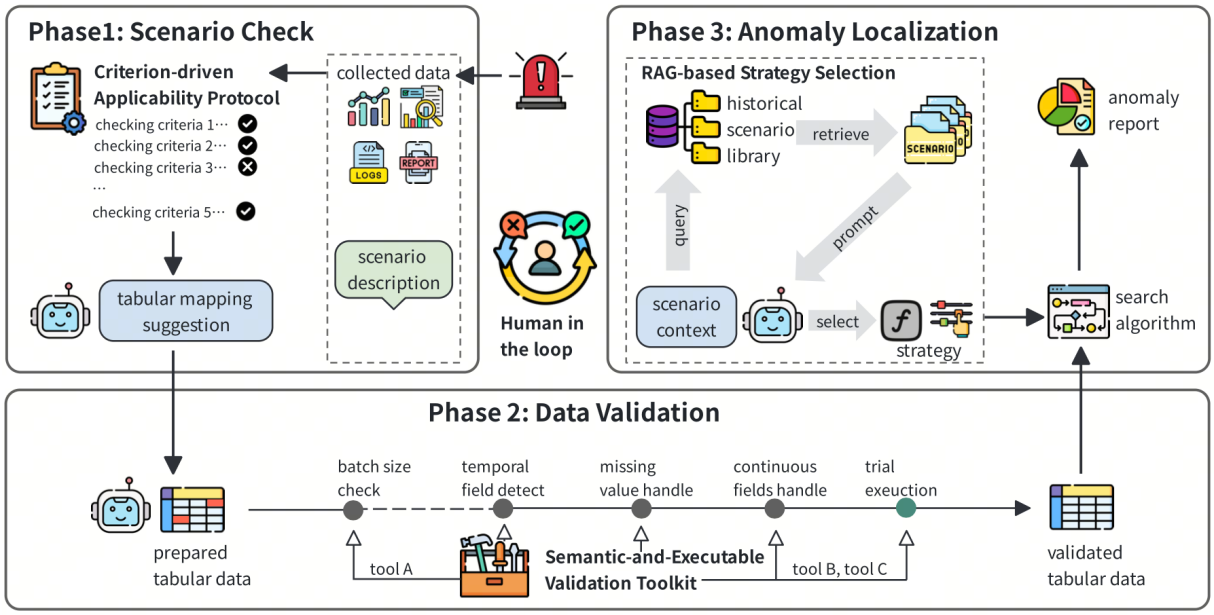


Figure 2: The overall workflow of Aloha facing an incoming incident

Experience grounding and contextual memory. Effective diagnosis often relies on prior experience, including historical incidents, known failure patterns, and previously successful configurations. The framework should maintain and leverage such contextual knowledge to summarize the current case, draw parallels to similar scenarios, and reuse validated diagnosis strategies where applicable.

Reasoning and method mapping. Given a concrete operational case, the framework should be able to reason about how it aligns with contrast-based batch failure diagnosis principles. This includes mapping real-world data to formal concepts such as instances, attributes, target and background sets, as well as identifying suitable pattern scoring formulations. Crucially, this reasoning step reduces the cognitive burden on engineers by mediating between informal problem descriptions and formal diagnostic abstractions.

Executable assistance. Beyond reasoning, the framework should provide actionable support, including data preprocessing, transformation, and validation, as well as invoking diagnosis procedures with appropriate configurations. By offering executable tools rather than conceptual guidance alone, the framework helps engineers progress from problem understanding to concrete diagnostic results with reduced manual effort.

Taken together, these capabilities outline the requirements of a practical batch failure diagnosis framework that directly targets the usability challenges identified earlier. By combining autonomous reasoning with guided human interaction, such a framework can make contrast-based diagnosis methods accessible and effective for real-world engineering practice.

3 Aloha

In this section, we will present the design of Aloha. Figure 2 shows the overview of Aloha, consisting of three phases: scenario check, data validation and anomaly localization. To address the challenges in each phase, we introduce three core mechanisms: the Criterion-driven Applicability Protocol, the Semantic-and-Executable Validation Toolkit, and the RAG-based Strategy Selection, complemented by a cross-cutting Human-in-the-loop Integration.

3.1 Criterion-driven Applicability Protocol

The Scenario Check phase is designed to assess whether a given failure scenario is eligible for batch diagnosis and to provide intermediate guidance for subsequent analysis. In this phase, Aloha takes as input heterogeneous raw data—such as JSON reports, metrics, or statistical tables—along with a brief description from the engineer. The outcome is twofold: a judgment on the applicability of batch diagnosis and a tabular mapping suggestion that indicates how the raw data could be organized into instances and attributes for downstream analysis. This process, however, is far from straightforward. Failure scenarios arise in diverse forms, ranging from request failures in cloud services to large-scale software crashes, service-level performance degradations, or complex virtualized infrastructure faults [50]. Across these varied scenarios, engineers often face ambiguity: it is unclear whether a scenario should be treated as a batch case, and even when batch diagnosis is assumed, the collected data may still fall short of the requirements for contrast-based anomaly localization, e.g., due to missing attributes, irrelevant information, or insufficient coverage of target and background instances. To address this challenge, we propose the Criterion-driven Applicability Protocol, which formalizes explicit criteria to guide eligibility judgments [8, 44].

To inform the design of protocal, we conducted an in-depth study of how human engineers recognize and handle batch failures in practice. Specifically, we analyzed historical batch failure cases using a modified Fault Tree Analysis (FTA) approach [3, 9], which decomposes the requirements for forming a batch failure case, to

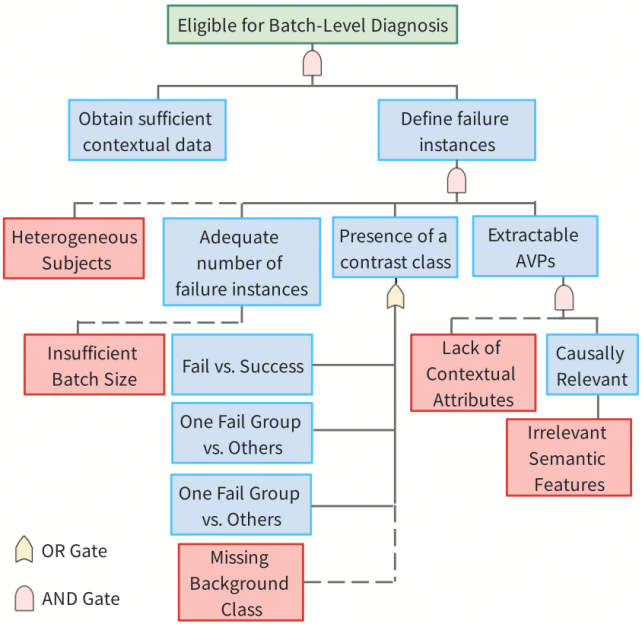


Figure 3: A modification of fault tree analysis (FTA) of batch failure diagnosis eligibility. Green nodes indicate the root decision, blue nodes represent conditions that must be satisfied for eligibility, red nodes mark situations where a condition is unmet, in which case the batch failure cannot be concluded. AND and OR gates depict how conditions combine to determine the overall outcome.

identify the fundamental conditions under which batch failure diagnosis is applicable.

Figure 3 shows the resulting tree structure. Through this analysis, we identified five recurring risk factors that frequently lead to incorrect judgments: heterogeneous subjects, insufficient batch size, absence of a background class, lack of contextual attributes, and missing semantically relevant features. These factors directly informed the definition of criteria C1–C5, each capturing a concrete aspect of scenario validity, including subject consistency, batch sufficiency, availability of a background class for contrast, and the presence of contextual and semantically meaningful attributes for diagnosis.

In operation, Aloha first generates a lightweight summary of the raw inputs—capturing file structures and sample contents—which is then evaluated by Aloha against criteria C1–C5. The engineer can confirm or adjust these judgments, after which the protocol issues an eligibility decision and, when applicable, a tabular mapping suggestion that outlines how the scenario could be structured into instances, attributes, and labels for downstream analysis. This process resolves ambiguity in deciding whether batch diagnosis is applicable and provides a consistent starting point for subsequent data preparation and anomaly localization.

3.2 Semantic-and-Executable Validation Toolkit

After applicability is confirmed, the next phase is to ensure that the prepared tabular data is "ready" for anomaly localization: not just structurally valid, but also semantically meaningful and executable. As highlighted in Challenge 2 on *Unreliable Data Quality*, this process is often undermined by subtle but frequent data-quality issues: weak label contrast, incomplete or mis-specified attributes, unhandled temporal fields, or hidden missing values. A common workaround is for engineers to manually patch the data with ad-hoc scripts, which is time-consuming and still risks overlooking critical details.

To tackle this challenge, we introduce the Semantic-and-Executable Validation Toolkit, which supports the agent with a thorough yet convenient data validation backbone. Inspired by recent advances in code-augmented agents and LLMs with tool-execution capabilities [14, 32, 36], the toolkit enables validation proceeds as a coherent flow where frequent but easily overlooked pitfalls are checked and repaired, as illustrated in Figure 4.

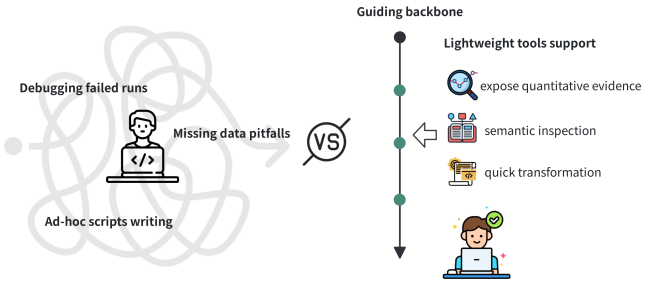


Figure 4: Comparison of tabular data validation. Manual preparation (left) is fragmented and error-prone, while the Semantic-and-Executable Validation Toolkit (right) equips the agent with lightweight tools that guide the process in a coherent and executable flow.

The toolkit equips the agent with lightweight yet executable tools: some expose quantitative evidence to ground judgments in real data values, others enable semantic inspection of individual fields, and still others offer quick transformations for tricky cases such as discretizing continuous values or restructuring hierarchies. These tools are implemented as callable functions that the agent can invoke during reasoning, allowing data validation to shift from passive checking to active problem-solving. In this way, the data validation phase proceeds as a guided process—thorough enough to prevent overlooked details, while remaining lightweight and convenient for practical use.

Overall, in this phase, the toolkit guides the tabular input through stepwise validation that surfaces pitfalls with engineer confirmation, and further supports a lightweight trial run—extracting AVPs, selecting a default objective, and executing a quick dry-run search—to ensure the data can be seamlessly consumed by the diagnosis algorithm. In this way, the validation phase shifts from ad-hoc manual checking to a dependable backbone that secures both data quality and algorithmic executability.

3.3 RAG-based Strategy Selection

Building on the validated tabular data from the previous phase, the final phase focuses on identifying root causes through attribute pattern discovery and generating a diagnostic report. As highlighted

in Challenge 3 on *Maladapted Strategies*, configuring the anomaly localization algorithm—choosing appropriate objective function and settings—can be time-consuming and cognitively demanding, potentially leading to inefficient searches or suboptimal root cause identification. To address this, we propose a *RAG-based Strategy Selection* mechanism, which leverages historical scenario knowledge to guide adaptive strategy and parameter choice. The vector indexing and retrieval follows the standard RAG paradigm [19], providing a robust foundation for scenario-aware decision-making [10, 38].

Specifically, we construct a scenario library $\mathcal{L} = \{s_1, s_2, \ldots, s_N\}$, where each scenario s_i records past failure cases sharing the same characteristics: scenario description, instance definitions, relevant attributes, chosen objectives, parameter settings, and reasoning behind these choices. s_i is embedded into a vector $\mathbf{v}_i \in \mathbb{R}^d$ encoding textual and structured information. For a new case, Aloha forms a query vector $\mathbf{q} \in \mathbb{R}^d$ by consolidating the scenario understanding and data characterization obtained in the previous phases, and computes the cosine similarity with each historical case embedding:

$$\text{sim}(\mathbf{q}, \mathbf{v}_i) = \frac{\mathbf{q} \cdot \mathbf{v}_i}{\|\mathbf{q}\| \, \|\mathbf{v}_i\|}.$$

The retrieved scenarios are ranked by their similarity scores and assigned graded relevance levels: highly similar scenarios provide direct guidance, while more loosely related scenarios serve as analogies that inspire partial adaptation.

On top of this retrieval, Aloha selects candidate strategies from the available objective function pool based on insights drawn from similar historical scenarios. Candidate strategies are briefly tested with a reduced search budget on the current data to test and compare their feasibility and performance. Finally, the most suitable strategy is executed in full, enabling the anomaly localization algorithm to identify high-scoring attribute patterns and generate the diagnostic report for the engineer.

Through this mechanism, strategy selection is no longer an opaque or arbitrary choice, but a process informed by both scenario-specific attributes and reusable historical experience. The integration of semantic retrieval and graded relevance reduces the need for ad-hoc manual tuning, mitigates the risk of maladapted strategies, and ensures that anomaly localization proceeds with both efficiency and interpretability.

3.4 Human-in-the-loop Integration

While our framework automates the three phases of anomaly localization, human expertise remains essential for reliable and interpretable outcomes. To this end, we implement a *Human-in-the-loop* (HITL) mechanism, allowing engineers to confirm, refine, or override agent judgments at key decision points throughout the process [27, 34]. This continuous integration of human oversight ensures that automated decisions align with domain knowledge, improves trust in the system, and enables feedback-driven refinement of agent behavior. Detailed demonstrations of the HITL workflow are provided in Section 4.

4 Case Study

We deployed the proof-of-concept version of Aloha in Microsoft's cloud service system. It has been running for over three month. During this deployment, we encountered diverse scenarios that reflect practical challenges such as missing background groups or incomplete attributes. We also collected feedback from engineers on their experience using Aloha, reporting that the time spent per case decreased from ~10 hours to ~0.5 hours. Separately, we gathered 127 real-world batch failure cases to pilot Aloha and compare its performance with the manual process. The framework successfully included the true root cause in the top-5 recommendations for 93.7% of cases. Table 2 summarizes Aloha's root cause localization accuracy (ACC@k) compared with CONAN [20].

Table 2: Comparison of root cause localization accuracy

Method	ACC@1	ACC@3	ACC@5	ACC@10
CONAN [20]	0.4741	0.6519	0.6963	0.7185
Aloha	0.6693	0.8504	0.9370	0.9685

Among these, we select one representative case for in-depth analysis. This case originates from a large-scale online service in Microsoft 365, where client applications interact with the service via REST APIs, and incidents are triggered when batches of requests fail. During such an incident, a snapshot of contextual data is collected, typically containing tens of millions of request instances, each representing a succeeded or failed request with many associated attributes, as shown in Figure 5.

```
2025-09-10T10:01:05.123Z
     logID=550e8400-e29b-41d4-a716-446655440110
     datacenter=DC1 cluster=PrdC01 node=N01
     api=GET-FILES apiversion=V3 latency=842ms
     status=Failure
2025-09-10T10:01:05.457Z
     logID=550e8400-e29b-41d4-a716-446655440111
     datacenter=DC1 cluster=PrdC01 node=N02
     api=GET-FILES apiversion=V3 latency=913ms
     status=Failure
2025-09-10T10:01:06.002Z
     logID=550e8400-e29b-41d4-a716-446655440112
     datacenter=DC1 cluster=NULL node=N03
     api=POST-RESET apiversion=V2 latency=75ms
     status=Success
2025-09-10T10:01:06.345Z
     logID=550e8400-e29b-41d4-a716-446655440113
     datacenter=DC2 cluster=PrdC04 node=N06
     api=GET-PWD apiversion=V4 latency=12ms
     status=Success
```

Figure 5: Excerpt of raw log entries from the incident.

The selected case not only contains a clearly identifiable fault-indicating attribute combination, but also exhibits common imperfections such as missing or ambiguous values. These issues exemplify the difficulties engineers face when applying contrast-based anomaly localization [20], while demonstrating how Aloha provides human-in-the-loop guidance to reduce cognitive load and support effective anomaly recognition and localization.

4.1 Scenario Check

When Aloha received the raw request-level logs along with a brief description of the scenario from the engineer, it first generated a lightweight summary of the inputs, capturing file structures, attribute distributions, and sample contents. This summary was then evaluated against the scenario suitability criteria C1–C5.

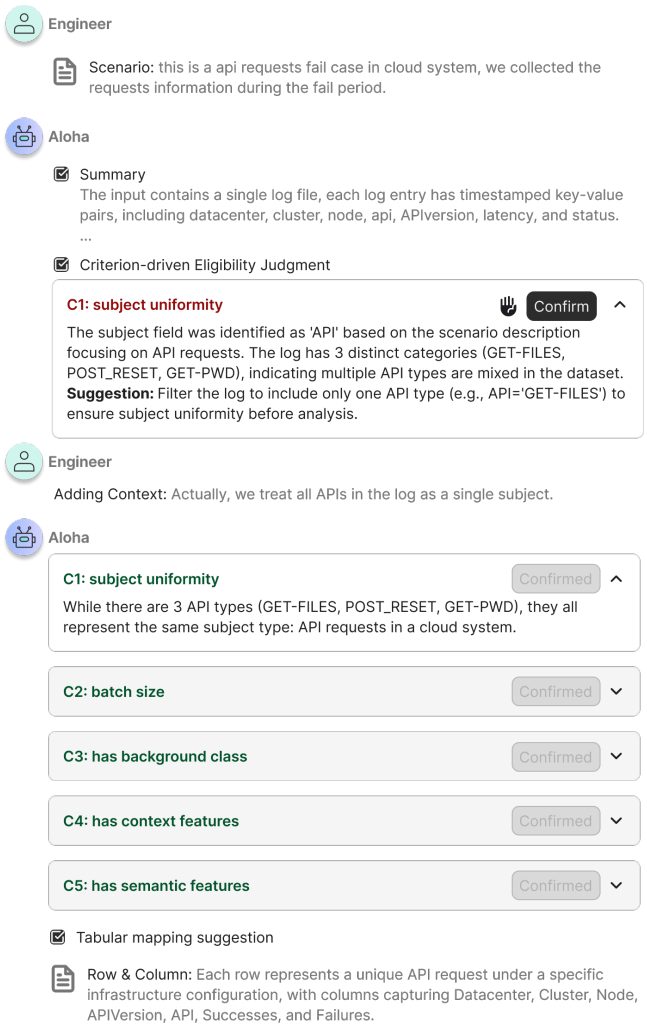


Figure 6: Phase 1 Scenario Check

As shown in Figure 6, Aloha initially misjudged C1 due to the presence of three distinct API types. Through human-in-the-loop interaction, the engineer provided clarifying information about the scenario and its requirements. With this guidance, Aloha corrected the judgment and proceeded to evaluate the remaining criteria. Upon completing all checks, the system successfully generated a tabular mapping suggestion, specifying which attributes should be extracted as columns and how instances should be represented as rows for subsequent analysis.

4.2 Data Validation

In Data Validation Phase, the tabular input provided by the engineer—consisting of 21,337 rows derived from raw request-level logs—is examined through the workflow shown in the Figure 7. Each row represents an individual request, with columns describing its attributes such as timestamp, datacenter, cluster, node, latency, and API outcome status.

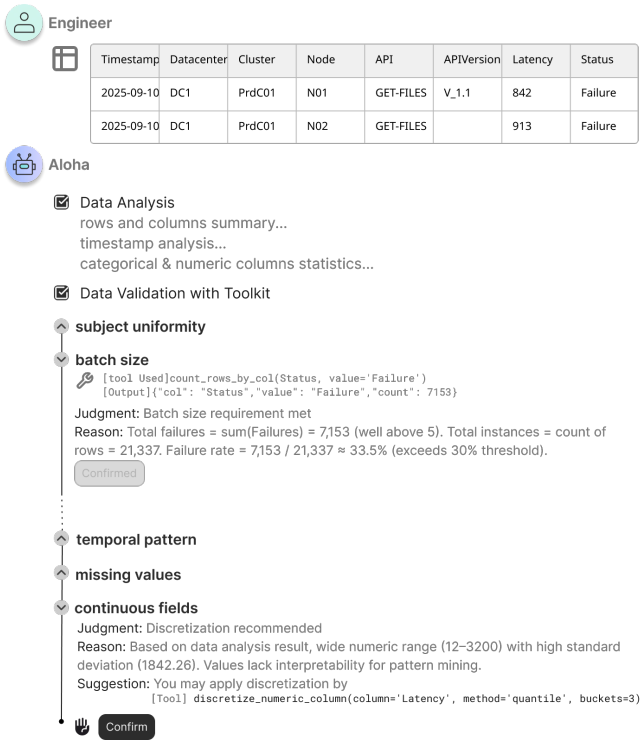


Figure 7: Phase 2a Data Validation

Aloha first conducts a preliminary data analysis to understand the table's overall structure, including its rows, columns, and field characteristics. The results serve as preparatory statistics that inform subsequent validation along the backbone of data checks. Along this backbone, each pitfall is checked with semantic and executable tools.

For the Batch Size check, Aloha called a tool to count failures, providing quantitative evidence for judgement. For the Continuous Fields check, Aloha detected that the *Latency* is a continuous attribute that may blur categorical contrasts, hindering interpretable pattern discovery. Accordingly, Aloha initially recommended discretization using quantile-based binning [17], but the engineer rejected this suggestion and required an alternative handling, as shown in Figure 8; Aloha then executed the instruction via tool and completed the judgment. By contrast, for the Label Type check the engineer agreed with Aloha's recommendation to aggregate discrete labels. Other checks were also performed but are omitted here for brevity. At the end of validation, the validated data was confirmed to support successful execution of the default search algorithm.

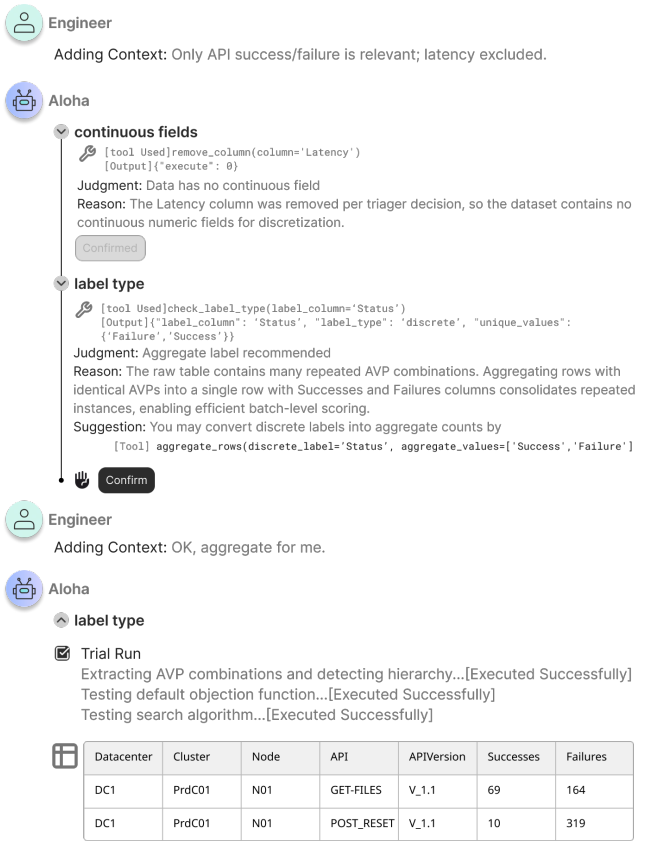


Figure 8: Phase 2b Data Validation

As shown in Figure 9, the validated data incorporates key adjustments, such as label aggregation and removal of irrelevant continuous fields, ensuring it can be directly consumed by the diagnosis algorithm.

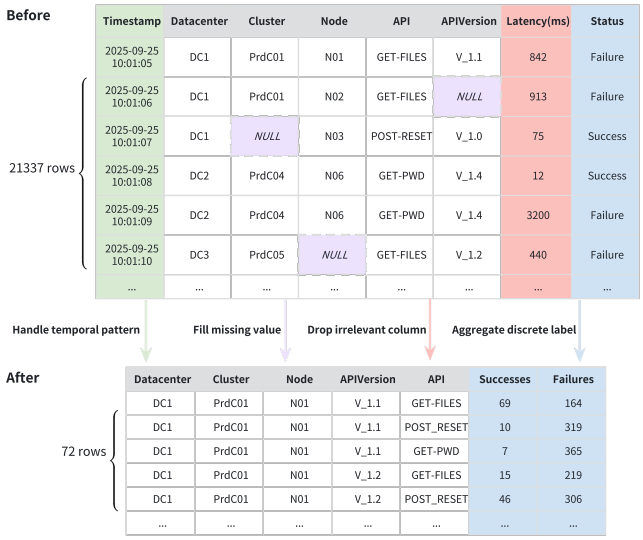


Figure 9: Data validation transformations (Before → After)

4.3 Anomaly Localization

Finally, Aloha performs contrast-based pattern search for anomaly localization. As illustrated in Figure 10, Aloha retrieves relevant historical cases via its *RAG-based Strategy Selection*, recommends target functions and parameters, and executes the meta-search algorithm. When encountering a new scenario with no prior cases, or when historical strategies have become outdated due to system updates, Aloha relies on generic contrast-based approaches guided by scenario understanding and data profiling to select suitable objectives and parameters, ensuring that the anomaly localization process can still proceed effectively. Table 3 presents the resulting anomaly localization report, including the top-k root causes, Aloha's recommendations, and related historical cases.

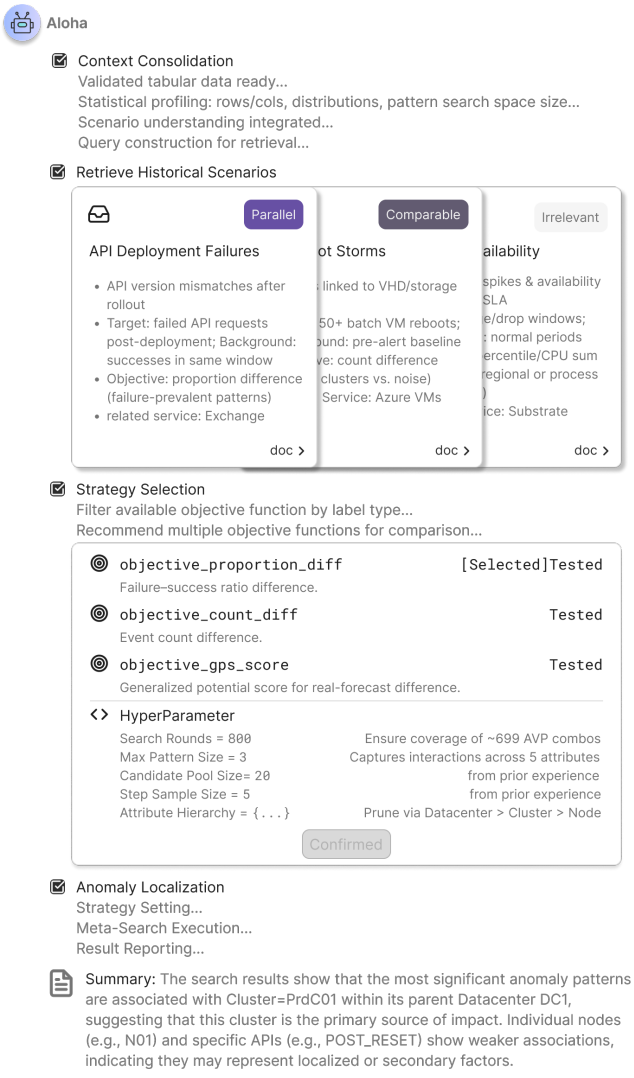


Figure 10: Phase 3 Anomaly Localization

This case study demonstrates how Aloha's three-phase workflow and HITL integration enable reliable anomaly localization despite imperfect data.

Table 3: Anomaly Report for an API Request Failure Case

Field	Description
Incident ID	522673
DateTime	2025-09-12 14:54:18
Top-5 Root Cause	1. Pattern {Cluster: PrdC01} Score: 0.7425 2. Pattern {Cluster: PrdC01, Datacenter: DC1} Score: 0.7425 3. Pattern {Datacenter: DC1} Score: 0.5592 4. Pattern {Node: N01} Score: 0.4029 5. Pattern {Node: N02} Score: 0.3395
Aloha's Recommendations	**Cluster PrdC01** – Investigate the cluster for potential configuration issues, service degradation, or resource contention. Apply targeted monitoring and corrective actions to stabilize request processing. **Cluster PrdC01 and Datacenter DC1** – Assess interactions between the cluster and its parent datacenter, including network latency, failover configurations, and cross-cluster dependencies. **Nodes N01 and N02** – Review each node for performance regressions, error logs, or abnormal load. Apply node-level remediation such as workload redistribution.
Related Historical Incidents	519986, 521238, 527881

5 Discussion

5.1 Observations from Usage

Motivated by a focus on engineers' practical needs, we collected feedback from engineers who used Aloha during batch failure investigations. Those observations suggest that Aloha changes how diagnosis work is approached, rather than merely accelerating existing workflows. Instead of manually reasoning about scenario suitability, data preparation, and diagnostic formulation in an ad-hoc manner, engineers reported that Aloha provided a clearer structure for organizing these decisions and making underlying assumptions explicit. This shift allowed engineers to focus more on validating interpretations and reasoning about results, rather than constructing the localization process itself.

As part of our usage study, we specifically asked engineers whether the human-in-the-loop design in Aloha was helpful in practice, rather than introducing additional overhead. Engineers generally did not expect full automation. Instead, they reported that being able to interact with Aloha during anomaly localization–confirming interpretations and revisiting intermediate decisions–was more aligned with their day-to-day workflows. Several engineers noted that during on-call fatigue, they may reflexively confirm intermediate decisions, increasing the risk of over-trusting the agent. In such cases, they found it useful that Aloha allows decisions to be adjusted incrementally, and supports phase-wise rollback with intermediate artifacts (e.g., agent reasoning traces and case metadata) preserved for review and correction. These observations underscore the need to ensure the agent's behavior remains safe and reliable, motivating future improvements in its robustness and error-handling mechanisms.

5.2 Threats to Validity

The validity of our study is constrained by limitations inherent to a proof-of-concept deployment. First, the **evaluation scope** is limited in scale. Although Aloha was applied to multiple real-world batch failure cases across diverse scenarios, the incident volume and deployment duration are insufficient for statistically robust metrics such as MTTM or override rates. Future work will involve longer-term deployment and broader organizational settings to enable more quantitative analysis.

Second, the **evaluation methodology** emphasizes usability and workflow support over direct algorithmic comparison. As Aloha targets the usability gap of contrast-based anomaly localization, effectiveness is assessed through case walkthroughs and engineer feedback rather than benchmark-style comparisons. Future studies will incorporate stage-wise experiments, tool ablations, and component-level evaluations to complement the current qualitative assessment.

6 Related Work

Contrast-based anomaly localization. As shown in Table 1, many prior methods target specific batch failure scenarios, where differences in scenarios often correspond to diverse data sources, including logs [12, 13, 29, 54], traces [45, 46], metrics [21, 22, 40, 42], and structured reports [4, 23, 28, 31]. CONAN [20] generalizes diagnosis across such scenarios through a unified formalization of batch failures. However, operational use reveals a usability gap, which Aloha mitigates by interactively guiding engineers via a three-phase workflow.

LLM-based Agent framework in AIOps. Recent advances in large language models have enabled agentic systems for IT operations [24, 25, 37, 49], supporting tasks like incident triage [18, 43, 47], root cause analysis [6, 15, 30, 41, 48, 51, 53], and failure mitigation [1, 16, 38]. While these works primarily focus on single-instance or full-service failures, our work targets batch failures level analysis. Aloha draws on these ideas to provide interactive anomaly localization in batch failure scenarios.

7 Conclusion

In this paper, we identify the problem of recognizing and localizing batch failures in cloud systems, which is insufficiently supported by existing approaches. To address this problem, we present Aloha, an end-to-end framework that leverages LLM-based agents to guide engineers through scenario confirmation, data preparation, and strategy selection for anomaly localization. Our deployment in Microsoft's cloud services shows that Aloha streamlines data handling and reduces engineers' cognitive burden, validating its practicality and demonstrating the value of integrating LLM-agent capabilities for more intelligent and reliable failure management.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (62272249, 62302244), and the Fundamental Research Funds for the Central Universities (XXX-63253249).

References

[1] Kaikai An, Fangkai Yang, Junting Lu, Liqun Li, Zhixing Ren, Hao Huang, Lu Wang, Pu Zhao, Yu Kang, Hua Ding, et al. 2024. Nissist: An incident mitigation copilot based on troubleshooting guides. *arXiv preprint arXiv:2402.17531* (2024).

[2] Stephen D Bay and Michael J Pazzani. 1999. Detecting change in categorical data: Mining contrast sets. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 302–306.

[3] Sid Black, Asa Cooper Stickland, Jake Pencharz, Oliver Sourbut, Michael Schmatz, Jay Bailey, Ollie Matthews, Ben Millwood, Alex Remedios, and Alan Cooney. 2025. RepliBench: Evaluating the Autonomous Replication Capabilities of Language Model Agents. *arXiv preprint arXiv:2504.18565* (2025).

[4] Marco Castelluccio, Carlo Sansone, Luisa Verdoliva, and Giovanni Poggi. 2017. Automatically analyzing groups of crashes for finding correlations. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 717–726.

[5] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 111–120.

[6] 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.

[7] Guozhu Dong and Jinyan Li. 1999. Efficient mining of emerging patterns: discovering trends and differences. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA) *(KDD '99)*. Association for Computing Machinery, New York, NY, USA, 43–52. doi:10.1145/312129.312191

[8] Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. 2025. DeepResearch Bench: A Comprehensive Benchmark for Deep Research Agents. *arXiv preprint arXiv:2506.11763* (2025).

[9] Clifton A Ericson and Clifton Ll. 1999. Fault tree analysis. In *System Safety Conference, Orlando, Florida*, Vol. 1. 1–9.

[10] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*. 6491–6501.

[11] Antal Haans. 2018. Contrast analysis: A tutorial. *Practical Assessment, Research & Evaluation* 23, 9 (2018), 1–21.

[12] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 60–70.

[13] Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, et al. 2022. An empirical study of log analysis at Microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1465–1476.

[14] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. 2024. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679* (2024).

[15] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, et al. 2024. Xpert: Empowering incident management with query recommendations via large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[16] 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.

[17] Randy Kerber. 1992. ChiMerge: Discretization of Numeric Attributes. *AAAI Press* (1992).

[18] Jinxi Kuang, Jinyang Liu, Junjie Huang, Renyi Zhong, Jiazhen Gu, Lan Yu, Rui Tan, Zengyin Yang, and Michael R Lyu. 2024. Knowledge-aware alert aggregation in large-scale cloud systems: a hybrid approach. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 369–380.

[19] 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.

[20] Liqun Li, Xu Zhang, Shilin He, Yu Kang, Hongyu Zhang, Minghua Ma, Yingnong Dang, Zhangwei Xu, Saravan Rajmohan, Qingwei Lin, et al. 2023. Conan: Diagnosing batch failures for cloud systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 138–149.

[21] Zeyan Li, Junjie Chen, Yihao Chen, Chengyang Luo, Yiwei Zhao, Yongqian Sun, Kaixin Sui, Xiping Wang, Dapeng Liu, Xing Jin, et al. 2023. Generic and robust root cause localization for multi-dimensional data in online service systems. *Journal of Systems and Software* 203 (2023), 111748.

[22] Zeyan Li, Chengyang Luo, Yiwei Zhao, Yongqian Sun, Kaixin Sui, Xiping Wang, Dapeng Liu, Xing Jin, Qi Wang, and Dan Pei. 2019. Generic and robust localization of multi-dimensional root causes. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 47–57.

[23] Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, and Dongmei Zhang. 2016. iDice: Problem identification for emerging issues. In *Proceedings of the 38th International Conference on Software Engineering*. 214–224.

[24] Jun Liu, Chaoyun Zhang, Jiaxu Qian, Minghua Ma, Si Qin, Chetan Bansal, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2025. Large language models can deliver accurate and interpretable time series anomaly detection. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 4623–4634.

[25] Yuhe Liu, Changhua Pei, Longlong Xu, Bohan Chen, Mingze Sun, Zhirui Zhang, Yongqian Sun, Shenglin Zhang, Kun Wang, Haiming Zhang, Minghua Ma, and Dan Pei. 2023. Opseval: A comprehensive benchmark suite for evaluating large language models' capability in it operations domain. *arXiv preprint arXiv:2310.07637* (2023).

[26] Microsoft Azure. 2025. Preliminary Post Incident Review – Multi-service impact in Switzerland North. https://azure.status.microsoft/en-us/status/history/. Accessed: October 5, 2025.

[27] Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal. 2023. Human-in-the-loop machine learning: a state of the art. *Artificial Intelligence Review* 56, 4 (2023), 3005–3054.

[28] Vijayaraghavan Murali, Edward Yao, Umang Mathur, and Satish Chandra. 2021. Scalable statistical root cause analysis on app telemetry. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 288–297.

[29] Karthik Nagaraj, Charles Killian, and Jennifer Neville. 2012. Structured comparative analysis of systems logs to diagnose performance problems. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 353–366.

[30] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, et al. 2025. Flow-of-Action: SOP Enhanced LLM-Based Multi-Agent System for Root Cause Analysis. In *Companion Proceedings of the ACM on Web Conference 2025*. 422–431.

[31] Rebecca Qian, Yang Yu, Wonhee Park, Vijayaraghavan Murali, Stephen Fink, and Satish Chandra. 2020. Debugging crashes using continuous contrast set mining. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 61–70.

[32] Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. 2023. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541* (2023).

[33] Xiang Jenny Ren, Sitao Wang, Zhuqi Jin, David Lion, Adrian Chiu, Tianyin Xu, and Ding Yuan. 2023. Relational Debugging—Pinpointing Root Causes of Performance Problems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 65–80.

[34] Carl Orge Retzlaff, Srijita Das, Christabel Wayllace, Payam Mousavi, Mohammad Afshari, Tianpei Yang, Anna Saranti, Alessa Angerschmid, Matthew E Taylor, and Andreas Holzinger. 2024. Human-in-the-loop reinforcement learning: A survey and position on requirements, challenges, and opportunities. *Journal of Artificial Intelligence Research* 79 (2024), 359–415.

[35] Raja R Sambasivan, Alice X Zheng, Michael De Rosa, Elie Krevat, Spencer Whitman, Michael Stroucken, William Wang, Lianghong Xu, and Gregory R Ganger. 2011. Diagnosing performance changes by comparing request flows. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*.

[36] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2023), 38154–38180.

[37] Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shachee Mishra Gupta, et al. 2024. Building AI Agents for Autonomous Clouds: Challenges and Design Principles. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 99–110.

[38] Binpeng Shi, Yu Luo, Jingya Wang, Yongxin Zhao, Shenglin Zhang, Bowen Hao, Chenyu Zhao, Yongqian Sun, Zhi Zhang, Ronghua Sun, et al. 2025. FlowXpert: Expertizing Troubleshooting Workflow Orchestration with Knowledge Base and Multi-Agent Coevolution. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 4839–4850.

[39] CRN Staff. 2025. The 10 Biggest Cloud Outages of 2025 (So Far). *CRN* (2025). https://www.crn.com/news/cloud/2025/the-10-biggest-cloud-outages-of-2025-so-far Accessed: 2025-10-05.

[40] Yongqian Sun, Daguo Cheng, Pengxiang Jin, Quan Ding, Shenglin Zhang, Xu Chen, Yuzhi Zhang, Minghan Liang, Dan Pei, Jianyan Zheng, et al. 2022. Robust anomaly clue localization of multi-dimensional derived measure for online video services. *IEEE Transactions on Services Computing* 16, 2 (2022), 1387–1401.

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

[42] Yongqian Sun, Youjian Zhao, Ya Su, Dapeng Liu, Xiaohui Nie, Yuan Meng, Shiwen Cheng, Dan Pei, Shenglin Zhang, Xianping Qu, et al. 2018. Hotspot: Anomaly localization for additive kpis with multi-dimensional attributes. *IEEE Access* 6 (2018), 10909–10923.

[43] Zexin Wang, Jianhui Li, Minghua Ma, Ze Li, Yu Kang, Chaoyun Zhang, Chetan Bansal, Murali Chintalapati, Saravan Rajmohan, Qingwei Lin, et al. 2024. Large language models can provide accurate and interpretable incident triage. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 523–534.

[44] 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.

[45] Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. 2014. Crashlocator: Locating crashing faults based on crash stacks. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. 204–214.

[46] Xiao Yu, Shi Han, Dongmei Zhang, and Tao Xie. 2014. Comprehending performance from real-world execution traces: A device-driver case. In *Proceedings of the 19th international conference on architectural support for programming languages and operating systems*. 193–206.

[47] Zhaoyang Yu, Aoyang Fang, Minghua Ma, Jaskaran Singh Walia, Chaoyun Zhang, Shu Chi, Ze Li, Murali Chintalapati, Xuchao Zhang, Rujia Wang, et al. 2025. Triangle: Empowering Incident Triage with Multi-Agent. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 674–686.

[48] 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.

[49] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip Yu, and Ying Li. 2025. A Survey of AIOps in the Era of Large Language Models. *Comput. Surveys* (2025).

[50] 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.

[51] Shenglin Zhang, Sibo Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, and Dan Pei. 2025. Failure diagnosis in microservice systems: A comprehensive survey and analysis. *ACM Transactions on Software Engineering and Methodology* (2025).

[52] Xu Zhang, Chao Du, Yifan Li, Yong Xu, Hongyu Zhang, Si Qin, Ze Li, Qingwei Lin, Yingnong Dang, Andrew Zhou, et al. 2021. Halo: Hierarchy-aware fault localization for cloud systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3948–3958.

[53] 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.

[54] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1253–1263.