An Agentic Framework for Triaging Incidents in Production Cloud Infrastructure

Yuhan Yao*
Microsoft
Cambridge, MA, USA
yuhanyao@microsoft.com

Yuxuan Jiang*
University of Michigan
Ann Arbor, MI, USA
jyuxuan@umich.edu

Minghua Ma
Microsoft
Redmond, WA, USA
minghuama@microsoft.com

Madhura Vaidya
Microsoft
Redmond, WA, USA
mavaidya@microsoft.com

Jieren Deng
Microsoft
Cambridge, MA, USA
jierendeng@microsoft.com

Yigong Hu
Boston University
Boston, MA, USA
yigongh@bu.edu

Chetan Bansal
Microsoft
Redmond, WA, USA
chetanb@microsoft.com

Ze Li
Microsoft
Redmond, WA, USA
zeli@microsoft.com

Murali Chintalapati
Microsoft
Redmond, WA, USA
muralic@microsoft.com

Abstract

Incidents occur frequently in large-scale cloud infrastructures and can cause significant service disruptions if not triaged and mitigated promptly. Existing automated triage systems rely on complete incident reports or uniform agent quality, which are often infeasible in production due to fragmented observability and heterogeneous team maturity. We present *Comfey*, a decentralized agentic triage framework designed for this operational reality. Each team in Azure deploys a lightweight *Comfey* triage agent that analyzes domain-specific data, decides whether to accept or reject an incident, and collaborates with other agents via a shared routing table. *Comfey* has been running in Azure production for 22 months, where it has triaged around 19,500 incidents with 91.5% accuracy (validated by on-call engineers). It reduces average triage time by 6 times and mitigation time by 4.3 times compared to manual processes, with minimal setup requirements.

CCS Concepts

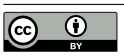
• **Software and its engineering** → *System administration*; *Ultra-large-scale systems*; • **Computer systems organization** → **Cloud computing**.

Keywords

Incident Triage, Cloud Infrastructure, Multi-agent Systems, Large Language Models, AIOps

ACM Reference Format:
Yuhan Yao, Yuxuan Jiang, Minghua Ma, Madhura Vaidya, Jieren Deng, Yigong Hu, Chetan Bansal, Ze Li, and Murali Chintalapati. 2026. An Agentic Framework for Triaging Incidents in Production Cloud Infrastructure. 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, 11 pages. https://doi.org/10.1145/3803437.3805228

*Both authors contributed equally to this research.



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3805228

1 Introduction

In today's cloud infrastructures, incidents occur frequently in production, significantly damaging both system reliability and user experience [4, 11, 14, 20, 35]. In 2025, a widely publicized AWS regional outage caused large-scale service degradation, disrupting core services such as EC2, S3, and Lambda, causing broad service disruptions across the internet and significant economic damage [3]. When a cloud incident occurs, it is critical to mitigate it quickly to prevent the impact from escalating [4, 9].

However, finding the correct team to mitigate an incident is often challenging. Modern cloud systems comprise numerous independent yet interacting services, each specializing in a specific functionality [24, 29]. For example, executing a VM creation job requires the compute service to interface with both the storage system and the cluster management service. An incident of high VM creation failure rate might be rooted in the compute, storage, or cluster management team. The process of assigning an incident to the appropriate team is known as incident triage [5, 25]. Incorrect triage can thus significantly prolong diagnosis and delay mitigation.

Consider a real-world cloud incident that resulted in a service outage due to slow triage, which delayed mitigation. During maintenance, an engineer decommissioned several storage nodes but failed to update the internal node list used by the compute team. As a result, when the compute service attempted to allocate a VM on a decommissioned node, the request failed, causing service degradation that eventually escalated into an outage (as shown in Figure 1). The health monitors captured an increased frequency of VM creation failures on the compute service side and reported the incident. The incident was thus mis-triaged to the compute team. Lacking awareness of the node decommission status, the compute team checked its own code but could not find any issue, and eventually

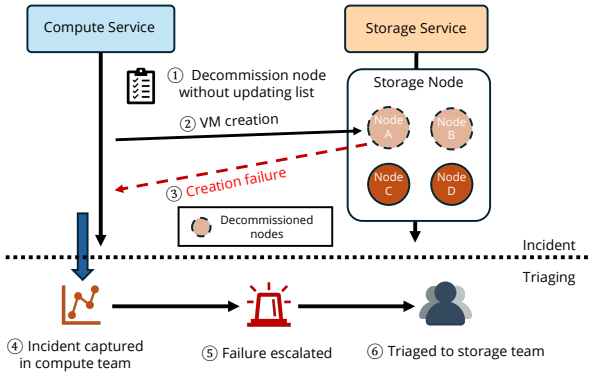


Figure 1: An incident of VM creation failure

dismissed the report, leaving the problem to escalate into a full system outage.

Current incident triage strategies primarily focus on leveraging historical incident data and machine learning techniques to automate the triage process [13, 16, 17, 22, 36]. By analyzing the incident data and how they are fixed, these approaches identify recurring patterns and extract rules to route new incidents to the appropriate teams more efficiently. However, they generally assume that incident reports are complete and accurate, containing detailed metrics and logs that capture critical patterns [2, 28]. In production cloud systems, incident reports are often noisy and incomplete due to system complexity and limited observability across components, making accurate triage difficult. Recent work [10, 27, 33, 38] addresses this limitation by leveraging large language models (LLMs) to extract relevant textual information from incident data to guide the triage process. LLMs show great potential in extracting useful information even when incident data are heterogeneous and incomplete. However, when deployed in large-scale production environments, such LLM-based triage systems still fail to perform reliably.

We observe three challenges when deploying automated triage systems in production cloud infrastructures. (1) **Complex operation chains:** cloud operations are typically executed through multiple interdependent services. When an incident occurs, every software component in the execution chain becomes a potential root cause, creating a large search space for triage. (2) **Fragmented data and sovereignty:** In a large organization, data is siloed. Teams guard their raw logs and monitoring data for privacy and compliance, making "centralized observability" impossible in practice. A workable system must respect these data boundaries. (3) **The Friction Barrier:** Perhaps the biggest hurdle is sociological. Teams are reluctant to adopt complex new tools that require significant manual effort (e.g., writing detailed specs or adhering to rigid agent schemas). A practical system must have low friction, allowing teams to onboard effortlessly using their existing assets (historical tickets) without mandating uniform agent quality or protocol compliance. This heterogeneity, where some agents are smart and others are basic, is a hard constraint of real-world deployment.

These challenges suggest that an effective triage system for production clouds should satisfy three properties. First, it should operate under fragmented observability, allowing each team to analyze incidents using its own local signals and tools rather than assuming centralized access to raw data. Second, it should support collaboration across teams without requiring tightly coupled coordination or uniform agent sophistication. Third, it should remain lightweight to adopt, so that teams can onboard using existing assets such as historical incidents and troubleshooting workflows rather than building complex new infrastructure.

In this paper, we present *Comfey*, an agentic triage system in large production cloud infrastructures. *Comfey* is highly scalable and can analyze incident data on millions of nodes in Azure. *Comfey* has high accuracy in detecting incidents to correct the team and helps on-call engineers pinpoint the root causes. At the same time, *Comfey* has low overhead and can run in a production environment.

Two key insights motivate the design of *Comfey* and enable it to achieve these properties. First, we purposely trade off the theoretical optimality of a centralized solver for the practicality of a decentralized system. While a highly orchestrated multi-agent system might theoretically achieve higher accuracy by negotiating every decision, such systems fail in production due to the fragility of "weakest link" agents. *Comfey* instead decouples agent success: it decomposes triage into local accept/reject decisions and a robust, history-driven routing mechanism. This ensures that even if some agents are basic, the Global Routing Table (built from successful paths) guides incidents correctly. Second, a decentralized and team-scoped approach is essential for non-intrusive metadata collection. Each agent operates with its team's domain insight, makes decisions on whether to accept an incident, and collaborates via "stigmergic" traces (the routing table) rather than direct negotiation.

Based on these insights, *Comfey* adopts a decentralized agentic workflow. Instead of a single centralized agent to handle all incidents, each team deploys a dedicated agent specialized in analyzing its domain-specific incident data. When a new incident occurs, *Comfey* first assigns it to the team most likely responsible. If that team's agent rejects the incident, *Comfey* forwards it to the next most likely team, guided by historical records of how incidents have been transferred across teams. This process continues until the incident is correctly assigned. Each agent in *Comfey* maintains a feedback loop that logs its selection decisions, along with engineer confirmations of correctness, enabling the agents to adapt to evolving infrastructure conditions continuously.

To handle fragmented incident data, *Comfey* introduces an *enrichment* pipeline that integrates directly with major monitoring platforms used in Azure. Each team can deploy this pipeline with lightweight parameters, such as table names or query specifications, to expose the raw monitoring data. The pipeline would automatically collect and analyze the noisy and heterogeneous incident data. *Comfey* applies a data distillation mechanism that extracts incident-relevant signals from logs, performance metrics, and engineers' discussions on incidents and formalizes them into a uniform incident summary. The distillation mechanism not only aligns incident symptoms with domain-specific knowledge but also guides how incidents should be routed across teams.

Each agent determines whether to accept an incident by matching it against historical incidents previously accepted by its team or the troubleshooting guides written by engineers. If a match is found, the agent accepts the incident; otherwise, it consults a Global Routing Table to identify the next most likely team. The Global Routing Table is collaboratively built across all teams, recording

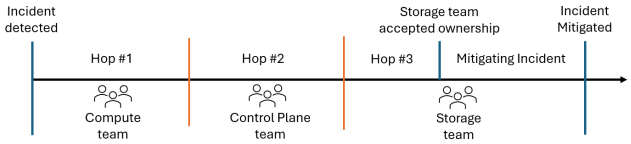


Figure 2: The incident triage process

each agent's most frequent historical incidents and routing history, which enables efficient and informed cross-team transfers.

Comfey provides a self-onboarding platform that allows any team to deploy it with fewer than ten command-line instructions. It has been running in production in Azure for over a year, successfully triaging around 19,500 incidents with an accuracy of 91.5%. By accelerating incident routing and diagnosis, *Comfey* has helped on-call engineers mitigate issues 4.3× faster on average.

In summary, our contributions are:

- We design and implement *Comfey*, a decentralized, agentic triage system that enables team-specific automation and collaborative routing across large-scale cloud platforms.
- We introduce a team-scoped agent framework that supports modular enrichment, retrieval-based reasoning, and semantic distillation of incident context, enabling robust triage from fragmented and heterogeneous signals.
- We deploy *Comfey* in production within Azure, where it has been used to triage over 19,500 real-world incidents across more than 20 teams, demonstrating scalability, adaptability, and effectiveness in live operations.

2 Background

2.1 In-Production Triage Process

In cloud services, incidents are generated by automated detection systems or user reports. In both cases, initial information about an incident is too limited to determine its root cause. The cloud system must therefore assign the incident to a specific team for further diagnosis and mitigation, a process known as incident triage.

The triage process typically has four stages: **(1) Initial Assignment.** When an incident occurs, it is usually assigned to the team responsible for the component where the error signal was captured. **(2) Team Assessment.** The assigned team reviews the incident report along with its own monitoring data to investigate the potential root cause. During this process, the team may involve other teams if they suspect that the case is related to another team. If the team determines that the incident is within its scope, it accepts ownership and begins mitigation efforts. **(3) Escalation Decision.** If the team determines that the incident falls outside its scope, the on-call engineer must escalate it to another team. This involves identifying candidate teams, preparing a transfer justification with supporting evidence, and communicating the relevant context to the receiving team. **(4) Iterative Resolution.** Steps (2) and (3) repeat until the correct team is identified and accepts responsibility for mitigation.

Figure 2 shows the triage process of the VM creation incident described in Figure 1. When the detection system observed an increasing frequency of VM creation failures, it automatically generated an incident ticket and assigned it to the compute team. The compute team verified the failures using its own monitoring data but was unable to identify the root cause, as the issue lies outside the compute service. After discussions with other teams, the incident was escalated to the control plane team, which is responsible for dispatching incidents. By reviewing historical incidents, the control plane team found that the storage team had previously resolved a similar VM failure. Based on this evidence, the incident was routed to the storage team, which confirmed the root cause and quickly mitigated the issue by updating the live node list.

This workflow highlights several challenges that incident triage must address. Teams are often required to make ownership decisions based on incomplete information and limited visibility into the domains of other teams. For example, the compute team may not be aware of which storage nodes have been decommissioned. The process must also operate under significant time pressure, as the triage must be completed quickly to prevent service degradation. In large cloud systems, each team may receive hundreds of incident tickets, many of which are false positives, while only one or two on-call engineers are available to handle them. Finally, the triage process is inherently collaborative but limited by organizational boundaries, observability between teams, and communication latency.

2.2 Key Findings in Incident Triage

To gain a deeper understanding of incident triage in cloud environments, we conduct an empirical study of the triage process using incident records from Azure's incident management system. The dataset includes incidents of all severity levels and spans fifteen teams during the period from March 2024 to Jan 2026. We focus on incidents that required manual investigation and exclude those that on-call engineers marked as duplicates or false alarms, as well as those automatically mitigated by monitoring and self-mitigation tools. The goal of this study is to identify the practical challenges that engineers encounter during triage and to characterize the standard practices they adopt when responding to incidents.

Due to time constraints, we were unable to review every incident manually. Fortunately, Azure's incident management system provides a structured interface that requires on-call engineers to record team assignments, discussion threads, bug patterns, final ownership, and conclusions. Using this structured data, we developed automated analysis tools to extract key metrics, such as diagnosis time and the teams involved, and to classify the diagnostic challenges the teams encountered.

Our analysis shows six critical findings that directly inform the design requirements for automated, cloud-scale triage systems.

Finding 1: Many real-world incidents involve multiple teams: Cloud incidents often span multiple services, typically involving between 5 and 30 teams. This is inherent to large-scale cloud systems, where a single operation requires coordination across several services. As a result, a substantial portion of the incidents we studied involved one or more team handoffs before reaching the team capable of resolving the issue. Some incidents required coordination among more than 20 teams, often due to ambiguous error symptoms or unclear ownership boundaries. These frequent handoffs introduced delays, duplicated diagnostic efforts, and misaligned investigations during the early stages of triage.

Finding 2: Incident reports often lack sufficient context for triage. Many incident tickets are generated automatically by health monitors and contain only high-level symptom descriptions (e.g.,

latency spike or error rate increase). These reports often lack actionable diagnostic information and may have highly similar titles and metadata, even when caused by different underlying issues. We observed that 85% of incidents had duplicate or near-duplicate reports assigned to different teams, reflecting the ambiguity in early signals and ownership boundaries. In many cases, additional investigation was needed before the correct team could be identified.

Finding 3: Diagnostic workflows are highly domain-specific. Each team in our study employed unique investigation practices based on their tools, logs, and heuristics. Even for similar symptoms, the steps taken and information referenced varied significantly across teams.

Finding 4: Human response latency slows triage. Our analysis shows that cross-team communication introduces substantial latency. Response delays averaged ~40 minutes during business hours and extended to 8 hours after hours. Misunderstandings, unclear explanations, and a lack of context further slowed handoffs.

Finding 5: Triage decisions evolve. Initial ownership assignments were often tentative. In 12.1% of Azure incidents in a month, teams initially accepted ownership but later escalated or transferred the case after further investigation. These decisions were based on confidence, not certainty.

Finding 6: Historical precedent is underutilized. Teams frequently referred to prior incidents informally when triaging similar cases. However, systematic use of historical precedent was rare. Incidents with known precedents are mitigated 2.62 times faster, but only 11.05% of cases included explicit precedent lookup.

Design Implication: These findings underscore the need for team-specific triage agents equipped with local diagnostic knowledge and infrastructure. Because cloud incidents involve a large number of candidate teams, and the automatically generated incident reports often contain limited information, effective triage requires using domain-specific signals, heuristics, and monitoring data available within each team to efficiently identify the correct owner.

2.3 Challenges and Opportunities

Many state-of-the-art triage systems assume global observability, where a single agent has access to metrics, logs, and traces across all services. This assumption fails in real-world cloud environments for two reasons. First, incident data is often siloed, with each service team collecting and storing its own logs and telemetry. Making triage decisions requires domain-specific knowledge that is not universally shared. Second, at cloud scale, incidents can involve millions of nodes, resulting in large data volumes that make centralized analysis expensive and slow. As a result, such systems often suffer from high false positive rates and long triage delays in practice.

Due to these limitations, most production environments still rely on manually authored triage rules documented in Troubleshooting Guides (TSGs). These guides instruct on-call engineers on how to collect and interpret various signals and when to escalate incidents to specific teams. However, TSGs are labor-intensive to create and update. They become outdated as services evolve and often lack coverage for new or team-specific failure modes.

Recent work on Large Language Models (LLMs) has shown strong capabilities in understanding and reasoning over textual artifacts such as logs, incident reports, and diagnostic summaries [10, 27, 32, 34, 37, 38]. Microsoft has explored using LLM-based agents for incident triage [33], where an agent processes logs to assist in incident handling. However, these systems face trade-offs at scale. Processing all available data provides context but increases cost and latency. Trimming inputs improves performance but risks omitting important signals and reducing accuracy.

Our experience revealed that the main bottleneck is centralization. A single LLM agent cannot scale effectively across services with diverse data and failure modes. To address this, we adopt a decentralized design. Each service runs its own LLM-based agent trained on domain-specific signals. These agents determine if an incident falls under their team's responsibility. If not, the incident is forwarded to the next most likely team, forming a collaborative triage workflow.

3 Design of the *Comfey* System

Motivated by the challenges observed in incident triage, we design *Comfey*, an agentic and decentralized framework deployed in Azure's production infrastructure to automate incident triage and assist engineers in efficient incident mitigation. *Comfey* deploys team-scoped agents that make local triage decisions on whether to accept an incident. When an agent rejects an incident, it collaborates with other agents to identify the next most likely team. Through this decentralized collaboration, *Comfey* achieves scalable and accurate triage without requiring global observability.

3.1 Design Requirement

There are several challenges and requirements for triaging incidents in large-scale cloud infrastructure software.

(1) **High scalability.** Cloud incidents are highly diverse in terms of components, root causes, incident patterns, and deployment scale. A triage framework must scale to handle thousands of concurrent incidents across millions of nodes.
(2) **Fragmented data.** Effective triage requires analyzing incident data scattered across different software components. Thus, the triage system must not assume visibility to complete incident data.
(3) **High accuracy.** Incident triage must be correct. False positives waste significant engineering time and can lead to delayed mitigation.
(4) **Non-intrusive and low-overhead.** The framework should not require code modifications to existing systems and must impose minimal runtime and operational overhead.
(5) **Adaptability to software evolution.** Cloud software evolves continuously. The triage system must update its knowledge automatically.

These requirements motivate the design of a decentralized, agentic framework. We highlight three key design choices addressing the above challenges:

Design Rationale 1: Decentralization via Teams. To address *Fragmented Data* and *Scale*, we reject the centralized "God Agent"

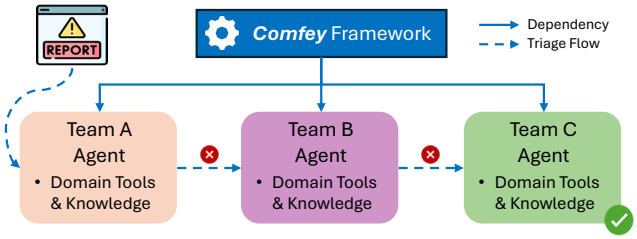


Figure 3: *Comfey*'s Decentralized Architecture.

model. Instead, we map agents 1:1 to engineering teams. This respects data sovereignty (agents run locally) and parallelizes the workload.

Design Rationale 2: Stigmergic Routing vs. Negotiation. To address *Heterogeneity* and *Low Friction*, we avoid complex negotiation protocols. Teams cannot be forced to implement complex agent schemas. Instead, agents collaborate indirectly via a shared "Global Routing Table" (a stigma), which records successful paths. This allows even "dumb" agents to participate by simply following the tabular hints, lowering the barrier to entry.

Design Rationale 3: Local Enrichment, Global Statistics. We decouple enrichment (local, expensive, LLM-heavy) from routing (global, cheap, statistical). This ensures that the global routing layer remains lightweight and fast, while the heavy lifting of understanding data happens only where the data resides.

3.2 Overview of *Comfey*

As shown in Figure 3, instead of relying on a single global agent, *Comfey* delegates triage responsibilities to a set of *team-scoped agents*, each tailored to a specific service. Each agent is instantiated through the *Comfey* framework, which provides a common workflow and interface for all teams. Teams can enrich their agents with domain-specific data and knowledge. When an incident occurs, it is initially assigned to the team where the error signal was captured or where users reported the affected software. The team's agent then determines whether to accept or reject the incident. If the agent *accepts* the incident, it provides diagnostic information and mitigation suggestions based on its enriched domain data. If the agent *rejects* the incident, it collaborates with other agents to identify the next most likely team to forward it to.

Figure 4 shows the workflow for each team-scoped agent. The agent first enriches the new incident through a *data enhancement module* that collects incident data maintained within the team and merges it with the original incident report. The agent then initiates a *decision module* that evaluates the incoming incident against expert knowledge, including historically triaged incidents and troubleshooting guides. If the agent finds a troubleshooting guide indicating that the incident belongs to its team, or a previously accepted incident that is similar to the new one, it *accepts* the incident. Otherwise, it *rejects* the incident and forwards it to the next most likely team. To adapt to changes in the cloud system , *Comfey* supports human-in-the-loop, allowing continuous refinement of both the enrichment and matching pipelines to improve accuracy over time.

3.3 Enriching Incidents with Distilled Data

The initial incident report is generated from monitors or user reports, which typically contain only high-level symptoms and descriptions that are insufficient for accurate triage. Aggregating raw data across teams, however, is both costly and noisy. Incidents in cloud environments exhibit diverse symptoms and root causes, and the monitoring data stored within each team displays varied patterns and semantics. Extracting meaningful information from each team's incident data requires domain-specific knowledge. The key insight is that most teams monitor and manage incident data through a small number of monitoring platforms in Azure, and engineers often write custom scripts or pipelines to manually extract raw data for each incident. Thus, *Comfey* provides a flexible interface that allows teams to register their existing scripts, which are used by *Comfey*'s enrichment pipeline to automatically collect raw data that the team already leverages.

Comfey implements built-in enrichment interfaces for the leading monitoring platforms used in Azure. Through the *Comfey* common interface, engineers can provide simple templates that instruct the pipeline on the type of data required. Incident data can include stack traces, error logs, time series metrics, recent change records, and resource or node health information, accessible via existing APIs or query endpoints. For example, in the incident shown in Figure 1, a storage-scoped agent can enrich the report with recent decommission events for the referenced nodes, which directly point to the underlying cause. The storage team achieves this by simply adding a Kusto query to retrieve node-level decommission records from their internal tables. *Comfey* has a set of default enrichment pipelines to collect general incident data such as log, stack traces, and engineer discussion threads.

Because an incident may be transferred across multiple teams, *Comfey* keeps all enrichment data collected by each agent and passes it to subsequent agents. This design allows later agents to work with a more complete view of the incident. Each agent encapsulates its raw incident data, evaluation results, and the rationale for rejecting the incident as structured attachments and attaches them to the incident report. Thus, later agents can reuse the previously collected data and trace the full transfer flow of the incident.

3.4 Semantic Distillation

The raw data extracted from the enrichment pipeline can be noisy and redundant. For example, one key data source for each incident is the engineers' discussion thread, which often contains critical contextual information. However, these discussions may include incorrect arguments, inconsistent descriptions across engineers, or duplicated statements across related incidents. Feeding such noisy data directly to the triage agent can mislead its reasoning and result in incorrect decisions. To address this, *Comfey* designs a *semantic distillation* phase within each team-scoped agent to eliminate noise and transform the raw incident data into a structured and semantically aligned representation.

The *semantic distillation* phase applies a summarization agent to compress the enriched incident data into a concise summary that explicitly captures the anomaly, the teams involved, and the rationale behind the team transfers. For example, the summary

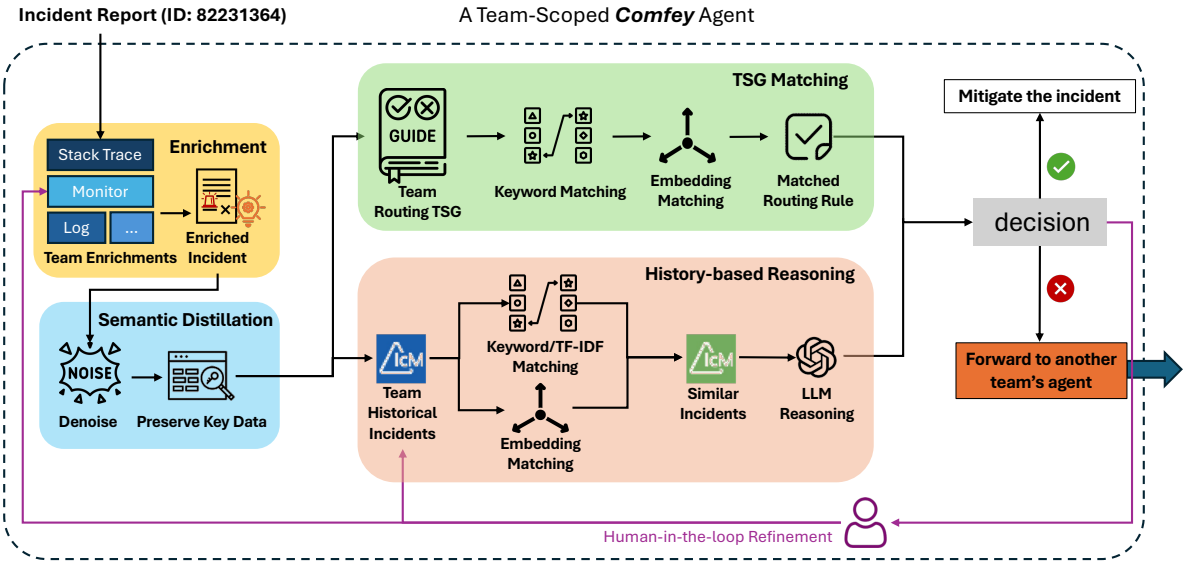


Figure 4: Workflow of *Comfey*'s team-scoped agent

can contain the sentence: 'messages containing "low node availability" are consistently followed by transfers to the storage team.' The distillation process first filters out information irrelevant to triage, such as boilerplate text or automated messages generated by monitoring tools. Then, it adds timestamps and author information to each discussion. The intuition is that we observe that the first ten minutes of incident discussions often contain the most crucial signals, including user-facing failure symptoms or abnormal metrics detected by monitors. Thus, adding the timing and author information could help the agent find the most important information. Finally, the data is passed to the summarization agent, which produces a structured summary defined by *Comfey*.

The semantic distillation pipeline operates in three steps using an LLM. First, it performs *Context Cleanup*, removing boilerplate (e.g., standard automated alerts) and non-informative text. Second, it extracts structural metadata, such as timestamps and author roles, to prioritize signals from the first ten minutes of an incident—a window we found to contain the most critical failure symptoms. Finally, the *Summarization Agent* generates a structured summary highlighting the anomaly, participating teams, and transfer rationale. This distilled form is used for both the current incident and as a canonical representation for historical cases, significantly reducing noise during matching.

3.5 Making Triage Decision

After enriching the incident, the team-scoped agent determines whether to accept the incident. We design this decision process to be *evidence-based* and *auditable*, utilizing the two primary assets teams already possess: (1) **Troubleshooting Guides (TSGs)**, which are concrete, deterministic workflows used daily by engineers to triage issues, and (2) **Historical Incidents**, which serve as a record of past triage patterns.

Algorithm 1: Agent Decision Logic

Input : Incident I, Enriched Data E, Team TSG T, Historical Incidents H
Output: Decision $D \in \{Accept, Reject\}$, Next Team N (if Rejected)

```
// Step 1: TSG Matching
foreach Rule r ∈ T do
    if Match(I, E, r) then
        return Accept, None
// Step 2: Historical Matching
Candidates ← Retrieve(H, I, E) ;          // TF-IDF + Embedding
Similar ← Filter(Candidates, threshold)
if Similar ≠ ∅ then
    Decision ← RS(I, Similar) ;           // Reasoning Service
    if Decision == Accept then
        return Accept, None
// Step 3: Reject and Route
N ← GlobalRouting(I) ;
return Reject, N
```

By anchoring decisions in these existing assets, *Comfey* achieves three goals outlined in our design rationale. First, it *leverages existing data* to avoid the "cold start" problem, enabling agents to be deployed without creating new labeled datasets (Design Rationale 2: Low Friction). Second, it ensures *ease of development*, as engineers simply point the agent to their existing documentation and ticket queues. Finally, it provides *auditability* (Design Rationale 1), as every decision is explicitly traced back to a specific TSG rule or a historical precedent, allowing engineers to verify exactly why an agent accepted a ticket.

The agent executes the decision logic described in Algorithm 1. It first checks the team's TSG. If a TSG explicitly covers the issue (e.g., 'accept if error code is X'), the agent accepts immediately. TSG rules are evaluated using a combination of keyword matching and semantic similarity. If no TSG rule matches, the agent retrieves similar historical incidents using a two-stage retrieval (TF-IDF followed by embedding-based cosine similarity). The agent then uses

an LLM-based Reasoning Service to compare the current incident with retrieved historical precedents. If highly similar cases were previously accepted by this team, it accepts. Otherwise, it rejects the incident.

3.6 Incident Routing Mechanism

When an agent rejects an incident, the system must determine the next destination. In a decentralized environment, no single agent has a global view. *Comfey* addresses this challenge by implementing a *stigmergic* coordination mechanism: a shared "Global Routing Table" that all agents consult and update.

3.6.1 Constructing the Global Routing Table. The Global Routing Table is not a static map but a dynamic, probabilistic record of team interactions. It is constructed on-the-fly by aggregating the *successful triage paths* of historical incidents relevant to the current case. When an agent processes an incident, it identifies similar historical cases and uses their resolution paths to update the table's statistics. Formally, for every incident i that was successfully accepted by team T_{final} after traversing a path $P_i = [T_{start}, \ldots, T_{final}]$, *Comfey* records the association between the incident's semantic signature $S(i)$ and the final owner T_{final}. The table stores these associations as a frequency map:

$$Table(Signature) \rightarrow \{(Team_A, count_A), (Team_B, count_B), \ldots\}$$

To avoid storage explosion, the signature $S(i)$ is the distilled summary generated by the Semantic Distillation phase (subsection 3.4).

Robustness against Local Failure: This statistical approach ensures that the "next plausible owner" is derived from global likelihoods rather than the reasoning of any single agent. Since agents update the table using concrete, verifiable incident IDs (paths), a single low-quality agent cannot undermine the system with hallucinated or inconsistent reasoning.

3.6.2 Next Candidate Selection. When an agent A rejects an incident I, it first checks its Troubleshooting Guides (TSGs). If a TSG rule matches, *Comfey trusts the TSG's recommendation directly*, as TSGs are created for high-frequency, well-understood patterns. If no TSG matches, the agent queries the Global Routing Table with I's distilled summary. The table returns a ranked list of potential teams based on historical frequency. $Candidates = Table.Query(S(I))$ The network routing logic then filters this list:

- **Loop Prevention:** By default, any team in the current traversal path $P_{current}$ is filtered out to prevent cycles.
- **Selection:** The highest-ranked remaining team is selected as the next hop.

This "Statistical Routing" is $O(1)$ and highly scalable, serving as the default fallback when explicit TSG rules are absent.

3.6.3 Handling Failures and Edge Cases. While statistical routing handles the majority of cases (> 90%), edge cases exist.

- **Mass/Unanimous Rejection:** To prevent infinite routing, we enforce a strict hop limit (default 10). If an incident is rejected by 10 teams, it is marked as "Unresolved" and routed to a designated human "Investigation Team" (or Triage DRI) to ensure no incident is dropped.
- **Looped Routing:** In tightly coupled service architectures, legitimate routing loops may occur (e.g., Team A → Team B → Team A) as teams coordinate to piece together a full diagnosis. While Statistical Routing avoids loops by default, *Comfey* allows configurable "max visits" per team (typically 1, but adjustable) to support these complex flows. Note that such loops primarily emerge from explicit TSG-based routing, where the loop is an intentional part of the diagnostic workflow. In our evaluation (Section 4.3), we find that such multi-turn loops account for less than 5% of cases.
- **Incorrect TSG Rules:** Since *Comfey* treats TSGs as high-priority ground truth, an incorrect rule can lead to sustained mis-routing. We accept this risk as a trade-off for auditability. In practice, such errors create visible spikes in "triage latency" for the target team, triggering organizational alerts that pressure the owning team to correct the faulty TSG rule rapidly.

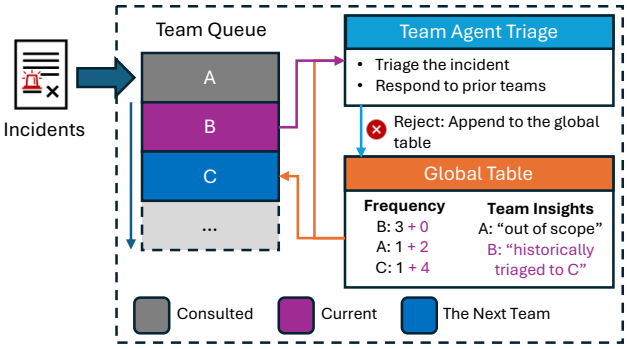


Figure 5: *Comfey*'s Multi-agent Workflow to Triage An Incident

3.7 Human-in-the-Loop

As cloud systems evolve, *Comfey* agents must adapt accordingly. Once deployed, *Comfey* allows engineers to label each incident with ground-truth outcomes, which are recorded and reused to improve future triage. The system also refreshes its historical incident database periodically to capture emerging issues. Engineers can further update the Troubleshooting Guide (TSG) by inserting or refining routing rules based on *Comfey*'s decisions, ensuring alignment with evolving domain expertise.

3.8 Self-Onboarding Platform

With hundreds of teams across Azure, a key challenge for *Comfey* is achieving scalable deployment. To address this, we developed a self-onboarding platform that enables domain experts to locally configure and launch their own team-scoped agents with minimal effort. By installing *Comfey* as a package and running fewer than 10 CLI commands, teams can build and deploy agents on a secure platform within Azure.

Comfey offers a configuration interface to enable common types of enrichment based on our experience with production teams. Engineers can, for example, define reusable query templates in this interface. *Comfey* also supports integration with Azure's internal observability systems and exposes a general API interface for connecting to custom telemetry backends. *Comfey* also includes a push-button utility to retrieve historical incidents and troubleshoot guidance related to the team. By default, it only fetches recent

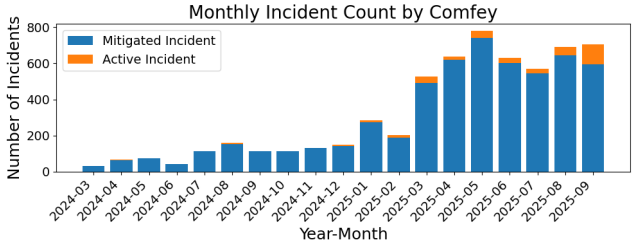


Figure 6: Monthly Incident Count by *Comfey*

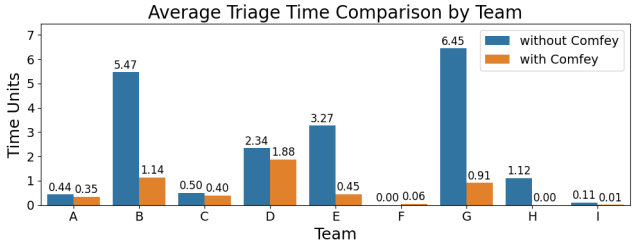


Figure 7: Triage Time Comparison By Team

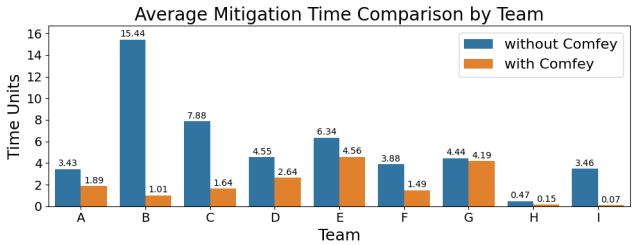


Figure 8: Mitigation Time Comparison By Team

high-severity cases, though engineers can apply additional filters to enhance signal quality.

Importantly, teams can begin with just historical incidents, requiring almost no manual effort, and incrementally improve routing quality by later adding enrichment hooks and TSGs. Once submitted, *Comfey* deploys the team-scoped agent to Azure Kubernetes Service for automatic scaling and fault tolerance.

4 Experiments

In this evaluation, we answer the following research questions.

(1) How is *Comfey*'s agent integrated into each team?
(2) How efficiently does *Comfey* triage cloud incidents?
(3) Can *Comfey* help engineers mitigate incidents?
(4) How accurate is *Comfey*'s triage in assigning incidents to the correct teams?
(5) How does *Comfey* perform compared to state-of-the-art automated triage approaches?

4.1 Deployment Status and Scale

Comfey has been running in production in Microsoft Azure since early 2024. It is deployed across fifteen different engineering teams and covers tens of services spanning compute, control, storage, and networking components. In total, *Comfey* monitors millions of hosts and has triaged approximately 19,500 newly emerged incidents to date. At the same time, *Comfey* analyzes ~75,000 historical incidents to match the new incidents. Figure 6 shows the incident *Comfey* reported in Azure from March 2024 to Jan 2026. Overall, *Comfey* triages ~885 incidents on average per month, and engineers mitigate ~785 incidents on average.

In Azure, there are various existing solutions in incident triage, so only complex and ambiguous incidents are escalated to *Comfey*. Typically, the initial team that receives an incident can independently investigate and mitigate it using their monitor data and documentation. Engineers may also follow troubleshooting guides (TSGs) to manually route the incident to another team or coordinate directly with other teams. Only when incidents are too complex for manual triage do they escalate to *Comfey*. In this workflow, *Comfey* serves as the last defense for production incidents that cannot be resolved through standard procedures.

4.2 Integration Effort

Integrating *Comfey* into a new team's workflow is designed to be lightweight and non-intrusive. Engineers do not need to modify existing codebases; instead, they complete setup via a self-onboarding platform with roughly ten command-line instructions. Based on user feedback and deployment data, new users typically spend about two hours becoming familiar with the platform. Once familiar, executing the setup commands takes only a few minutes.

In addition to enabling the agent, engineers can further customize the *Comfey* agent by writing troubleshooting guides and enriching their incident data. Translating existing routing rules into *Comfey*'s TSG format typically takes around five hours, while using the *Comfey* interface to enrich the agent with domain-specific incident data takes about two hours. This relatively low integration cost is partially because most teams already maintain scripts for extracting and analyzing their own data; they only need to provide templates for *Comfey* to use. For example, when integrating with the Kusto monitoring platform, engineers simply adapt their existing Kusto queries from their conventional triage workflows to *Comfey* by templatizing the queries and specifying parameters such as node IDs and container IDs.

4.3 Effectiveness of Incident Triage

To evaluate the effectiveness of *Comfey*, we measure the amount of triage time it saves. Triage time is defined as the duration from when an incident ticket is created to when it is finally transferred to the correct team. We collect approximately 2,400 incidents that occurred before the deployment of *Comfey* and calculate their triage times as a baseline. We also analyze around 15,700 incidents processed by *Comfey* over the past six months to assess its impact.

Figure 7 shows the average triage time across the fifteen teams where *Comfey* was deployed. Every team except team F reduced its average triage time, because team F used to allow many non-actionable tickets into their queue and ended up auto-mitigating them. With *Comfey*, these tickets are being rejected and transferred out, therefore delaying triage time. On average, *Comfey* achieves a 6× speedup compared to manual triage. This improvement primarily comes from *Comfey*'s automated operation, as each agent can analyze incidents faster than engineers. In some cases, *Comfey* takes slightly longer than manual triage because it explores additional intermediate teams during routing before reaching the correct one.

A key observation is that *Comfey*'s decentralized and parallel design offers significant advantages during incident bursts. Since agents operate independently, multiple incidents can be processed concurrently using the Azure Kubernetes backend, whereas on-call engineers must handle incidents sequentially. In production incident bursts, *Comfey* substantially reduces the average and tail triage time compared to manual processes.

4.4 Effectiveness of Mitigation

In addition to improving the speed of triage, *Comfey* also helps engineers diagnose and mitigate incidents through its enriched contextual data. Moreover, *Comfey* is integrated with Azure's auto-mitigation framework. When the agent detects that a troubleshooting rule includes a link to an automated mitigation procedure, it can directly trigger or redirect the incident to the corresponding mitigation engine.

We evaluated the mitigation time using the same set of incidents discussed in Section 4.3. Mitigation time is defined as the duration from when an incident is transferred to the correct mitigating team to when the issue is fully resolved. Figure 8 shows the comparison between incidents handled manually and those processed by *Comfey* in the past two months. On average, *Comfey* achieves a 4.3× speedup in mitigation time. This improvement stems from auto-mitigation and automated enrichment, which reduces diagnostic overhead and integration with existing mitigation workflows.

4.5 *Comfey* Accuracy

Evaluating the accuracy of incident triage is challenging because collecting reliable ground truth labels is difficult. In *Comfey*'s deployment, ground truth labels are provided by on-call engineers through two complementary mechanisms. Due to the high incident volume, engineers are not required to explicitly label every case. Instead, they are instructed to carefully clean up their incident queues and ensure that transfers reflect the true ownership of each incident, which is part of their standard on-call workflow and requires no additional effort. The ground truth for each incident is inferred from whether the final owning team matches the team represented by the *Comfey* agent. In rare cases where this inference is incorrect, engineers can manually tag incidents to override the labels.

We collected approximately 16,000 incidents with verified ground truth labels over a period of 1 year and 10 months for evaluation. We evaluate the accuracy of *Comfey*'s triage decisions using standard classification metrics, including accuracy, precision, recall, and F_1 score. In our formulation, a positive case refers to the agent's decision to reject an incident, while a negative case refers to the decision to accept it. Accordingly, a true positive (TP) occurs when the agent correctly rejects an incident that should be rejected, and a false positive (FP) occurs when the agent rejects an incident that should have been accepted. Conversely, a true negative (TN) represents an incident correctly accepted, and a false negative (FN) represents an incident incorrectly accepted. Based on these definitions, *Comfey* achieves an average of 91.5% accuracy, 64.3% precision, 62.1% recall, and 63.1% F_1 score across all evaluated incidents.

4.6 Effectiveness of Components (Ablation Study)

To understand the contribution of each component to *Comfey*'s overall accuracy, we conducted an offline ablation study on ~1,300 incidents randomly sampled from the past three months. We re-ran these incidents through the *Comfey* framework while selectively disabling specific components: Semantic Distillation, TSG Matching, and the Global Routing Table.

- **w/o Semantic Distillation**: Disabling distillation did not significantly impact accuracy but notably increased triage latency by **40.8%**. This confirms that distilling noisy logs into concise summaries is critical for efficiency and reducing LLM context processing time.
- **w/o TSG Matching**: Removing the TSG matching stage reduced triage accuracy by **12.33%**. This highlights the importance of leveraging explicit, team-authored routing rules for high-precision decision making.
- **w/o Global Routing Table**: Replacing the history-based Global Routing Table with a naive agent negotiation mechanism (where agents randomly query neighbors) reduced accuracy by **40%**. This demonstrates that the shared routing table is the primary driver of successful cross-team collaboration, providing a stable and empirically grounded routing path.

4.7 Reliability and Failure Modes

While *Comfey* achieves high accuracy, we analyzed failure modes to understand system behavior in edge cases. A rare failure mode is *unanimous rejection*, where an incident traverses a chain of agents without being accepted. To mitigate this, organizations designate "investigation teams" that serve as a fallback. In our deployment, if an incident is rejected by 5 consecutive teams (a configurable depth limit), it is automatically routed to a human on-call engineer or a general investigation queue. This ensures that no incident is dropped, prioritizing safety and eventual resolution over pure automation in highly ambiguous scenarios.

4.8 Comparison with State-of-the-Art

Before the deployment of *Comfey*, Azure used COMET [33] for automated incident triage. COMET employs an AutoExtractor to filter non-critical logs and leverages a large language model (LLM) for keyword extraction. We compared incidents triaged by COMET with those handled by *Comfey*. We observed that *Comfey* improved triage accuracy by approximately 7.55%, reduced the average triage time by 4.38 times, and reduced the average mitigation time by 2.91 times. The similar accuracy between the two systems can be attributed to their shared use of LLMs for information extraction. However, *Comfey* achieves significantly lower triage and mitigation times due to its decentralized architecture and enriched data collection.

Comparison with Multi-Agent Frameworks. Recent academic works, such as Triangle (ASE '25), propose multi-agent frameworks that rely on complex negotiation protocols and assume high-quality agents across all teams uniformly. In contrast, *Comfey* is designed for the reality of large-scale industrial environments where team

maturity varies. Comfey's "low friction" design allows teams to start with a minimal agent based solely on historical data (zero manual effort) and incrementally improve it. Instead of relying on real-time negotiation, which can be unstable when agents have varying capabilities, Comfey uses the Global Routing Table to ensure robust collaboration. This design choice makes *Comfey* more practical for organization-wide deployment where "heterogeneous agent quality" is the norm.

4.9 Runtime Cost

The *Comfey* agents use a combination of `gpt-4.1`, `gpt-4o-mini`, and `text-embedding-ada-002` for reasoning, summarization, and embedding generation. The total cost of operating *Comfey* consists of two components. First, historical incident processing: when setting up a *Comfey* agent locally, on-call engineers must download and index historical incidents. Processing approximately 1,000 incidents consumes about 265,260 tokens, costing around $3.85. This is primarily a one-time setup cost, although the historical incident database is periodically refreshed to include newly reported incidents. Second, new incident triage: after the agents are deployed in production, each new triage call consumes about 5,804 tokens and costs roughly $0.02. These values are based on an average calculated over approximately 500 triage calls.

5 Related Work

Incident Triage. Prior work has explored automated incident triage using centralized deep learning models. **DeepCT** [5] and **DeepTriage** [25] apply RNNs or transformers to structured incident data for team prediction, but require extensive labeled data and retraining to adapt to new teams. **CONAN** [18] retrieves similar historical cases for batch triage, but assumes incident recurrence and lacks human-in-the-loop refinement. **COMET** [33] uses LLMs for root cause analysis and operator suggestions, but remains centralized and difficult to customize for team-specific workflows.

Comfey differs fundamentally in its decentralized, team-scoped architecture. We position *Comfey* within the broader spectrum of Multi-Agent System (MAS) coordination strategies.

- **vs. Contract Net / Negotiation:** Approaches like Triangle (ASE '25) or classic Contract Net protocols [30] rely on real-time negotiation or auctions. These offer high flexibility but incur high communication costs ($O(N^2)$ or similar) and require all agents to speak a sophisticated common language. In our heterogeneous environment, this is impractical.
- **vs. Blackboard / Stigmergy:** *Comfey* aligns more closely with blackboard systems [23] or stigmergic coordination [31]. The Global Routing Table acts as a shared memory (blackboard), observing the "trace" of successful incident resolutions. Agents interact with this shared memory rather than directly negotiating with each other. This decoupling allows the system to tolerate immense heterogeneity in agent quality—simple agents contribute to and benefit from the routing table just as well as complex ones, without needing to synchronize protocols.
- **vs. LLM-based Agents (MetaGPT / ChatDev):** Recent LLM-powered multi-agent frameworks like MetaGPT [12] and ChatDev [26] assign specialized roles (e.g., Coder, Tester) to agents who collaborate via natural language reasoning. While powerful for closed-domain tasks, these systems typically rely on a "Reasoning Routing" model where agents discuss and agree on the next step. In contrast, *Comfey* employs "Statistical Routing". Given the massive scale of incident types, relying on LLM contextual reasoning to route every ticket is cost-prohibitive and slow. By leveraging the Global Routing Table (statistics of past success), *Comfey* bypasses the need for complex inter-agent reasoning for the majority of routing decisions, reserving expensive LLM calls for local enrichment and summarization only.

This "collaboration via history" approach allows *Comfey* to scale to hundreds of agents with minimal overhead and tolerance for heterogeneity in agent quality, trading the precision of real-time negotiation for the robustness of statistical routing.

Root Cause Analysis. A large body of research has focused on identifying the root causes of system incidents to aid debugging and recovery. These efforts typically rely on analyzing system logs [6], performance metrics [19], or traces [1, 7, 8, 15, 21] to infer the sequence of events leading to a failure. These techniques surface anomalies or reconstruct causal paths to reduce debugging effort. However, many production issues arise from cross-service interactions or misconfigurations without a clear root cause. Existing tools may surface correlated symptoms, but cannot always determine team ownership. As a result, incident triage, which assigns responsibility to the right team, remains critical even when the precise cause is unknown.

6 Conclusion

We present *Comfey*, a decentralized framework for automated incident triage in large-scale cloud systems. *Comfey* deploys team-scoped agents that analyze domain-specific data and collaborate via a shared routing mechanism to identify the responsible team. It combines data enrichment, semantic distillation, and lightweight integration to enable accurate triage without centralized observability or intrusive changes. Deployed in Azure for over a year, *Comfey* triaged 19,500 incidents with 91.5% accuracy, reducing triage time by 6 times and mitigation time by 4.3 times. These results highlight the effectiveness of decentralized agent-based triage at cloud scale.

References

[1] Mona Attariyan, Michael Chow, and Jason Flinn. 2012. X-ray: automating root-cause diagnosis of performance anomalies in production software. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, USA, 307–320.

[2] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems* (1st ed.). O'Reilly Media, Inc.

[3] Amber Carlson. 2025. *AWS down: How a single network outage rippled through businesses, institutions and the economy.* CU Boulder Today, University of Colorado Boulder. https://www.colorado.edu/today/2025/10/22/aws-down-how-single-network-outage-rippled-through-businesses-institutions-and-economy

[4] 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)*. 111–120. doi:10.1109/ICSE-SEIP.2019.00020

[5] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. Continuous incident triage for large-scale online service systems. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 364–375. doi:10.1109/ASE.2019.00042

[6] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 674–688. doi:10.1145/3627703.3629553

[7] Michael Chow, David Meisner, Jason Flinn, Daniel Peek, and Thomas F. Wenisch. 2014. The Mystery Machine: End-to-end Performance Analysis of Large-scale Internet Services. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 217–231. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chow

[8] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: Practical & Scalable ML-Driven Performance Debugging in Microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 135–151. doi:10.1145/3445814.3446700

[9] 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* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 126–141. doi:10.1145/3542929.3563482

[10] Drishti Goel, Fiza Husain, Aditya Singh, Supriyo Ghosh, Anjaly Parayil, Chetan Bansal, Xuchao Zhang, and Saravan Rajmohan. 2024. X-lifecycle learning for cloud incident management using llms. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 417–428.

[11] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. 2016. Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/2987550.2987583

[12] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI] https://arxiv.org/abs/2308.00352

[13] Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. 2014. Effective Bug Triage Based on Historical Bug-Fix Information. In *Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering (ISSRE '14)*. IEEE Computer Society, USA, 122–132. doi:10.1109/ISSRE.2014.17

[14] Junjie Huang, Jinyang Liu, Zhuangbin Chen, Zhihan Jiang, Yichen Li, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R. Lyu. 2024. FaultProfIT: Hierarchical Fault Profiling of Incident Tickets in Large-scale Cloud Systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (Lisbon, Portugal) *(ICSE-SEIP '24)*. Association for Computing Machinery, New York, NY, USA, 392–404. doi:10.1145/3639477.3639754

[15] Yuxuan Jiang, Ziming Zhou, Boyu Xu, Beijie Liu, Runhui Xu, and Peng Huang. 2025. Training with Confidence: Catching Silent Errors in Deep Learning Training with Automated Proactive Checks. In *Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI '25)*. USENIX Association, Boston, MA, USA.

[16] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. 2016. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Softw. Engg.* 21, 4 (Aug. 2016), 1533–1578. doi:10.1007/s10664-015-9401-9

[17] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 926–931. doi:10.1145/3106237.3117776

[18] Liqun Li, Xu Zhang, Shilin He, Yu Kang, Hongyu Zhang, Minghua Ma, Yingnong Dang, Zhangwei Xu, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. CONAN: Diagnosing Batch Failures for Cloud Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 138–149. doi:10.1109/ICSE-SEIP58684.2023.00018

[19] Chang Lou, Cong Chen, Peng Huang, Yingnong Dang, Si Qin, Xinsheng Yang, Xukun Li, Qingwei Lin, and Murali Chintalapati. 2022. RESIN: A Holistic Service for Dealing with Memory Leaks in Production Cloud Infrastructure. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*. USENIX Association, Carlsbad, CA, USA, 109–125. https://www.usenix.org/conference/osdi22/presentation/lou-resin

[20] Chang Lou, Peng Huang, and Scott Smith. 2020. Understanding, detecting and localizing partial failures in large system software. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 559–574.

[21] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. 2015. Pivot tracing: dynamic causal monitoring for distributed systems. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 378–393. doi:10.1145/2815400.2815415

[22] Hoda Naguib, Nitesh Narayan, Bernd Brügge, and Dina Helal. 2013. Bug report assignee recommendation using activity profiles. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, 22–30.

[23] H. Penny Nii. 1986. PART ONE: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Mag.* 7, 2 (June 1986), 38–53. doi:10.1609/aimag.v7i2.537

[24] Gunjan Pathak and Monika Singh. 2023. A Review of Cloud Microservices Architecture for Modern Applications. In *2023 World Conference on Communication & Computing (WCONF)*. 1–7. doi:10.1109/WCONF58270.2023.10235199

[25] Phuong Pham, Vivek Jain, Lukas Dauterman, Justin Ormont, and Navendu Jain. 2020. DeepTriage: Automated Transfer Assistance for Incidents in Cloud Services. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3281–3289. doi:10.1145/3394486.3403380

[26] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. arXiv:2307.07924 [cs.SE] https://arxiv.org/abs/2307.07924

[27] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring llm-based agents for root cause analysis. In *Companion proceedings of the 32nd ACM international conference on the foundations of software engineering*. 208–219.

[28] 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 *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA) *(NSDI'11)*. USENIX Association, USA, 43–56.

[29] Alexei Scerbakov, Martin Ebner, and Nikolai Scerbakov. 2015. Using cloud services in a modern learning management system. *Journal of computing and information technology* 23, 1 (2015), 75–86.

[30] Reid G. Smith. 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers* C-29, 12 (1980), 1104–1113. doi:10.1109/TC.1980.1675516

[31] Guy Theraulaz and Eric Bonabeau. 1999. A Brief History of Stigmergy. *Artificial life* 5 (02 1999), 97–116. doi:10.1162/106454699568700

[32] Sai Prasad Veluru and Mohan Krishna Manchala. 2024. Using LLMs as Incident Prevention Copilots in Cloud Infrastructure. *International Journal of AI, BigData, Computational and Management Studies* 5, 4 (2024), 51–60.

[33] Zexin Wang, Jianhui Li, Minghua Ma, Ze Li, Yu Kang, Chaoyun Zhang, Chetan Bansal, Murali Chintalapati, Saravan Rajmohan, Qingwei Lin, Dongmei Zhang, Changhua Pei, and Gaogang Xie. 2024. Large Language Models Can Provide Accurate and Interpretable Incident Triage. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. 523–534. doi:10.1109/ISSRE62328.2024.00056

[34] 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.

[35] Yifan Wu, Bingxu Chai, Ying Li, Bingchang Liu, Jianguo Li, Yong Yang, and Wei Jiang. 2023. An Empirical Study on Change-induced Incidents of Online Service Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 234–245. doi:10.1109/ICSE-SEIP58684.2023.00027

[36] Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving Automated Bug Triaging with Specialized Topic Model. *IEEE Trans. Softw. Eng.* 43, 3 (March 2017), 272–297. doi:10.1109/TSE.2016.2576454

[37] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. 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* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 38–49. doi:10.1145/3663529.3663826

[38] 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* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 266–277. doi:10.1145/3663529.3663846