Automatic Root Cause Analysis via Large Language Models for Cloud Incidents

Yinfang Chen$^{\diamond\S}$, Huaibing Xie$^{\diamond\P}$, Minghua Ma$^{\triangle*}$, 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*, Tianyin Xu§

Microsoft*, University of Illinois at Urbana-Champaign§, Peking University¶
Huazhong University of Science and Technology†, National University of Singapore‡

Abstract

Ensuring the reliability and availability of cloud services necessitates efficient root cause analysis (RCA) for cloud incidents. Traditional RCA methods, which rely on manual investigations of data sources such as logs and traces, are often laborious, error-prone, and challenging for on-call engineers. In this paper, we introduce RCACOPILOT, an innovative *on-call system empowered by the large language model* for automating RCA of cloud incidents. RCACOPILOT matches incoming incidents to corresponding incident handlers based on their alert types, aggregates the critical runtime diagnostic information, predicts the incident's root cause category, and provides an explanatory narrative. We evaluate RCACOPILOT using a real-world dataset consisting of a year's worth of incidents from Microsoft. Our evaluation demonstrates that RCACOPILOT achieves RCA accuracy up to 0.766. Furthermore, the diagnostic information collection component of RCACOPILOT has been successfully in use at Microsoft for over four years.

CCS Concepts: • **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → **Maintaining software**.

Keywords: Root Cause Analysis, Large Language Models, Cloud Systems

---

$\diamond$ This research was primarily conducted during an internship at Microsoft Research Asia.
$\triangle$ Minghua Ma is the corresponding author.

---

1 Introduction

Cloud computing serves as an indispensable infrastructure for numerous applications and services upon which people rely daily. As the adoption of cloud services continues to grow, ensuring their reliability, availability, and security becomes increasingly vital [12, 17, 24, 32, 36, 43, 44, 47]. However, the complexity of cloud systems makes them vulnerable to a variety of incidents that could pose significant challenges to these crucial properties [54]. A typical incident life-cycle consists of four stages: (1) *Detection* [37, 51, 52]: When an anomalous system behavior is observed, an alert is raised by monitors or users of the service (internal engineers or external customers). (2) *Triaging* [4, 8, 9]: After the detection, the incident is assigned to the appropriate engineering team after an initial assessment. (3) *Diagnosis* [34]: Assigned on-call engineers (OCEs) inspect different aspects of the incident and have several rounds of back-and-forth communication to identify the root cause. (4) *Mitigation* [1, 20]: Several actions are taken by OCEs to mitigate the incident and to restore service health.

Root cause analysis (RCA) is pivotal in promptly and effectively addressing these incidents. By accurately diagnosing the underlying problem and preventing its recurrence, RCA not only restores service availability swiftly but also fortifies the overall reliability of cloud services. However, identifying the root causes of these incidents often represents a daunting and time-consuming task that requires significant human expertise and intervention [36].

Traditional approaches to cloud incident RCA typically involve the manual collection and analysis of various types of data, such as logs [18, 19, 27, 31, 56], metrics [14, 38, 49], traces [53, 61], and incident tickets [20, 42]. This manual process is not only laborious and error-prone, but can also be challenging due to varying levels of available information - what we term as the "information spectrum". The "information spectrum" describes a continuum of information availability, ranging from situations with too little information to those inundated with an excess. At either end of this spectrum, RCA can become particularly challenging. The relevant information for RCA might be buried within the voluminous data, leading to an information overload for OCEs. OCEs may find it challenging to quickly pinpoint the

relevant information amidst the sea of data, hindering efficient incident resolution. Conversely, OCEs also encounter situations where they lack the necessary information to understand and address the root causes of incidents accurately. Beyond these challenges, the collected data itself is often noisy, incomplete and inconsistent, further complicating the RCA process.

Specifically, the engineering team documents the frequent troubleshooting steps in the form of troubleshooting guides (TSGs) to facilitate the handling of future incidents. However, the volume of TSGs is overwhelming for OCEs, making the search for the most relevant guide a time-consuming task that might cause system downtime. Moreover, TSGs struggle to keep pace with the ever-evolving nature of cloud systems, thus often falling short when new incident types emerge. Even when a relevant TSG is located, it may not cover all the intricacies of the specific incident. This could be due to variations in system configurations, the presence of multiple interacting root causes, or previously unknown issues.

At the heart of RCA lies the fundamental challenge of *efficiently collecting and interpreting comprehensive, incident-specific data* within a limited time frame. OCEs must quickly discern the relevance of various data types to the incident at hand and interpret them correctly. However, the complexity and sheer volume of data generated by cloud systems often impede rapid decision-making. Furthermore, the expertise required to analyze various data types, along with the diverse range of possible incident causes, exacerbates the difficulty of the task. As a result, OCEs may spend an inordinate amount of time analyzing data and formulating hypotheses, detracting from time that could be better spent resolving the incident and restoring system functionality.

Data-driven and Artificial Intelligence (AI) techniques have been leveraged for automating the incident management [9, 10]. While there are existing techniques that recommends relevant TSGs [20] and automates the workflows [42] of TSGs, their utility is limited by the inherent challenges associated with TSGs. Despite these automated processes, OCEs still find themselves investing significant manual effort in sifting through the vast amounts of information, interpreting the data, and identifying the root causes of incidents.

The recent advent and success of large language models (LLMs) in performing complex tasks [21, 28, 46], suggests a promising avenue for enhancing RCA. Specifically, LLMs can be used to parse through high-volume data, discern relevant information, and produce succinct, insightful outputs. This significantly alleviates the burden on OCEs to manually sift through vast amounts of data, helping them focus on resolving the incident more quickly and effectively. Additionally, LLMs can adapt to new and evolving types of incidents, learning from previous data to improve future predictions. While LLMs can process and generate text efficiently, they lack intrinsic domain-specific knowledge, especially in specialized areas such as cloud incident management. This lack of understanding of specific contexts, such as cloud incidents, can limit their accuracy in predicting incident root causes and generating appropriate explanations.

Recently, Ahmed *et. al.* [1] proposed to finetune a LLMs with domain-specific datasets for generating root causes of an incident just by leveraging the title and summary information available at the time of incident creation. While they have demonstrated promises of LLMs in incident root causing, finetuning has several limitations: (1) As accurate RCA requires various sources of complex unstructured or semi-structured data (e.g., logs, telemetry, traces, and natural language description), just using a generic title and summary might miss useful signals to reach conclusive diagnosis details; (2) finetuning is costly and may require a huge volume of training samples; (3) it is challenging to continuously update a finetuned GPT model with evolving nature and scope of incidents; therefore such models are prone to generate more hallucinated results over time.

In this paper, we introduce RCACopilot, a novel on-call system presenting an automatic end-to-end approach to cloud incident RCA. RCACopilot operates as an on-call system, empowering OCEs to construct 'incident handlers' - automated workflows tailored to each alert type, made up of reusable actions reflecting OCEs' expertise. These predefined handlers automatically streamline the collection of incident-specific diagnostic information from multiple sources, thus ensuring a more focused and relevant data accumulation process to avoid issues on either end of the information spectrum. Subsequently, the LLM component of RCACopilot processes this diagnostic data, predicting the category label of incident root causes and providing corresponding explanations. The combination of incident handlers and the LLM allows RCACopilot to significantly enhance adaptability and scalability in incident response. As a result, RCACopilot can effectively handle a diverse types of incidents while reducing the need for extensive human intervention.

The diagnostic information collection component of RCACopilot has been in use at Microsoft for over four years. In recent developments, a root cause prediction component was prototyped and, following a successful preliminary phase, has been actively deployed by an incident management team at Microsoft for a period spanning several months.

Contributions. This paper makes three main contributions:

- We propose RCACopilot, an automated end-to-end solution for cloud incident root cause analysis that enables on-call engineers to construct incident-specific automatic workflows for efficient data collection from multiple sources.
- We introduce the integration of a large language model within RCACopilot that autonomously analyzes the collected diagnostic data to predict incident root cause categories and generate explanations, demonstrating the potential of the large language model in root cause analysis.

- We showcase the real-world applicability of RCACOPILOT by presenting its successful adoption within Microsoft. This illustrates its practical effectiveness in enhancing root cause analysis efficiency, demonstrating the feasibility and benefits of our approach in real-world cloud scenarios.

2 Background and Motivation

In this section, we first introduce the concept and importance of incident root cause analysis. We then present real-world examples of troubleshooting guides and illustrate their inherent limitations. Lastly, we discuss the potential advantages of integrating a large language model into the RCA process, which motivates our work.

2.1 Incident Root Cause Analysis

In the realm of cloud services, an incident refers to any event that disrupts normal service operations or causes degradation in the quality of services. When such incidents occur, root cause analysis is performed to identify the underlying issue causing the disruption.

RCA in cloud services is a multi-faceted process:

- *Data Collection:* Gathering incident-related data from various sources such as logs, metrics, traces, or alerts is the first step in RCA.
- *Data Analysis:* The collected data is then analyzed to identify patterns, anomalies, or correlations that can possibly provide clues about the root cause of the incident.
- *Hypothesis Verification:* Based on the data analysis, hypotheses about the possible root cause are formulated and then verified by OCEs.

Given the complexity and dynamism nature of cloud systems, along with the immense volume of data involved, conducting RCA is a challenging task, which requires substantial expertise and time. Take the scale of Microsoft's email service as an example, which delivers over 150 billion messages daily. Ensuring the smooth operation of such a large-scale service demands an efficient and effective RCA approach. This is pivotal in maintaining a reliable and high-performing communication infrastructure, particularly for organizations that rely heavily on Microsoft's email servers.

2.2 The Opportunities and Challenges of Multi-Source Data in Incident Management

Managing incidents in the complex ecosystem of cloud services necessitates a comprehensive understanding of system states. This comprehension often stems from the consolidation of multi-source data, which includes traces, logs, and metrics. Traces represent tree-structured data detailing the flow of user requests, logs are semi-structured text recording hardware and software events, while metrics monitor service status or user-perceived metrics, forming time series data. While these individual data sources yield valuable insights,

> Troubleshooting Guide for Poisoned Messages
>
> 1. Go to the Poisoned Message Dashboard. This page gives a real-time, high-level view of the Poison Message feature. The charts should indicate whether the problem has resolved itself or is ongoing, as well as some sense of where it is occurring …
> 2. *The Dashboard newly implements an Exception Table* that has poisoned messages within a time frame. In most cases, whatever exception is causing an alert will rise to the top of the table …
> 3. You may also check the Poison Message Logs …
> …

Figure 1. A TSG for a poisoned message incident.

capitalizing on their potential has challenges. Traditional approaches such as TSGs, though useful, may fail to exploit the full wealth of multi-source data due to inherent limitations.

2.2.1 Opportunities of Multi-Source Data. Different data sources provide different perspectives on the system state. For instance, logs can offer detailed event sequences, metrics can reflect system performance over time, and traces can reveal the propagation of requests across services. Integrating these data sources can provide a more comprehensive view of the system, enabling more accurate and efficient incident diagnosis and resolution. Furthermore, multi-source data can facilitate correlation and causality analysis, which is crucial for root cause analysis. By analyzing the relationships between different data sources, we can identify patterns and anomalies that may indicate the root cause of an incident.

2.2.2 Challenges of Multi-Source Data. Despite its potential, effectively leveraging multi-source data in incident management is challenging. The sheer volume and complexity of data from various sources can be overwhelming, making it difficult to extract meaningful insights. Worse still, different data sources may provide inconsistent or conflicting information. Moreover, real-world data is often noisy, which can complicate analysis and lead to false conclusions.

2.2.3 Limitations of TSGs. Traditional TSGs represent an early attempt to leverage multi-source data for incident management. They guide OCEs to gather and analyze data from various sources to diagnose and resolve incidents. However, TSGs face several inherent limitations:

- *Manual data integration:* TSGs typically require OCEs to gather data from different sources manually. This process can be time-consuming and error-prone. Notwithstanding the existence of diverse troubleshooting guides and TSG recommendation techniques [20], dependence on TSGs

still remains a significant stress and burnout for OCEs due to the inherent limitations of the manual process.

- *Outdated information:* TSGs, as static documents, often struggle to stay up-to-date with the evolving system changes and new insights about incident root causes. This lag can lead OCEs to follow outdated or suboptimal troubleshooting steps. For example, a new feature ("Exception Table") to check Poison Message exceptions, mentioned as the second step in Figure 1, was not immediately incorporated into the TSG upon its release, causing potential inefficiencies in incident resolution.
- *Insufficient details and coverage:* High-level instructions often appear in TSGs, lacking in detail and specific guidance, which forces OCEs into additional research and prolongs incident mitigation. In the TSG example from Figure 1, the third step instructs to check the Poison Message Logs, leaving out crucial details and causing confusion for OCEs unfamiliar with this incident type. Additionally, TSGs may overlook common checks, e.g., disk space checks, leading to partial or inadequate incident resolutions.

2.3 The Promise of Large Language Models for Incident Management

The rapid advancements in natural language processing and machine learning have led to the development of powerful LLMs, which are reported to be effective at various downstream tasks with zero-shot and few-shot learning [5, 11, 28]. These models have shown exceptional performance in translation, summarization, and question-answering. Leveraging their potential for incident management in cloud computing systems could revolutionize the way OCEs identify and resolve incidents. By automating the interpretation aspect of incident management, LLMs can help alleviate the stress and cognitive load associated with complex on-call tasks for OCEs, which enables OCEs to focus more on higher-level jobs and decision-making.

2.4 Our Motivation

The motivation for our work is rooted in the challenges faced when using manual TSGs to diagnose incidents and identify the underlying root causes. Our goal is to develop an automated diagnostic process that harnesses the capabilities of LLMs to address various cloud incidents more effectively.

Different from previous work [42], which employs AI techniques to generate automated workflow from existing TSGs, our goal is to enable experienced OCEs to construct an automated pipeline for incident diagnosis. This approach allows OCEs to be directly assisted in identifying the root cause without the need to investigate intermediate diagnostic information, though they still have the option to do so.

We envision a future in which root cause analysis is predominantly automated, requiring minimal manual verification only when necessary. Our approach seeks to provide OCEs with timely, relevant, and accurate information for specific incidents, leading to more efficient RCA. By leveraging LLMs to predict root cause category, our research aims to alleviate the stress and cognitive load associated with incident management, ultimately enhancing the efficiency and effectiveness in addressing incidents.

3 Insights from Incidents

We conducted a comprehensive study of the one-year incidents from an email service from Microsoft, employing rigorous qualitative analysis methods. Specifically, each incident was carefully reviewed and categorized based on the characteristics of the problem, the source of the issue, and the impact on the system by our experienced OCEs. We paid particular attention to the root causes of the incidents, the effectiveness of the response, and the recurrence of similar issues. While our insights were indeed intuitively derived, they were firmly grounded in empirical data and analysis. Our study not only yielded valuable insights into incident patterns and challenges but also informed the development and refinement of our approach.

Insight 1: determining the root cause based on a single data source can be challenging. As an illustration, consider Incident 2 in Table 1, where a single server failed to perform DNS resolution for incoming packets due to the exhaustion of UDP hub ports on a front door machine. This example highlights the difficulties in relying solely on a single source (monitor alert) to diagnose complex issues.

When a mailbox server sends mail to external email recipients, it uses specific front-door servers (proxies). However, each front-door server has a limited number of available SMTP outbound proxy connections. If a mailbox server's proxy connection request fails, it will be unable to send messages to external recipients. In this incident, the monitor first raises an alert indicating detected failures when connecting to the front door server. However, this alert only signifies a connection issue between the mail server and the front door server, without even suggesting a DNS resolution problem. Consequently, the root cause remains unclear.
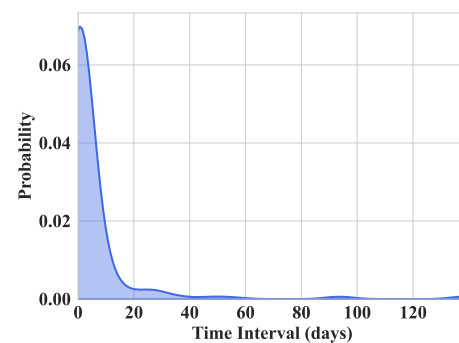

**Figure 2.** Recurring incidents proportion vs. time interval.

Table 1. Examples of cloud incidents in different root cause categories.

No.	Sev.	Scope	Category	Occur.	Symptom	Cause
1	1	Forest	AuthCertIssue	3	Tokens for requesting services were not able to be created. Several services reported users experiencing outages.	A previous invalid certificate overrode the existing one due to misconfiguration.
2	2	Machine	HubPortExhaustion	27	A single server failed to do DNS resolution for the incoming packages.	The UDP hub ports on the machine had been run out.
3	2	Forest	DeliveryHang	6	Mailbox delivery service hang for a long time.	Number of messages queued for mailbox delivery exceeded the limit.
4	2	Forest	CodeRegression	15	An SMTP authentication component's availability dropped.	Bug in the code.
5	2	Forest	CertForBogusTenants	11	The number of concurrent server connections exceeded a limit.	Spammers abused the system by creating a lot of bogus tenants with connectors using a certificate domain.
6	1	Forest	MaliciousAttack	2	Forest-wide processes crashed over threshold.	Active exploit was launched in remote PowerShell by serializing malicious binary blob.
7	2	Forest	UseRouteResolution	9	Poisoned messages sent to the forest made the system unhealthy.	A configuration service was unable to update the settings leading to the crash.
8	2	Forest	FullDisk	2	Many processes crashed and threw IO exceptions.	A specific disk was full.
9	2	Forest	InvalidJournaling	11	Messages stuck in submission queue for a long time.	The customer set an invalid value for the Transport config and caused TenantSettingsNotFoundException.
10	3	Forest	DispatcherTaskCancelled	22	Normal priority messages across a forest had been queued in submission queues for a long time.	Network problem caused the authentication service to be unreachable.

Insight 2: incidents stemming from similar or identical root causes often recur within a short period. We found that most recurring incidents (93.80%) tend to reappear within a brief span of 20 days, as shown in Figure 2. For instance, consider the category of Incident 9 from Table 1. This type of incident, triggered by invalid customer configuration, led to an accumulation of unprocessed messages in the queue, thereby significantly undermining its availability. Intriguingly, incidents of this category recurred 11 times in a span of merely 15 days. Likewise, the DispatcherTaskCancelled incidents (No. 10 in Table 1) and the DeliveryHang incidents (No. 3) reappeared 22 times and 6 times within a week and a single month, respectively. These can be attributed to several factors. Unresolved root causes from the initial response may lead to the same issue re-emerging, especially if the problem is complex or not fully understood. Secondly, systemic vulnerabilities, if not addressed, can be repeatedly exploited, causing similar incidents. Thirdly, external dependencies, such as reliance on a service that frequently experiences outages, can also lead to recurring incidents. These patterns suggest that by leveraging insights from previous incidents, we could swiftly identify the root cause of new occurrences with the same root cause.
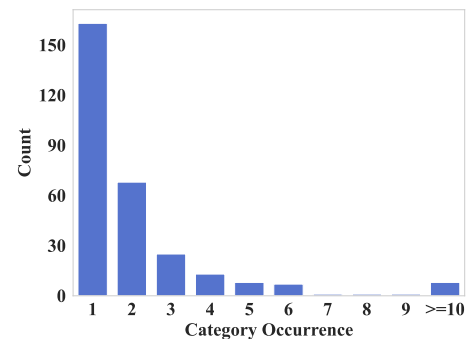
Figure 3. Distribution of incident category frequency.

Insight 3: incidents with new root causes occur frequently and pose a greater challenge to analyze. TSGs can help OCEs diagnose issues by providing clear investigation guidance. However, when incidents arise from new, previously unencountered root causes, OCEs face a set of challenges. For such incidents, no TSG exists, and OCEs may struggle to identify the underlying issues. For instance, Incident 1 is a high-severity (severity 1) incident caused by misconfiguration, which blocked the authentication token generation to lead to severe outages. Similarly, Incident 6 is a malicious attack caused by an attacker launching an exploit with a malicious blob. This type of attack had never been encountered before, leaving OCEs without an existing TSG to reference. Lower severity level (severity 2) incidents, such as Incident 5, are also susceptible to this challenge when the spammer first abuses the system. As Figure 3 shows, incidents with a new root cause category account for 24.96% (163 among 653) of all incidents. If OCEs spend their time searching for nonexistent TSGs, the incident's impact could escalate further. Recognizing this challenge, it is necessary to propose a new approach that can effectively infer, categorize and explain the root causes for such unseen incidents, thereby reducing the time OCEs take to identify and address these unique incidents.

4 RCACopilot

RCACopilot has two stages: the diagnostic information collection stage and the root cause prediction stage as shown in Figure 4.

Diagnostic information collection stage: This is the initial stage, where the incident is parsed and matched to the pre-defined incident handler. Each incident handler is tailored to a specific alert type. Upon matching the incident with the appropriate handler, RCACopilot proceeds to collect relevant diagnostic data from a variety of sources.

Root cause prediction stage: Once the diagnostic information is collected, RCACopilot transitions into the root cause prediction stage. In this phase, RCACopilot applies its predictive module to determine the likely root cause category of the incident. This prediction is not a mere categorization, but it is also supplemented with an explanation detailing how RCACopilot arrived at the given prediction. Subsequently, the predicted category label is presented to experienced OCEs for review.

4.1 Diagnostic Information Collection Stage

Driven by Insight-1 in Section 3, RCACopilot aims to collect multi-source data for RCA. Specifically, for each alert type, an incident handler is constructed, comprising a series of actions to collect diagnostic information. Alert types are used to categorize alerts based on specific monitors. Incidents sharing the same alert type exhibit similar symptoms, though they may stem from different root causes.

The RCACopilot incident handler is a workflow that consists of a series of actions. Each action is a function that can be executed to collect specific diagnostic information from a target data source. OCEs can build and modify these handlers based on their expertise. The handler includes three distinct actions: *scope switching action*, *query action*, and *mitigation action*, which will be explained in Section 4.1.2. Each action generates an output, guiding the control flow of the incident handler. We use a RCACopilot handler that diagnoses Incident 7 in Table 1 as an example to illustrate the handler usage.

4.1.1 Incident handler. The decision-making process that OCEs employ when handling an incident resembles a decision tree's control flow. The root node in the incident handler is the incident alert type, which is gathered from the system monitor. We distilled OCE operations into three actions when constructing the incident handler. As OCE operations can be similar to different incident types (e.g., conducting a common disk check or query to a database), we designed RCACopilot handler actions to be reusable across all handlers. We also maintain the versions of the handlers in the database, which can be used to track their historical changes.

RCACopilot's incident handlers are constructed manually first and can be updated and modified dynamically by OCEs, allowing them to stay abreast with the most recent system changes and newly discovered root causes. For instance, when a new metric is introduced into the system, OCEs only need to construct a new action to collect the relevant data and incorporate it into the corresponding incident handler, which can ensure timely adaptation.

4.1.2 Handler action. RCACopilot leverages the synergy of multi-source data. The system uses predefined reusable actions in the incident handler to automatically collect relevant diagnostic information from diverse sources. The automated integration of data not only saves time but also reduces the likelihood of human error. It provides a more comprehensive view of the system state, facilitating efficient and accurate incident resolution. This significantly lightens the workload of OCEs, reducing stress and burnout, and enhancing the effectiveness of the incident resolution process. The action in the handler could be one of the following:

Scope switching action: This action facilitates precision in RCA by allowing adjustments to the data collection scope based on the specific needs of each incident. For instance, as depicted in Figure 5, if an alert originates at the 'forest' level, signifying an issue within a specific forest, and the problem type is identified as 'Busy Hub', the scope switching action can adjust the scope to the 'machine' level. This modification allows for a more fine-grained investigation, specifically assessing if a singular hub server is overly taxed.

The implementation of this action ensures that we efficiently navigate the information spectrum. When the investigation requires a more targeted approach, this action
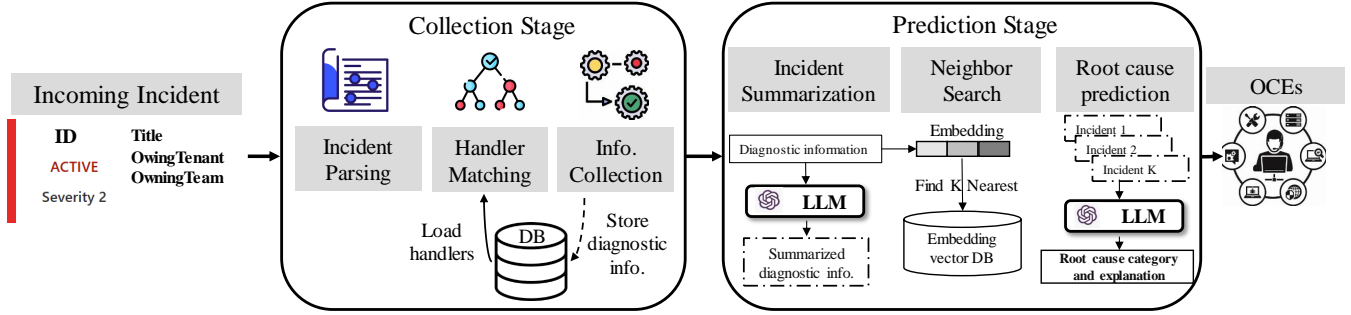
Figure 4. RCACopilot architecture.

can narrow the data collection scope. Conversely, if a more holistic view is necessary, it can widen the scope, say from a single machine to an entire forest. This flexibility contributes to a more balanced and effective diagnostic data collection.

Query action: Query action can query data from different sources and output the query result as a key-value pair table. This type of action can also be hooked to executing a specific script with pre-defined parameters. Usually, scripts are internal automatic investigation tools for a service, and only the service team has access to the tools.

For instance, in Figure 5, the "Known issue?" action node queries the database to see whether the current incident is a known one or not based on its alert messages. If it is a known issue, execution flow will enter the "True" branch to give mitigation actions directly. Otherwise, a query script that can aggregate threads with the same stack traces will be executed. It will obtain an instantaneous list of the stacks on all the managed threads in the target process and then group common stacks together in order to identify potential deadlocks/blocking code paths in the process.

The query action can also output an enum value to decide the next action node to execute, e.g., after getting the top error message on the exception stack traces, i.e., "Get top error msg" node, the next action node to be run depends on the exception type. Based on the error messages, a specific team will be reported and engaged, as shown in Figure 5.

Mitigation action: This action refers to the strategic steps suggested to alleviate an incident, such as "restart service" or "engage other teams", as depicted in Figure 5. It's important to note that handlers do not always provide exact mitigation strategies for every incident, due to handlers' pre-defined nature, which may not cover all possible situations. For instance, Incident 4 in Table 1, categorized under code regression, presents a case where identification and rectification of such code issues can be challenging. In cases where the incident handler is uncertain, it will offer intermediate diagnostic information to the OCEs without mitigation.

4.1.3 Multi-source diagnostic information. RCACopilot's diagnostic information collection stage serves as a valuable tool for OCEs by aggregating data from a myriad

of sources. OCEs only need to customize the action in the handler to acquire the diagnostic information from a target source. For instance, as illustrated in Figure 6, RCACopilot can assimilate diverse data such as error logs, exception stack traces, and socket metrics related to a specific incident. The error log and exception stack trace alone does not provide sufficient insight to identify the root cause of the incident. However, when supplemented with the socket metrics, a more comprehensive picture emerges. In this example, it is clear that the UDP socket is exhausted, which is the root cause.

In the case of new incidents, RCACopilot can perform a range of common checks, such as evaluating the provisioning status or analyzing thread stacks. This assists OCEs in gaining a holistic understanding of the situation. Note that the information collected is pre-defined in the actions of the RCACopilot handler, ensuring that only relevant data is gathered, thus avoiding overwhelming information that is unnecessary. By providing this comprehensive diagnostic information, RCACopilot empowers OCE teams to troubleshoot issues efficiently. They can use the gathered information as guidance to address incidents more effectively.

4.2 LLMs for Incident Explanation

Upon thorough investigation, each incident within our service is manually assigned a root cause category by our seasoned OCEs. OCEs will use the categories to classify the historical incidents and guide the new incoming incidents' RCA. However, reasoning the incidents and inferring their categories are time-consuming and potentially overwhelming for OCEs, who have a tight time budget. Given this, we have identified the categorization of incident root causes as our primary downstream task.

Recently, LLMs have demonstrated remarkable capabilities in understanding the context of downstream tasks and generating relevant information from demonstrations, making them a possible choice for incident RCA. However, reasoning the incident root cause is not a simple task, and LLMs may not be able to achieve the optimal results on long-tail or
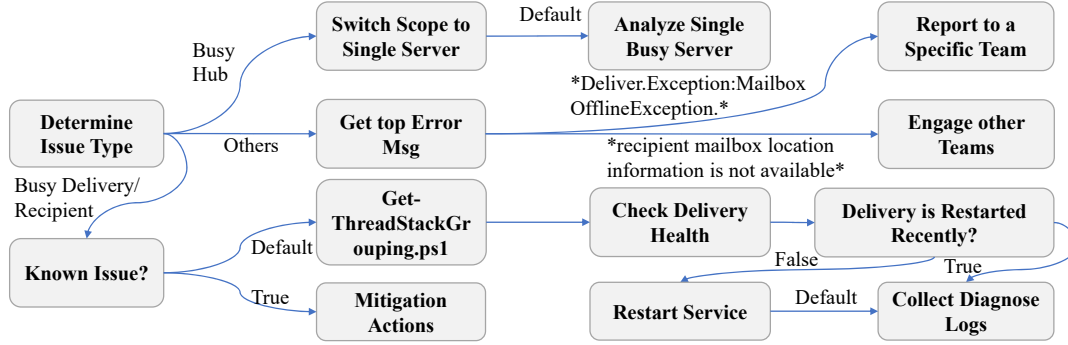
Figure 5. A RCACopilot handler for too many messages stuck in the delivery queue alert.

```
DatacenterHubOutboundProxyProbe probe log result from
[MachineID].
Total Probes: 2, Failed Probes: 2
 Id   Level   Created                 Description
 -    --      ---                     ---
 2    Error   11/21/2022  2:04:20 AM  Probe result
 2    Error   11/21/2022  1:49:20 AM  Probe result
Failed probe error:
Name: No such host is known.
A WinSock error: 11001 encountered when connecting to
host: [HOST NAME]
Count: 2
...
Exceptions:
InformativeSocketException: No such host is known.
A WinSock error: 11001 encountered when connecting to
host: [HOST NAME]
at TcpClientFactory.Create(...)
at SimpleSmtpClient.Connect(...)
...
Total UDP socket count: 15276
Total UDP socket count by process and processId (top
5 only):
14923: Transport.exe, 203736
15: w3wp.exe, 102296
8: svchost.exe, 4748
7: Microsoft.Transport.Store.Worker.exe, 74060
7: Microsoft.Transport.Store.Worker.exe, 87724
```

Figure 6. Diagnostic information for hub port exhaustion.

domain-specific tasks without any guidance [6, 22]. Chain-of-Thoughts (CoT) prompting is a gradient-free technique that elicits LLMs to generate intermediate reasoning steps that lead to the final answer. In few-shots CoT prompting, a few manual demonstrations that are composed of a question and a reasoning chain that leads to an answer for each of them. Inspired by the above ideas, diagnostic information provided by RCACopilot handlers can be used as ingredients for the reasoning process of the incidents.

4.2.1 Embedding model. Our observation is that the *semantics of incidents can be revealed from the context in which the diagnostic information is described.* A common approach to extracting such contextual semantics involves the use of embedding models. The objective is to map the diagnostic information into an embedding space (i.e., numeric vector space), where the distances between vectors represent the semantic similarity of incidents. Choosing a computationally efficient embedding model allows us to preserve accuracy while handling a large number of incidents.

We employ FastText as our embedding model, which is efficient, insensitive to text input length, and generates dense matrices, making it easy to calculate the Euclidean distance between similar vectors. Furthermore, since our downstream task is domain-specific to the incident root cause reasoning, and the incident-related information is internal to our company, we opt to train a FastText model on our historical incidents rather than using a pre-trained large language model as our embedding model, which is costly and inefficient. Additionally, we provide users with the flexibility to customize their embedding model if desired.

4.2.2 Nearest neighbor search. Incidents are heterogeneous, making it impractical to combine all past incidents' information for sampling due to the prompt length limitations, even after summarization. To selectively choose past cases as samples in the prompt, we design a new similarity formula:

$$Distance(a, b) = ||a - b||_2$$

$$Similarity(a, b) = \frac{1}{1 + Distance(a, b)} * e^{-\alpha|T(a) - T(b)|}$$

to calculate the similarity between two incidents. It first computes the Euclidean distance for every pair of incident vectors. Importantly, it also takes into account the temporal distance between incidents, reflecting our Insight-2 in Section 3. Here, $T(x)$ stands for the date of incident x. This consideration of temporal distance is crucial as it influences the relevance of past incidents to the current ones. After calculating similarities, we select the top K incidents from different categories as demonstrations for the LLM. This

approach ensures a diverse and representative set of incidents for effective LLM reasoning. The values of α and K have been determined as 0.3 and 5, respectively, through empirical evaluation, as will be presented in Section 5.4.

4.2.3 Diagnostic information summary.

LLMs have shown potential for automatic summarization [40]. Nonetheless, the length of the diagnostic information collected from RCACopilot handlers is often too extensive. As shown in Figure 6, the diagnostic information of an incident can have more than 2000 tokens with low readability of the log messages. The considerable number of tokens in the incident description can pose challenges for the LLM to effectively process and may introduce noise. Therefore, feeding the diagnostic information of an incident directly into the LLM to make a prediction could not be an ideal choice, let alone using the information from multiple sources. In this regard, we add another layer to leverage the LLM's ability to summarization to summarize the diagnostic information first before making the diagnosis reasoning. We construct the prompt in the way of Figure 7. We ask LLM to summarize the diagnostic information into 120-140 words without outputting any unrelated information. This summarization process makes the diagnostic information more concise and informative, which forms the basis for the later CoT prompting. Figure 8 illustrates a more readable and concise text generated by RCACopilot, which is a summary (113 tokens) of the previous diagnostic information example in Figure 6, highlighting the key details such as the number of UDP ports used and the process utilizing the most. Specifically, we employ the tiktoken [41] tokenizer to count text tokens.

> "*Please summarize the above input. Please note that the above input is incident diagnostic information. The summary results should be about 120 words, no more than 140 words, and should cover important information as much as possible. Just return the summary without any additional output.*"

Figure 7. Prompt to summarize diagnostic information.

4.2.4 Prediction prompt construction.

CoT prompting is a gradient-free technique that guides LLMs to produce intermediate reasoning steps leading to the final answer. In few-shot CoT prompting, several demonstrations include a question and a reasoning chain that directs the answer. By drawing inspiration from automatically constructing the prompt to form the reasoning chains [60], we can view the summarized diagnostic information and the labeled root cause categories as questions and reasoning, so finding the nearest incident neighbor is the automatic reasoning chain construction, aligning with the CoT prompting context well. Note that we use the original incident information to do the

> "*The DatacenterHubOutboundProxyProbe has failed twice on the backend machine, with both failures due to a WinSock error 11001 indicating that the host is unknown. This error was encountered while attempting to connect to the host. The error is associated with the EOP service and has not been notified yet. The failure context suggests the same issue.* **The total UDP socket count is 15276, with the majority being used by the Transport.exe process.** *The issue seems to be related to the SMTP connection and requires further investigation.*"

Figure 8. The summarized diagnostic information.

embedding and nearest neighbor search, and use the corresponding summarized information as part of demonstrations in the prompt. We construct the prompt like Figure 9 to ask the LLM to choose the most likely incident that has the same root cause as the current incident, and also we explicitly push the LLM to reason by using "give your explanation" indications in the prompt.

> **Context:** The following description shows the error log information of an incident. Please select the incident information that is most likely to have the same root cause and **give your explanation** (just give one answer). If not, please select the first item "Unseen incident".
> **Input:** The DatacenterHubOutboundProxyProbe probe result from [BackEndMachine] is a failure ...
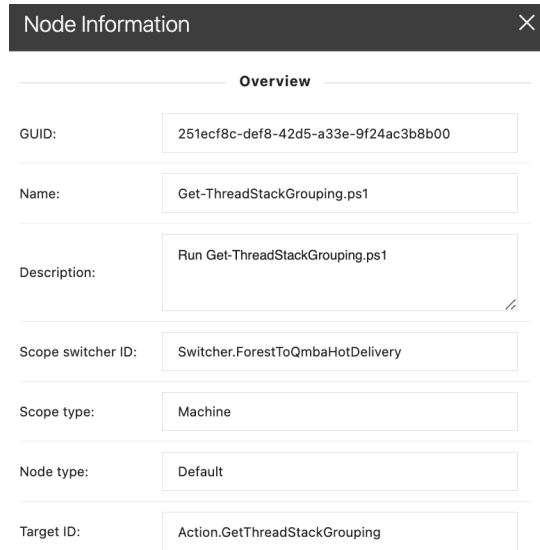> Options:
>   **A:** Unseen incident.
>   **B:** The DatacenterHubOutboundProxyProbe has failed twice ... category: **HubPortExhaustion**.
>   **C:** There are 62 managed threads in process TransportDelivery ... category: **AuthCertIssue**.

Figure 9. The prompt to predict incident category.

4.3 Implementation

We have developed and deployed RCACopilot using a combined total of 58,286 lines of code, consisting of 56,129 lines of C# and 2,157 lines of Python.

To facilitate the building of the RCACopilot incident handler, we have implemented RCACopilot's handler construction as a web application as shown in Figure 10. To support a new type of alert in RCACopilot, OCEs only need to add a new handler in the handler construction GUI according to her expertise. After the new handler has been constructed, it will be stored in the database, and OCEs can modify it by creating new action nodes or deleting old nodes.

Figure 10. Web-based user interface of RCACOPILOT for handler construction.

5 Evaluation

We aim to answer the following questions in our evaluation:

(1) How effective and efficient is RCACOPILOT as an on-call system when predicting root cause categories and assisting OCEs? RCACOPILOT achieves 0.766 and 0.533 for Micro-F1 and Macro-F1 separately when predicting the root cause category of cloud incidents, outperforming all our baselines with a low running overhead (4.205 seconds). RCACOPILOT is also able to generate new root cause category labels for unseen incidents with explanations.

(2) How do different components of RCACOPILOT facilitate its diagnosis and prediction? RCACOPILOT has proven that the diagnostic information collection component, GPT summarization, and chain-of-thoughts prompting all contribute to RCACOPILOT's prediction effectiveness.

(3) Is RCACOPILOT suitable for deployment in real production services, and are RCACOPILOT's results trustworthy? RCACOPILOT's diagnostic information collection module has been deployed across 30 teams within Microsoft for over four years. To evaluate the trustworthiness of RCACOPILOT, each experiment was conducted over three rounds, and RCACOPILOT can consistently achieve a high Micro-F1 score of over 0.70 and a Macro-F1 score exceeding 0.50.

All experiments are performed on the server with Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz, 32.0 GB physical memory, and Intel UHD Graphics 630. The OS of the server is Windows 11 Enterprise.

5.1 Target System and Dataset

We evaluate RCACOPILOT in a global email service system named Transport within the Microsoft. The Transport team focuses on developing and maintaining the components responsible for mail flow, routing, and delivery. This system interacts with various other services to ensure seamless integration with a multitude of products and services, including serviceA, serviceB, and serviceC. Hence, it is representative of complex, real-world systems that interact with multiple components. With around 150 billion messages being delivered daily, Transport operates at a colossal scale and caters to customers worldwide, adding another layer of diversity and complexity. The system ensures the secure and effective transmission of emails between users, utilizing various protocols such as SMTP, IMAP, and POP3. Given its crucial role in communications infrastructure, it is essential to have effective and efficient incident management capabilities.

We collect a one-year dataset of 653 incidents from Microsoft's Transport service to investigate RCACOPILOT's efficacy in practice. It is important to note that each of these incidents represents complex issues in a large-scale, globally distributed system, and thus each provides valuable insights. The dataset is manually labeled with root cause categories by experienced OCEs, which serves as our ground truth. We divide the incidents into train (75%) and testing sets (25%).

We conduct experiments on two large language models in RCACOPILOT, *i.e.*, GPT-3.5-turbo, and GPT-4 (8K tokens), which are the latest models from OpenAI. We choose GPT-4 as the default model in RCACOPILOT because it has the best performance.

5.2 Compared Approaches

We have selected XGBoost, FastText, and fine-tuned LLMs as our baselines to compare with RCACOPILOT. After training or fine-tuning with the training dataset, we directly apply these approaches to the testing set to do the classification task. We have also made another two variants, i.e., GPT-4 Prompt and Embed. to evaluate the design of RCACOPILOT.

- **XGBoost** provides a parallel tree boosting that has been commonly used in the networking system diagnosis.
- **FastText** is a popular lightweight textual embedding approach, which has been adopted in testbed studies with fault injections for root cause diagnosis tasks. We directly apply FastText to our dataset to do the classification.
- **Fine-tune GPT** is to fine-tune a pre-trained GPT-3.5 model with our training dataset and evaluate its performance on our testing dataset with the temperature parameter set to 0. It does not use a prompt design (i.e., CoT prompting) like RCACOPILOT but directly predicts the category with the original diagnosis information. Note that GPT-4 is currently not available for fine-tuning.

- **GPT-4 Prompt** is a variant of RCACᴏᴘɪʟᴏᴛ that directly predict category with RCACᴏᴘɪʟᴏᴛ's diagnosis information summaries. Its prompt only contains the incident being predicted, so there is no historical incident information as demonstrations.
- **GPT-4 Embed.** is a variant of RCACᴏᴘɪʟᴏᴛ that changes the embedding model from FastText to GPT embedding.

Table 2. Effectiveness of different methods.

Method	F1-score		Avg. Time (s)	
	Micro	Macro	Train.	Infer.
FastText [61]	0.076	0.004	10.592	0.524
XGBoost [3]	0.022	0.009	11.581	1.211
Fine-tune GPT [1]	0.103	0.144	3192	4.262
GPT-4 Prompt	0.026	0.004	–	3.251
GPT-4 Embed.	0.257	0.122	1925	3.522
RCACᴏᴘɪʟᴏᴛ (GPT-3.5)	0.761	0.505	10.562	4.221
RCACᴏᴘɪʟᴏᴛ (GPT-4)	**0.766**	**0.533**	10.562	4.205

5.3 Effectiveness and Efficiency

We evaluate RCACᴏᴘɪʟᴏᴛ's effectiveness by predicting the root cause category of an incident based on the summarized diagnostic information using micro and macro F1-score metrics. These metrics calculate the harmonic mean of the precision and recall. The micro F1-score aggregates the performance of all classes, taking into account the contribution of each sample, while the macro F1-score focuses on the performance of each individual class. RCACᴏᴘɪʟᴏᴛ achieves a micro F1-score of 0.766 and a macro F1-score of 0.533 on our testing dataset.

As shown in Table 2, RCACᴏᴘɪʟᴏᴛ outperforms other approaches, and it tends to incur an acceptable higher runtime overhead. The performance of baseline approaches is poor, since multiple root cause categories exhibit a long tail (imbalanced) distribution, as shown in Figure 3, and traditional machine learning models (FastText and XGBoost) and fine-tuning GPT model need a large amount of training data to produce accurate predictions. Directly employing GPT-4 prompt or GPT-4 embedding approach without our design lacks domain-specific knowledge for GPT-4 to make decisions. On the contrary, RCACᴏᴘɪʟᴏᴛ leverages the powerful LLM to learn the domain-specific knowledge from minimal cases, so that it can achieve the best performance. Results indicate that RCACᴏᴘɪʟᴏᴛ not only provides higher accuracy but also maintains a reasonable level of efficiency, making it a suitable choice for incident root cause analysis.

When facing incidents that RCACᴏᴘɪʟᴏᴛ has never seen before, RCACᴏᴘɪʟᴏᴛ is capable of generating a new category keyword to depict the new incident case. For example, Incident 8 in Table 1 is a new incident case that RCACᴏᴘɪʟᴏᴛ

has never encountered. RCACᴏᴘɪʟᴏᴛ's prediction component is able to predict it as a new category "I/O Bottleneck". Although OCEs subsequently categorize it as "DiskFull" in post-investigation, the fundamental aspects of the problem identified by RCACᴏᴘɪʟᴏᴛ align closely with the human-derived label. The corresponding RCACᴏᴘɪʟᴏᴛ's explanation, illustrating how it arrived at the "I/O Bottleneck" categorization, is provided in Figure 11.

> *The prediction of "I/O Bottleneck" was made based on the occurrence of System.IO.IOExceptions within crucial functions handling input/output operations, suggesting an issue with data processing. The nested exception within the DiagnosticsLog module reinforces this notion. These errors, combined with crashes on different backend machines, point to a system struggle with handling data flow.*

Figure 11. RCACᴏᴘɪʟᴏᴛ's explanation of an incident.

5.4 Comparison Analysis

To understand how different components of RCACᴏᴘɪʟᴏᴛ facilitate root cause analysis, we conduct an ablation study on the different RCACᴏᴘɪʟᴏᴛ's components.

Evaluation on diagnostic information. First, we evaluate the impact of diagnostic information on effectiveness. In particular, we compare diagnostic information collected from the collection stage with other different incident-related information, namely, incident alert information and RCACᴏᴘɪʟᴏᴛ handler action output. AlertInfo includes the alert type and alert scope. Alert type is a pre-defined anomaly description from a monitor, which only reflects a symptom of the incident instead of the root cause, e.g., an exception type from external monitors. The alert scope is the scope of the incident, e.g., a single machine. ActionOutput is the output of a series of executed RCACᴏᴘɪʟᴏᴛ actions, which are hashed as key-value pairs. As shown in Table 3, using diagnostic information alone can outperform others in both Micro-F1 (0.689) and Macro-F1 scores (0.510). The interesting observation here is that mixing the diagnostic information

(a) F1 micro.   **(b)** F1 macro.
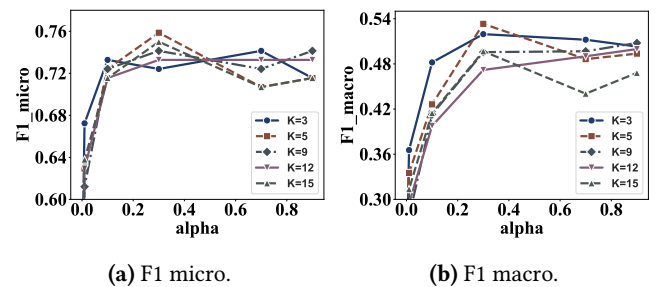
Figure 12. Effectiveness of using different K and alpha.

Table 3. Effectiveness of different prompt context for RCACOPILOT. ✓^{sum.} stands for the summarized diagnostic information.

Data Source			F1-score	
AlertInfo	DiagnosticInfo	ActionOutput	Micro	Macro
	✓		0.689	0.510
	✓^{sum.}		**0.766**	**0.533**
✓			0.379	0.245
✓	✓		0.525	0.511
✓		✓	0.431	0.247
	✓	✓	0.501	0.449
✓	✓	✓	0.440	0.349

with others will not enhance RCACOPILOT's predictive capabilities. This demonstrates that an excess of information can negatively impact the LLM's prediction performance.

Evaluation on GPT summarization. We evaluate the role of GPT summarization in enhancing RCACOPILOT's effectiveness. As depicted in Table 3, utilizing summarized diagnostic information leads to the highest Micro-F1 and Macro-F1 scores, marking improvements of 0.077 and 0.023, respectively, over the non-summarized diagnostic information. The results demonstrate that the summarization step effectively condenses the information, allowing for more efficient and accurate processing of incident data.

Evaluation on few-shots CoT reasoning. We assess how few-shots CoT reasoning contributes to improving effectiveness. GPT-4 Prompt approach in Table 2, which directly predicts the category without any sample, only achieves 0.026 and 0.004 for Micro-F1 and Macro-F1 respectively. As shown in Figure 12a and Figure 12b, we compare the performance of RCACOPILOT with different numbers of samples in the Chain-of-thoughts reasoning. Our analysis reveals that the best combination of the number of samples and alpha values are 5 and 0.3, which achieves the highest F1 scores. Note that more samples in the CoT reasoning do not always incur an improvement for RCACOPILOT, and the value of the alpha plays an important role in deciding the effectiveness. When the alpha is appropriate, it allows RCACOPILOT to better capture the time relationships between different incidents, leading to more accurate predictions.

5.5 Deployment Status and Scale

We have successfully deployed RCACOPILOT's diagnostic information collection module across over 30 teams within Microsoft, where it has been in active use for over four years. The system is tailored to each team's specific requirements, with custom handlers built for each unique setting. Not all handlers are currently enabled in the production environment, as some are still under development and rigorous testing. We select the top 10 teams that utilize the most

RCACOPILOT incident handlers as shown in Table 4. We observe that the average running time for each incident ranges from 15 seconds to 841 seconds. The highest running time is attributable to the team's large-scale and complex system infrastructure. The root cause prediction module has also been rolled out in the Transport service.

As part of our commitment to continuous improvement and quality user experience, we have incorporated a feedback mechanism in incident notification emails to get user perspectives from OCEs. According to the collected feedback, most OCEs expressed satisfaction with RCACOPILOT. Despite the manual effort involved in creating a new incident handler, OCEs find the process convenient when reusing and modifying handler actions from the database. RCACOPILOT is able to save OCEs a significant amount of time to collect diagnostic information, triage incident, perform mitigation and do postmortem analysis.

Table 4. Teams in Microsoft using RCACOPILOT to automatically collect diagnostic information.

Team	Avg. exec. time (seconds)	# Enabled handler
Team 1	841	213
Team 2	378	204
Team 3	106	88
Team 4	449	42
Team 5	136	41
Team 6	91	34
Team 7	449	32
Team 8	255	32
Team 9	323	31
Team 10	22	18

5.6 Tustworthiness

While GPT has shown great potential and impressive results in various tasks, it is known to exhibit some instability in certain complex tasks such as question answering, as noted by Tan et al. [45]. These instabilities could potentially lead to variable results. In order to ensure the trustworthiness and stability of the GPT's predictive capabilities in RCACOPILOT, each experiment has been conducted three rounds. In each round, RCACOPILOT was able to maintain a high level of performance, with the Micro-F1 consistently above 0.70 and the Macro-F1 remaining above 0.50.

6 Discussion

RCACOPILOT's effectiveness depends on the ability of the LLM. Currently, RCACOPILOT is only integrated with OpenAI's GPT models, and we have not yet explored the potential effectiveness of other available LLMs. As such, the model's performance may vary depending on the strengths and weaknesses of the specific LLM employed.

We conducted our evaluation of RCACopilot's prediction module using the incident dataset from Transport. The dataset was prepared with the assistance of experts in Transport team, given their extensive experience and established practice of incident labeling. Note that the effectiveness of RCACopilot is also influenced by the quality of the root cause category labels written by human. Currently, all root cause categories are manually labeled by our experienced OCEs. RCACopilot's diagnosis information collection has been deployed in over 30 teams. Consequently, a valuable future work would be to evaluate RCACopilot across different services to gain a more comprehensive understanding of its generalizability and adaptability.

The handler in RCACopilot is designed to initiate responses based on alerts from monitors/watchdogs. This ensures that when there is a designated incident handler for a particular alert type, it gets activated with an accuracy rate of 100%. Nevertheless, it's crucial to highlight that RCA-Copilot's capabilities are constrained in scenarios where the monitors fail to detect an incident, or when there is an absence of a corresponding handler for a particular incident. This, in turn, limits the applicability of RCACopilot.

We conducted three rounds of experiments to evaluate RCACopilot's effectiveness. However, the occasional instability of LLMs can influence their effectiveness, causing variations across different rounds. Another potential threat to internal validity lies in the implementation of our approach and those we compared against. To mitigate this risk, two authors have carefully checked the code. In particular, our implementation is based on the matured frameworks.

7 Related Work

Root cause analysis. Root cause analysis in large cloud services has become a popular topic of research in the system and software engineering communities [2, 7, 15, 16, 23, 30, 33, 36, 50, 59]. It aims to identify the root causes of failures and performance issues based on various data sources, such as metrics, logs, and traces. Previous studies have proposed different approaches for root cause analysis using one of these data sources. For example, some methods rely on metrics to extract failure patterns [36, 58] or to construct service dependency graphs [25, 35]. Others use logs to analyze a subset of log messages [1, 57] or to examine the details within each log message [27, 56]. Moreover, some techniques utilize trace to locate the faulty service [26, 29, 48, 54]. Different from prior work, we build a system that can automatically integrate metrics, logs, and traces for root cause analysis with state-of-the-art large language models.

Large Language Models. In recent years, the rise of LLM has brought new opportunities to the field of software systems by enabling various tasks such as code generation, summarization, repair, testing, and root cause analysis [1, 13, 39, 40]. For example, Mastropaolo *et al.* [40] studied the

ability of fine-tuned T5 in the following tasks: automatic bug fixing, generation of assert statements, code summarization, and injection of code mutants. LANCE [39] uses fine-tuned T5 to automatically generate logging statements for Java methods. VulRepair [13] also fine-tune T5 on vulnerability repairs datasets to automatically propose vulnerability fixes. Zhang *et al.* [55] proposes to use prompting for LLM to improve code version control. Ahmed *et al.* [1] fine-tune GPT-x models to recommend root causes and mitigation steps to facilitate cloud incident management. In contrast to previous studies, RCACopilot employs advanced LLMs to summarize diagnosis data and leverage the chain-of-thoughts ability to predict and explain root causes.

8 Conclusion

RCACopilot represents a pioneering tool in the realm of cloud incident management, facilitating efficient root cause analysis for OCEs. It introduces a unique approach to multi-source data collection through its diagnostic information collection stage, utilizing predefined incident handlers. These handlers, constructed by OCEs, systematically gather multi-source diagnostic information, which sets the foundation for the subsequent analysis. Furthermore, RCACopilot integrates a large language model in its root cause prediction stage. This model autonomously processes the collected diagnostic data, predicting and explaining the root cause category. This integration of AI techniques into cloud incident management demonstrates the potential of RCACopilot in enhancing the efficiency and accuracy of root cause analysis.

Acknowledgement

References

[1] Ahmed, T., Ghosh, S., Bansal, C., Zimmermann, T., Zhang, X., and Rajmohan, S. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE'23)* (2023).

[2] Alquraan, A., Takruri, H., Alfatafta, M., and Al-Kiswany, S. An analysis of network-partitioning failures in cloud systems. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)* (2018).

[3] Arzani, B., Ciraci, S., Loo, B. T., Schuster, A., and Outhred, G. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM'16)* (2016).

[4] Bansal, C., Renganathan, S., Asudani, A., Midy, O., and Janaki-raman, M. Decaf: Diagnosing and triaging performance issues in large-scale cloud services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (2020).

[5] BROWN, T., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J. D., DHARI-WAL, P., NEELAKANTAN, A., SHYAM, P., SASTRY, G., ASKELL, A., ET AL. Language models are few-shot learners. *Advances in neural information processing systems* (2020).

[6] CHALKIDIS, I. Chatgpt may pass the bar exam soon, but has a long way to go for the lexglue benchmark. *arXiv preprint arXiv:2304.12202* (2023).

[7] CHEN, H., DOU, W., JIANG, Y., AND QIN, F. Understanding exception-related bugs in large-scale cloud systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19)* (2019).

[8] CHEN, J., HE, X., LIN, Q., XU, Y., ZHANG, H., HAO, D., GAO, F., XU, Z., DANG, Y., AND ZHANG, D. 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'19)* (2019).

[9] CHEN, J., HE, X., LIN, Q., ZHANG, H., HAO, D., GAO, F., XU, Z., DANG, Y., AND ZHANG, D. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19)* (2019).

[10] CHEN, J., ZHANG, S., HE, X., LIN, Q., ZHANG, H., HAO, D., KANG, Y., GAO, F., XU, Z., DANG, Y., ET AL. How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE'20)* (2020).

[11] CHEN, M., TWOREK, J., JUN, H., YUAN, Q., PINTO, H. P. D. O., KAPLAN, J., EDWARDS, H., BURDA, Y., JOSEPH, N., BROCKMAN, G., ET AL. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[12] CHEN, Y., SUN, X., NATH, S., YANG, Z., AND XU, T. Push-Button Reliability Testing for Cloud-Backed Applications with Rainmaker. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)* (2023).

[13] FU, M., TANTITHAMTHAVORN, C., LE, T., NGUYEN, V., AND PHUNG, D. Vulrepair: a t5-based automated software vulnerability repair. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'22)* (2022).

[14] GANATRA, V., PARAYIL, A., GHOSH, S., KANG, Y., MA, M., BANSAL, C., NATH, S., AND MACE, J. Detection is better than cure: A cloud incidents perspective. In *Proceedings of the 31st Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (2023).

[15] GAO, Y., DOU, W., QIN, F., GAO, C., WANG, D., WEI, J., HUANG, R., ZHOU, L., AND WU, Y. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering (ESEC/FSE'18)* (2018).

[16] GHOSH, S., SHETTY, M., BANSAL, C., AND NATH, S. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing* (2022).

[17] GU, J. T., SUN, X., ZHANG, W., JIANG, Y., WANG, C., VAZIRI, M., LEGUNSEN, O., AND XU, T. Acto: Automatic End-to-End Testing for Operation Correctness of Cloud System Management. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP'23)* (2023).

[18] HE, S., ZHANG, X., HE, P., XU, Y., LI, L., KANG, Y., MA, M., WEI, Y., DANG, Y., RAJMOHAN, S., ET AL. 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 (ESEC/FSE)* (2022).

[19] INAM, M. A., CHEN, Y., GOYAL, A., LIU, J., MINK, J., MICHAEL, N., GAUR, S., BATES, A., AND HASSAN, W. U. Sok: History is a vast early warning system: Auditing the provenance of system intrusions. In *2023 IEEE Symposium on Security and Privacy (S&P'22)* (2022).

[20] JIANG, J., LU, W., CHEN, J., LIN, Q., ZHAO, P., KANG, Y., ZHANG, H., XIONG, Y., GAO, F., XU, Z., ET AL. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20)* (2020).

[21] JIN, P., ZHANG, S., MA, M., LI, H., KANG, Y., LI, L., LIU, Y., QIAO, B., ZHANG, C., ZHAO, P., ET AL. Assess and summarize: Improve outage understanding with large language models. In *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (2023).

[22] KASAI, J., KASAI, Y., SAKAGUCHI, K., YAMADA, Y., AND RADEV, D. Evaluating gpt-4 and chatgpt on japanese medical licensing examinations. *arXiv preprint arXiv:2303.18027* (2023).

[23] LEESATAPORNWONGSA, T., STUARDO, C. A., SUMINTO, R. O., KE, H., LUKMAN, J. F., AND GUNAWI, H. S. Scalability bugs: When 100-node testing is not enough. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS'17)* (2017).

[24] LI, H., MA, M., LIU, Y., QIN, S., QIAO, B., YAO, R., CHATURVEDI, H., TRAN, T., CHINTALAPATI, M., RAJMOHAN, S., LIN, Q., AND ZHANG, D. Codec: Cost-effective duration prediction system for deadline scheduling in the cloud. In *Proceedings of the 34th IEEE International Symposium on Software Reliability Engineering* (2023).

[25] LI, M., MA, M., NIE, X., YIN, K., CAO, L., WEN, X., YUAN, Z., WU, D., LI, G., LIU, W., ET AL. Mining fluctuation propagation graph among time series with active learning. In *Database and Expert Systems Applications: 33rd International Conference* (2022).

[26] LI, Z., CHEN, J., JIAO, R., ZHAO, N., WANG, Z., ZHANG, S., WU, Y., JIANG, L., YAN, L., WANG, Z., ET AL. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service* (2021).

[27] LI, Z., LUO, C., CHEN, T.-H., SHANG, W., HE, S., LIN, Q., AND ZHANG, D. Did we miss something important? studying and exploring variable-aware log abstraction. *arXiv preprint arXiv:2304.11391* (2023).

[28] LIAN, X., CHEN, Y., CHENG, R., HUANG, J., THAKKAR, P., AND XU, T. Configuration validation with large language models. *arXiv preprint arXiv:2310.09690* (2023).

[29] LIU, D., HE, C., PENG, X., LIN, F., ZHANG, C., GONG, S., LI, Z., OU, J., AND WU, Z. Microhecl: High-efficient root cause localization in large-scale microservice systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'21)* (2021).

[30] LIU, H., LU, S., MUSUVATHI, M., AND NATH, S. What bugs cause production cloud incidents? In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS'19)* (2019).

[31] LIU, Y., ZHANG, X., HE, S., ZHANG, H., LI, L., KANG, Y., XU, Y., MA, M., LIN, Q., DANG, Y., ET AL. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022* (2022).

[32] LOU, C., CHEN, C., HUANG, P., DANG, Y., QIN, S., YANG, X., LI, X., LIN, Q., AND CHINTALAPATI, M. 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)* (2022).

[33] LOU, C., HUANG, P., AND SMITH, S. Understanding, detecting and localizing partial failures in large system software. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)* (2020).

[34] LUO, C., LOU, J.-G., LIN, Q., FU, Q., DING, R., ZHANG, D., AND WANG, Z. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014).

[35] MA, M., XU, J., WANG, Y., CHEN, P., ZHANG, Z., AND WANG, P. Automap: Diagnose your microservice-based web applications automatically. In *Proceedings of The Web Conference 2020* (2020).

[36] MA, M., YIN, Z., ZHANG, S., WANG, S., ZHENG, C., JIANG, X., HU, H., LUO, C., LI, Y., QIU, N., ET AL. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment (VLDB'20)* (2020).

[37] MA, M., ZHANG, S., CHEN, J., XU, J., LI, H., LIN, Y., NIE, X., ZHOU, B., WANG, Y., AND PEI, D. Jump-starting multivariate time series anomaly detection for online service systems. In *2021 USENIX Annual Technical Conference (ATC'21)* (2021).

[38] MA, M., ZHANG, S., PEI, D., HUANG, X., AND DAI, H. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE'18)* (2018).

[39] MASTROPAOLO, A., PASCARELLA, L., AND BAVOTA, G. Using deep learning to generate complete log statements. In *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)* (2022).

[40] MASTROPAOLO, A., SCALABRINO, S., COOPER, N., PALACIO, D. N., POSHYVANYK, D., OLIVETO, R., AND BAVOTA, G. Studying the usage of text-to-text transfer transformer to support code-related tasks. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE'21)* (2021).

[41] OPENAI. Tiktoken: A python library for tokenizing text. https://github.com/openai/tiktoken, 2023.

[42] SHETTY, M., BANSAL, C., UPADHYAYULA, S. P., RADHAKRISHNA, A., AND GUPTA, A. Autotsg: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'22)* (2022).

[43] SUN, X., CHENG, R., CHEN, J., ANG, E., LEGUNSEN, O., AND XU, T. Testing Configuration Changes in Context to Prevent Production Failures. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)* (2020).

[44] SUN, X., LUO, W., GU, J. T., GANESAN, A., ALAGAPPAN, R., GASCH, M., SURESH, L., AND XU, T. Automatic Reliability Testing for Cluster Management Controllers. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)* (2022).

[45] TAN, Y., MIN, D., LI, Y., LI, W., HU, N., CHEN, Y., AND QI, G. Evaluation of chatgpt as a question answering system for answering complex questions. *arXiv preprint arXiv:2303.07992* (2023).

[46] WEI, J., WANG, X., SCHUURMANS, D., BOSMA, M., CHI, E., LE, Q., AND ZHOU, D. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[47] WU, Y., CHEN, A., HAEBERLEN, A., ZHOU, W., AND LOO, B. T. Automated bug removal for software-defined networks. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)* (2017).

[48] XIE, Z., XU, H., CHEN, W., LI, W., JIANG, H., SU, L., WANG, H., AND PEI, D. Unsupervised anomaly detection on microservice traces through graph vae. In *Proceedings of the ACM Web Conference 2023* (2023).

[49] YAN, X., HSIEH, K., LIYANAGE, Y., MA, M., CHINTALAPATI, M., LIN, Q., DANG, Y., AND ZHANG, D. Aegis: Attribution of control plane change impact across layers and components for cloud systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering:*

Software Engineering in Practice (ICSE-SEIP'23) (2023).

[50] YUAN, D., LUO, Y., ZHUANG, X., RODRIGUES, G. R., ZHAO, X., ZHANG, Y., JAIN, P., AND STUMM, M. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)* (2014).

[51] ZENG, J., CHUA, Z. L., CHEN, Y., JI, K., LIANG, Z., AND MAO, J. Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics. In *Network and Distributed System Security Symposium (NDSS'21)* (2021).

[52] ZENG, J., WANG, X., LIU, J., CHEN, Y., LIANG, Z., CHUA, T.-S., AND CHUA, Z. L. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *2022 IEEE Symposium on Security and Privacy (S&P'22)* (2022).

[53] ZENG, Z., ZHANG, Y., XU, Y., MA, M., QIAO, B., ZOU, W., CHEN, Q., ZHANG, M., ZHANG, X., ZHANG, H., ET AL. Traceark: Towards actionable performance anomaly alerting for online service systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'23)* (2023).

[54] ZENG, Z., ZHANG, Y., XU, Y., MA, M., QIAO, B., ZOU, W., CHEN, Q., ZHANG, M., ZHANG, X., ZHANG, H., GAO, X., FAN, H., RAJMOHAN, S., LIN, Q., AND ZHANG, D. Traceark: Towards actionable performance anomaly alerting for online service systems. In *To appear in Proc. of ICSE* (2023).

[55] ZHANG, J., MYTKOWICZ, T., KAUFMAN, M., PISKAC, R., AND LAHIRI, S. K. Using pre-trained language models to resolve textual and semantic merge conflicts (experience paper). In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (2022).

[56] ZHANG, T., QIU, H., CASTELLANO, G., RIFAI, M., CHEN, C. S., AND PIANESE, F. System log parsing: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[57] ZHANG, X., XU, Y., QIN, S., HE, S., QIAO, B., LI, Z., ZHANG, H., LI, X., DANG, Y., LIN, Q., ET AL. 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* (2021).

[58] ZHANG, Y., GUAN, Z., QIAN, H., XU, L., LIU, H., WEN, Q., SUN, L., JIANG, J., FAN, L., AND KE, M. Cloudrca: a root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (2021).

[59] ZHANG, Y., YANG, J., JIN, Z., SETHI, U., RODRIGUES, K., LU, S., AND YUAN, D. Understanding and detecting software upgrade failures in distributed systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)* (2021).

[60] ZHANG, Z., ZHANG, A., LI, M., AND SMOLA, A. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations (ICLR'23)* (2023).

[61] ZHAO, C., MA, M., ZHONG, Z., ZHANG, S., TAN, Z., XIONG, X., YU, L., FENG, J., SUN, Y., ZHANG, Y., ET AL. Robust multimodal failure detection for microservice systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2023).