Large Language Models Can Provide Accurate and Interpretable Incident Triage

Zexin Wang[†‡], Jianhui Li[†], Minghua Ma[§*], Ze Li[§], Yu Kang[§], Chaoyun Zhang[§], Chetan Bansal[§]
Murali Chintalapati[§], Saravan Rajmohan[§], Qingwei Lin[§], Dongmei Zhang[§], Changhua Pei[†¶], Gaogang Xie[†‡]
[†]Computer Network Information Center, Chinese Academy of Sciences [‡]University of Chinese Academy of Sciences
[§]Microsoft [¶]Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences

***Abstract*—Large-scale cloud services frequently experience incidents that can have a significant impact on their stability. Incident triage is a critical process that assigns incidents to dedicated teams for resolution. However, traditional rule-based methods, commonly employed in various systems, have limitations due to a finite set of rules that necessitate continuous updates, leading to suboptimal performance. Current state-of-the-art approaches primarily rely on textual information, utilizing classifiers or unsupervised clustering. Unfortunately, the abundance of textual information, combined with considerable noise, presents a significant challenge to the accuracy of these methods. To tackle these challenges, we introduce COMET, an innovative system that utilizes an AutoExtractor to filter out non-critical logs and employs a Large Language Model (LLM) for keyword extraction. This approach effectively mitigates the complexity arising from disordered textual information. Additionally, COMET incorporates significant domain knowledge during keyword extraction, enhancing the LLM's comprehension of the text. We deployed COMET on multiple cloud services within Microsoft, where it has operated continuously for over six months. Offline and online evaluations have shown that COMET achieves enhanced accuracy and reduced Time to Mitigation (TTM).**

***Index Terms*—Incident triage, Large language model, Keywords extraction**

I. Introduction

A large-scale cloud service system typically consists of multiple modules. Incidents such as hardware failures and network issues frequently occur within these modules due to system updates, iterative improvements, user errors, or engineer misconfigurations [1]–[6]. If these incidents are not promptly addressed, they can result in severe consequences and significant economic losses. To ensure the stability of large-scale service systems, major corporations have implemented incident-handling systems. Fig. 1 illustrates the incident handling process for some of Microsoft's services. Initially, automated monitors [7]–[13] promptly report new incidents upon detecting faults. The second step involves incident triage, where a specific team is assigned to identify the root causes and resolve associated issues. Currently, these services employ automated decision trees with predefined rules to determine the appropriate team. In case of an incorrect initial team assignment, engineers from different teams engage in discussions and reassignment. The third step involves incident mitigation and resolution. Once an incident is correctly assigned to the appropriate team, they can promptly identify the root cause and take measures to recover from the fault. This framework ensures a systematic and efficient response to incidents within large-scale service systems, minimizing downtime and mitigating potential impacts on users and the overall system.

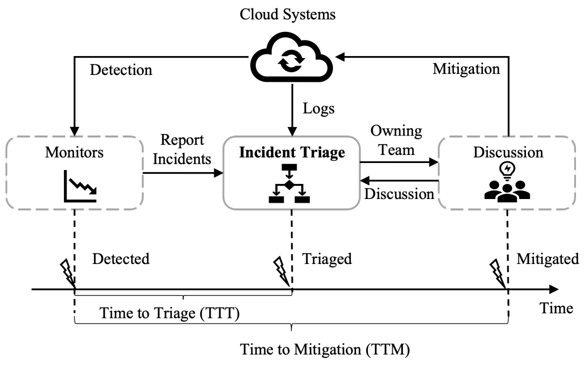


Fig. 1. The overview of incident management within Microsoft.

Incident triage plays a crucial role in the overall framework and significantly impacts the Time to Mitigation (TTM) of incidents [14]. However, the current deployment of numerous rules falls short of providing comprehensive coverage for all possible scenarios. Moreover, the constant need for rule updates, especially when encountering novel situations, imposes a significant burden on manpower. These limitations contribute to a relatively low accuracy in the initial incident triage, necessitating collaborative discussions among engineers from different teams for reassignment. Unfortunately, such engineers' discussions consume a substantial amount of time, which is unacceptable for incidents of significant importance.

Therefore, the accuracy of incident triage is vital in minimizing the TTM. Several approaches [15]–[21] have been developed to enhance the accuracy of incident triage, including classifier-based [15]–[19] and clustering-based [19] methods. These techniques primarily rely on textual information, which is the most crucial aspect within incidents.

Challenge 1: Which textual information is the most important for triage: logs, discussions, or others? In our context, textual information primarily consists of logs and

*Corresponding author. Email: minghuama@microsoft.com discussions among engineers following an incident. However, neither of these sources is suitable for direct involvement in incident triage. Firstly, as a cloud service serving a large user base, our system generates a vast amount of logs. Swiftly and accurately matching log templates is challenging, let alone effectively utilizing this extensive information for subsequent models. Especially when dealing with lengthy texts, even the most efficient language models can introduce significant comprehension biases. Therefore, using log texts directly as input for our models is considered inappropriate. Secondly, discussions require a significant amount of time to accumulate as they are crafted by engineers based on their domain expertise. This temporal requirement contradicts the objective of promptly resolving incidents. Moreover, discussions often contain substantial noise, such as abbreviations or hyperlinks. Consequently, using logs or discussions directly in incident triage is not suitable.

Challenge 2: How do we extract the key points from massive textual information? Effectively utilizing obtained textual information can be challenging. Many methods simply convert the entire text into vectors using embedding model, which is not suitable in many situations. This approach may lead to the loss of important information, especially when dealing with lengthy texts. Therefore, it is necessary to extract key points from a large volume of textual information. While many methods [22] attempt to summarize the text, generative summarization introduces randomness and irrelevant words, resulting in inconsistent summaries for the same text, which is unacceptable for incident triage. Furthermore, even a summary of an incident may still contain hundreds of words, making it easy for important information to get lost among unrelated words when using embedding model. Consequently, extracting key points from vast information poses a critical challenge.

Challenge 3: How can we integrate domain knowledge into the model? In the domain of incident triage, engineers heavily depend on domain knowledge, even when aided by automated models. This knowledge encompasses the definitions of specific terms and domain-specific documents. However, current text-based methods fail to incorporate this knowledge, leading to limited comprehension abilities of the models when dealing with textual information. For instance, the model's limited comprehension of certain word abbreviations often leads to their inadvertent exclusion when extracting key points. However, these abbreviated terms are precisely the ones of interest to engineers. Consequently, the accuracy of triage is significantly compromised.

In this paper, we present COMET (code name of our designed triage system), an innovative incident triage system based on Large Language Models (LLMs). Initially, we employ AutoExtractor to process a substantial amount of log data and perform template matching. Subsequently, by applying specific rules, we filter out irrelevant logs to generate a relevant textual information—TrimmedLogs (detailed in III-A2). Then, utilizing LLMs, we extract keywords from the TrimmedLogs, incorporating a significant amount of domain knowledge to enhance LLMs' comprehension of the text. Finally, we utilize the generated keywords to fine-tune a pre-trained FastText model [23], [24] and generate embeddings for incidents. During the online phase, the system calculates the similarity between the embeddings of historical incidents and the embedding of the new incident to retrieve similar historical incidents. The teams associated with these incidents are then predicted as the final assignment.

The paper's main contributions are as follows:

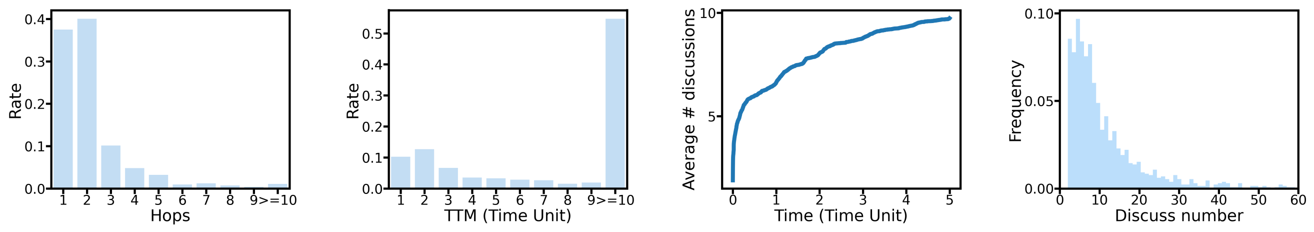
- We use AutoExtractor to collect relevant information for incident triage. This involves gathering multiple logs and using a selection mechanism to generate optimal textual information for extracting keywords.
- We pioneer using keywords for incident triage, leveraging LLMs for keyword extraction, and incorporating domain knowledge during the extraction process.
- We have deployed COMET on two large cloud services at Microsoft over the last six months. Online experiments have demonstrated that COMET improves accuracy by 30% and reduces the TTM by 35%.

II. Background

In this section, we present a comprehensive empirical study conducted on two large-scale services within Microsoft to shed light on the challenges associated with incident triage and underscore the motivation behind COMET. These two services primarily provide users with virtual machines. We collected a year's worth of data from these services to conduct the empirical study.

A. Incident Triage at Microsoft

As previously mentioned, ensuring the initial accuracy of incident triage is crucial, as incorrect triage leads to repeated reassignment. Delayed triage of high-priority incidents can increase the TTM, posing security risks to the system and negatively impacting the user experience. Fig. 2(a) illustrates that incidents requiring only one triage hop (triage hop refers to assigning an incident to the appropriate team for resolution) account for no more than 40%, indicating a significant portion of incidents do not receive accurate initial triage outcomes. While most incidents can be resolved within three hops, 12% of incidents exceed this threshold. It is important to note that in practical operational scenarios, engineers handle a substantial volume of incidents daily, and each hop may require a significant amount of time. Fig. 2(b) also supports our assertion about time costs, as it shows that less than 10% of incidents are resolved within a single time unit, indicating a prolonged resolution process for a significant number of incidents. Incidents can generate an overwhelming number of logs, sometimes reaching the thousands or more. This abundance of information can impede the swift identification of the root cause even with engineers' extensive domain knowledge. Moreover, if the triage process is incorrect, engineers from the new team will have to repeat the same steps, resulting in a significant time overhead. Additionally, incident triage is often a complex task that requires substantial involvement of



(a) The distribution of hops (b) The distribution of TTM (c) Average # discussions (d) The distribution of # discussions

Fig. 2. Analysis of incident triage characteristic

engineers, including communication and collaboration, which further consumes time. Consequently, we observe that 50% of incidents are resolved after surpassing 10 time units, which is considered unacceptable for incidents of any security level.

Lesson 1. The crucial aspect in minimizing the incident triage TTM resides in the reduction of the number of necessary steps to accurately assign incidents to the appropriate team. Ensuring precise triage to the correct team right from the beginning would yield a substantial multiplier effect.

B. Textual Information Comparison

During the incident triage process, textual information, including discussions and logs, plays a crucial role in determining the appropriate team to handle the incident. However, effectively and accurately utilizing this textual information presents a significant challenge. DeepCT [17] proposes the use of discussions for continuous triage. However, as shown in Fig. 2(c), the number of discussions increases over time. After 1 unit of time, the average number of discussions is only 6. This necessitates continuous waiting for engineers to provide additional details to gather sufficient information, which contradicts the objective of promptly resolving incidents. Fig. 2(d) illustrates the distribution of discussions for resolved incidents, indicating that a significant portion of incidents require 5 or even 10 discussions to reach a resolution. Considering that generating 10 discussions on average takes 5 units of time according to Fig. 2(c), this timeframe is considered unacceptable for incident triage. Furthermore, the quality of discussions varies widely. Initial discussions are predominantly generated by machines and offer limited assistance in incident triage. Discussions generated by engineers are often disorganized. Some discussions may consist of only a few phrases, an image, or a link, making them challenging for both humans and models to comprehend.

The utilization of logs for incident triage is a prevalent approach, primarily due to the critical fault information they often contain. However, logs encounter several challenges. Firstly, in large-scale cloud services within Microsoft, the volume of logs generated can be enormous, with millions of log entries produced in just one second [25]. Additionally, these logs' templates constantly evolve as cloud systems are updated. Many existing log processing tools [26]–[29] struggle to handle the massive and continuously changing templates. Therefore, a significant challenge lies in efficiently and accurately performing template matching on logs. Secondly, log data often includes numerous irrelevant entries that are generated by related software deployed across all teams. These entries provide little assistance in incident triage and can introduce noise interference. While most embedding methods like BERT can handle a large context, the longer the text information, the greater the risk of important information being overshadowed by irrelevant details. Even in advanced language models like GPT-4, this issue has not been fundamentally resolved. Consequently, extracting relevant information from a large volume of logs pose another significant challenge.

To address these challenges, we propose the utilization of TrimmedLogs as a solution (detailed in section III-A2). To validate the effectiveness of TrimmedLogs, we conducted experiments using the state-of-the-art (SOTA) method DeepCT [17] as a baseline. Specifically, we compared the performance of TrimmedLogs and the discussions used in the original DeepCT paper on two large-scale cloud services within Microsoft. Apart from the difference in input textual information, the experimental settings were kept the same. The results presented in Table I demonstrate that, regardless of considering metrics such as ACC@1 and ACC@5 (detailed in (1)), the utilization of TrimmedLogs consistently outperforms the use of discussions.

$$ACC@N = \frac{Sum(Correct\ Team\ in\ Top\ N\ Teams)}{Test\ Size} \quad (1)$$

TABLE I
ACCURACY OF THE TEST DATA.

Textual Information	ACC@1	ACC@5
Discussions	0.54	0.76
TrimmedLogs	0.57	0.78

Lesson 2. Discussions and logs are deemed unsuitable for direct utilization as textual inputs for models. However, filtered log data demonstrates greater suitability.

C. Textual Information Compression

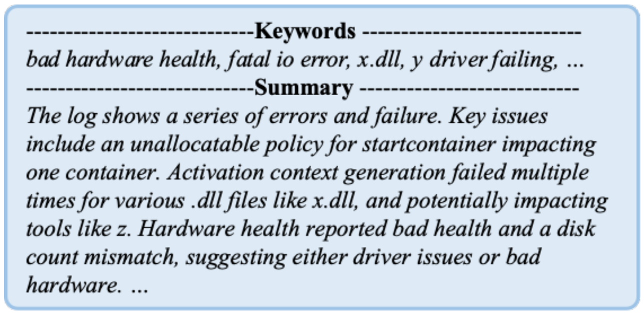

-----------------------------**Keywords** ---------------------------
bad hardware health, fatal io error, x.dll, y driver failing, ...
-----------------------------**Summary** ---------------------------
The log shows a series of errors and failure. Key issues include an unallocatable policy for startcontainer impacting one container. Activation context generation failed multiple times for various .dll files like x.dll, and potentially impacting tools like z. Hardware health reported bad health and a disk count mismatch, suggesting either driver issues or bad hardware. ...

Fig. 3. Keywords and summary of an incident

While TrimmedLogs is utilized to extract crucial information from logs that reflects the root causes of incidents, it often consists of several thousand words. Even with SOTA embedding methods, extended text lengths still result in significant information loss. Importantly, only a fraction of the TrimmedLogs contains the key information that engineers are concerned with, making it susceptible to loss during the embedding process. As a result, the embeddings may not accurately represent incidents, compromising accuracy despite subsequent remedial methods. Therefore, it becomes imperative to compress or extract key points from TrimmedLogs.

Currently, prevalent extraction methods involve using summaries [22] or keywords [30] to represent the original text. Extraction methods for summaries and keywords primarily fall into the categories of statistical extractive and generative approaches. However, statistical extractive methods like Term Frequency-Inverse Document Frequency (TF-IDF) prove unsuitable for our scenario, as high word frequency does not necessarily indicate the importance of information. On the contrary, keywords that appear infrequently and are specific to certain teams are often more important. We provide a more detailed discussion on the use of TF-IDF in section VI-A. Generative approaches, on the other hand, face challenges in the incident triage domain due to limitations in model comprehension. LLMs, with their strong understanding of textual information, have emerged as effective solutions to these challenges and have been promptly employed in various extraction tasks [30].

TABLE II
RESULTS OF USING KEYWORDS OR SUMMARY.

	ACC@1	ACC@5
Keywords	0.65	0.82
Summary	0.22	0.55

In the context of incident triage, while summaries effectively communicate the entire incident process to engineers, specific keywords are the true facilitators of incident triage. These keywords may pertain to specific deployment-related components or indicate certain hardware malfunctions. Fig. 3 illustrates the results obtained by employing LLMs to extract keywords and a summary from the TrimmedLogs of an incident. We can observe that, while summaries may be more readable for engineers or individuals outside the domain, the truly pivotal elements are specific keywords. Furthermore, summaries often contain numerous connective words added for coherence, introducing randomness in generative text. These factors can significantly impact the model's ability to make accurate judgments. To further validate our conclusion, we conducted experiments with our model using keywords and summaries for the two services mentioned earlier, and the results are presented in Table II. It is evident that the use of keywords yields significantly better results compared to summaries.

Lesson 3. Keywords are deemed a more appropriate approach for extracting key points in comparison to summaries. This preference arises from the fact that keywords possess a lower degree of randomness than summaries and are better equipped to concentrate on the essential information that engineers prioritize.

III. APPROACH

The architecture of COMET is depicted in Fig. 4, consisting of two phases: the offline phase and the online phase. Overall, COMET analyzes the status of the virtual machine where the incident occurred, along with logs and other relevant data, to ultimately determine the team responsible for the incident, as well as provide key insights and a summary of the incident.

A. Offline Phase

1) AutoAnalysis: In the absence of external tools, triaging a new incident to the appropriate team becomes challenging when the underlying root cause is unknown, as different teams specialize in specific domains, such as networking or hardware-related issues. To address this challenge, we have implemented AutoAnalysis in our services. AutoAnalysis is a rule-based decision tree, depicted in Fig. 1, that utilizes the historical experience of engineers to define rules for identifying the ultimate root cause of incidents. The input for AutoAnalysis is the status of the virtual machine where the incident occurred, and the output is a text explanation of the root cause of the incident. For instance, in Fig. 5, the HINodeTriage (a rule-based system for incident triage) module is invoked when incidents exhibit an unhealthy state. It assigns a team based on the results obtained from HostRCA (a rule-based system for root cause analysis), which systematically assesses whether the issue lies with hardware or the operating system. If the root cause is not identified after traversing all nodes, HostRCA returns an "Inconclusive" result. Subsequently, HINodeTriage determines the final team assignment based on the results from HostRCA.

Discussion about AutoAnalysis. However, as discussed in section I, the current methodology is considered inaccurate due to the constant emergence of new incidents, which require continuous updates to the rules. This places a burden on

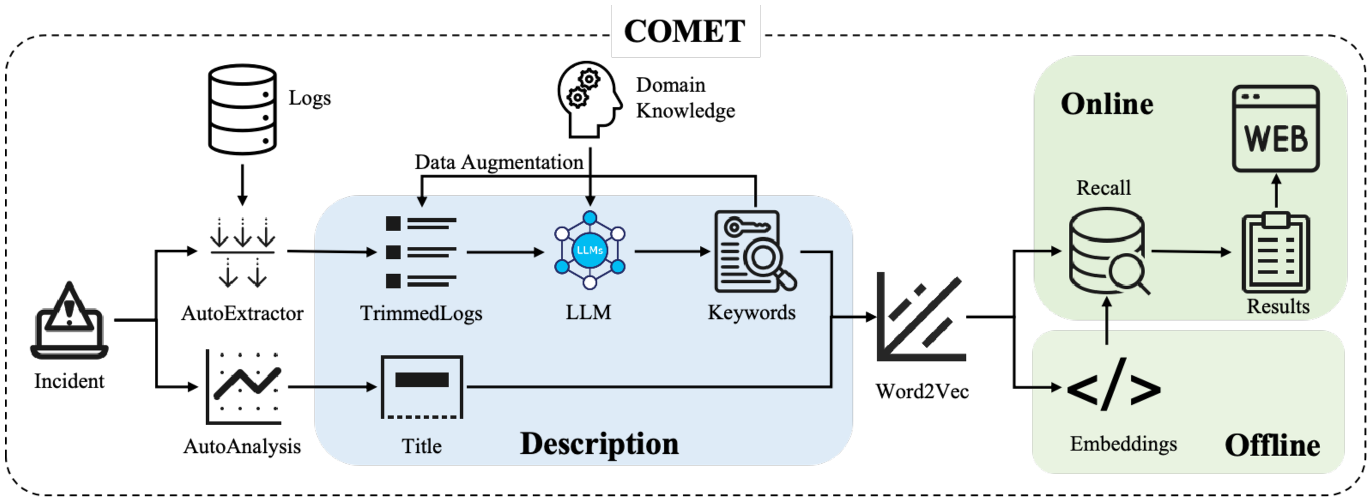


Fig. 4. Architecture of COMET

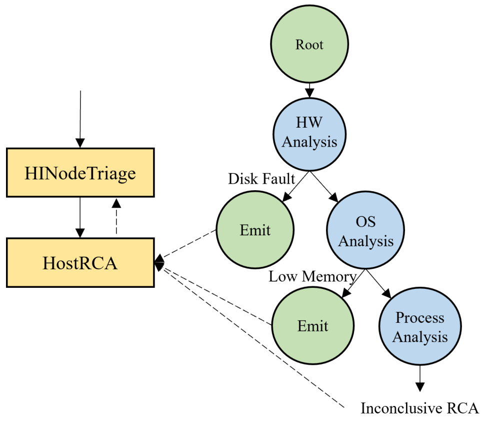


Fig. 5. A sample of AutoAnalysis

engineers and lacks scalability when transitioning to new domains. Additionally, when rules fail to pinpoint a specific root cause, the outcome is marked as "Inconclusive," which is common in real-world scenarios and significantly diminishes triage accuracy. Nevertheless, these rules incorporate extensive domain knowledge and offer intrinsic value despite occasional inaccuracies. Therefore, we utilize the results from AutoAnalysis as incident titles for subsequent processes. Our experiments in section V-C further demonstrate that even if the results provided by AutoAnalysis have some deviations, they can still be used in conjunction with keywords to achieve better results.

2) *AutoExtractor:* When incidents occur, a significant amount of diverse log information is collected and consolidated using intelligent analytic services, as described in [31]. These services provide insights into incident-related information from various perspectives across different stacks, including software agents on the host, network traffic, host/guest OS logs, resource utilization logs, and hardware logs. The integration of this diverse log information contributes to a more comprehensive understanding of incidents, serving as the fundamental basis for incident data analysis. Below is an introduction to some commonly used log categories in COMET.

•**Change Management Logs (CML).** Software changes are important fault information that some operation and maintenance personnel are very concerned about.
•**Crash Logs (CL).** Crash logs can provide real-time kernel and user mode crashes with call stack attribution.
•**System Event Logs (SEL).** SEL records most server-related events, such as over and under voltage, temperature events, fan events, and events from BIOS.
•**Windows/Linux System Events logs (WL-SEL).** WL-SEL contains the Windows/Linux host os/guest os logs.
•**Container Management Logs (CTML).** CTML contains the life cycles of containers/VMs with associated system logs.

To extract useful information from a large quantity of logs, we proposes AutoExtractor. The architecture of AutoExtractor is depicted in Fig. 6.

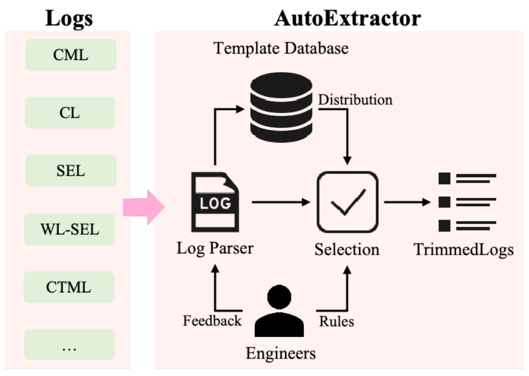


Fig. 6. Architecture of AutoExtractor

Log Parser. After an incident occurs, we collect a large amount of diverse raw log information from the database using Kusto functions. However, due to the complex formatting of these logs, they pose challenges for both experts and LLMs to comprehend. This complexity has led to the development of various log parsers aimed at extracting templates. In our specific context, the challenge arises from different log categories corresponding to different templates. Moreover, as the system undergoes continuous updates, new templates emerge. Additionally, the sheer volume of log entries burdens the system if not processed in parallel. As a result, many existing log parsers [28], [32]–[35] are deemed unsuitable for our purposes. To address these challenges, we implemented a log parser similar to [36] in our cloud services. The log parser incorporates clustering and scheduling for different log groups, enabling fast and accurate processing of large quantities of logs. Moreover, it incorporates a feedback mechanism to ensure accurate parsing even when confronted with evolving log data.

Selection Mechanism. Processing large quantities of log data not only consumes significant time but also risks drowning out crucial information. Additionally, when leveraging LLMs, token size limitations become a constraint that must be considered. To tackle this issue, we have adopted a unique selection mechanism. This mechanism assigns different priorities to various log entries based on two criteria. Firstly, we synthesize domain knowledge from operations personnel to formulate rules. Log entries that adhere to these rules are assigned higher priority, as operations personnel can discern which types of messages hold greater importance. For example, we predefine certain keywords such as "crash" and "hardware issue." Logs containing these keywords are not discarded. Secondly, we employ a TF-IDF-like approach, where log entries that frequently occur across teams are assigned lower priority, while those that appear infrequently and possess uniqueness are assigned higher priority. This approach allows us to extract concise yet crucial log information, referred to as the "**TrimmedLogs**".

In summary, TrimmedLogs represents a subset of log entries that have undergone our selection process. Typically, post-selection log entries are limited to only a few dozen, alleviating the challenge of dealing with an abundance of text exceeding the LLM's maximum token size. This subset of log entries encompasses information of significant interest to operations personnel or contains unique key information indicative of event categories. The TrimmedLogs is subsequently utilized in downstream processes for incident triage.

3) Extracting Keywords:

TrimmedLogs Preprocessing. In the initial stage, we retrieve the title and TrimmedLogs for each incident from the database using automated tools. However, the TrimmedLogs often contains a significant amount of extraneous information, which can consist of thousands of words. This extraneous information includes website links or image URLs, which not only pose challenges for the comprehension of LLMs but also consume a considerable number of tokens. This consumption may potentially limit the input of more pertinent information into LLMs. To address this issue, we perform a filtering process on the TrimmedLogs before extracting keywords. This process eliminates website links, image URLs, and irrelevant identifiers such as container IDs. By doing so, we streamline the TrimmedLogs and remove unnecessary elements, enabling a more focused and efficient analysis.

TABLE III
PROMPTS DESIGNED BASED ON DOMAIN KNOWLEDGE

1. Pay attention to keywords that indicate the occurrence of failure or error, and identify what specifically failed or caused the error.
2. Take note of fault codes, event IDs, and other codes that indicate fault information, and give them special attention.
3. When encountering common computer terms such as FPGA, identify what specifically is going wrong.
4. Be particularly attentive to tools and deployments.
5. Avoid using commonly used words in keywords.

Introducing Domain Knowledge. After the preprocessing stage, we input the TrimmedLogs into LLMs while adjusting the prompt to guide the extraction of keywords. However, since LLMs are trained on a corpus of all natural languages, their understanding of specific domains may not be as robust as expected. Our experimental results in section V-C further confirm that LLMs' performance is suboptimal without incorporating domain-specific knowledge. To address this limitation, there are two main approaches for introducing domain knowledge into LLMs: fine-tuning LLMs [37] or incorporating additional knowledge into the prompt [38], [39]. However, fine-tuning LLMs for the field of incident triage incurs significant additional costs. Therefore, we propose a novel approach of **introducing domain knowledge into the prompt** to enhance LLMs' ability to extract keywords in the incident triage domain. While introducing additional knowledge to lengthen prompts is a common practice, in keyword extraction, current methods treat it as a generic task across all domains by directly instructing the LLM to extract keywords. However, this approach often overlooks crucial task information. Determining which words are crucial and which ones should be disregarded varies significantly across different domains in keyword extraction tasks. For instance, while "crash" holds importance in the operation domain, it is inconsequential in the legal domain. Therefore, it is imperative to impart domain-specific task information to the LLM, which constitutes our innovative approach. In conclusion, this approach allows us to leverage existing domain knowledge without the need for extensive fine-tuning, resulting in improved keyword extraction performance.

Domain Knowledge. The domain knowledge introduced in our approach is outlined in Table III, derived from the analysis of numerous cases. Firstly, prompt 1 is formulated based on the understanding that engineers typically prioritize fault-related information amidst the diverse content in the TrimmedLogs. This prompt helps guide LLMs towards extracting keywords related to faults or issues. Secondly, prompt 2 addresses the presence of critical codes or IDs in the TrimmedLogs that signify errors. Without explicit attention, LLMs might overlook important information surrounding these codes or

IDs. This prompt ensures that LLMs focus on extracting keywords associated with these critical identifiers. Prompt 3 is established based on the observation that incidents often involve specific hardware components. By incorporating this prompt, LLMs can extract keywords related to the relevant hardware components, providing valuable context for incident triage. Prompt 4 is derived from the observation that during an incident occurrence, there are often automated mitigation measures or troubleshooting procedures. By considering this prompt, LLMs can extract keywords related to these procedures. Finally, prompt 5 is devised to prevent the inclusion of confusing keywords. By informing LLMs about commonly recurring terms, they can avoid unnecessary focus on these terms, allowing for more accurate keyword extraction.

Alleviating hallucination. Although LLMs are susceptible to hallucination, even with the prompts mentioned above, domain-specific TrimmedLogs still presents challenges for LLMs. To facilitate improved downstream tasks for LLMs, we employ two strategies. Firstly, we provide explanations for certain domain-specific terms, such as specific word abbreviations that exist exclusively within the current domain and are difficult for LLMs to comprehend. Secondly, we convey the structural information of the TrimmedLogs to LLMs. The TrimmedLogs we generate consists of a list of individual entries, each formatted as a JSON object, encompassing various details such as the current log entry category. Without guidance on the structure, LLMs may struggle to comprehend the TrimmedLogs accurately. Lastly, we inform LLMs about the specific meanings of frequently occurring log categories (detailed in section III-A2) within the TrimmedLogs. These log categories hold important information regarding the incidents. These strategies collectively enhance LLMs' ability to comprehend and extract keywords from domain-specific TrimmedLogs, enabling more accurate and effective incident triage in the field of incident management.

4) Training Embedding Model:

Embedding Model Selection. After extracting keywords, many methods commonly employ direct keyword matching [19]. However, in our scenario, this approach is not feasible due to the substantial volume of logs generated in incidents. Certain words within the logs, although distinct in form, may carry similar meanings in this domain. For example, expressions like "fail to start," "fail while starting," and "fail to initialize" exhibit significant lexical differences but share the common implication that a particular container or functionality failed to initiate. Most embedding methods are pretrained on extensive corpora from the natural world, causing semantically related terms to have close embeddings. This aligns well with our requirements. We choose FastText here and a comparative analysis of various embedding methods will be presented in section VI-B. However, since many keywords are domain-specific, using a pretrained model directly may introduce significant biases in the embeddings of these words. Therefore, we leverage the extracted keywords to fine-tune the pretrained model, resulting in the ultimate model for our application.

Data Augmentation. When fine-tuning the embedding model and recalling incidents, an imbalanced distribution of incidents among different teams can introduce biases in the results. Therefore, data augmentation becomes necessary. In contrast to traditional sampling-based data augmentation methods, we propose a novel data augmentation approach based on LLMs. By adjusting various parameters of LLMs, such as temperature, we perform multiple extractions on incidents with fewer occurrences, aiming to enhance diversity in the dataset.

Deriving Embeddings. We employed the model trained to convert all keywords into vectors. Subsequently, we obtained the incident embedding by averaging all keyword vectors and incorporating the vector corresponding to the incident's title.

B. Online Phase

In the online phase, when a new incident occurs, we obtain embeddings by following the steps outlined in the offline phase. Subsequently, we calculate the similarity between the embeddings of the new incident and historical incidents using (2) with support from the *Faiss* package. Based on the similarity rankings, we recall incidents with higher similarity, and we select the team affiliation of the top-ranking similar incidents as our predictive result. By employing this well-established recall process commonly used in recommendation systems [40], we not only effectively address the classifier's limitations in handling class imbalance scenarios but also leverage the characteristic that similar incidents typically belong to the same team. Since incident triage already achieves high precision in recalling incidents that meet the relevant service requirements, we omit subsequent reranking to minimize additional time and resource consumption.

$$\begin{aligned} Similarity &= \frac{1}{1 + Distance(X, Y)} \\ Distance(X, Y) &= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \end{aligned} \tag{2}$$

IV. Online Deployment Status

In the context of a real-world online incident management system, each incident is reported to the engineers through an entry on a dedicated website that contains descriptive information, including a title and discussions. To enhance the system's functionality without disrupting normal operations, we implemented specific improvements. Specifically, as shown in Fig. 4, we adjusted the output of AutoAnalysis to serve as the incident title and included the TrimmedLogs generated by AutoExtractor as a discussion entry on the website. Additionally, we provided the model's final predictions for the team, similar incidents, and keywords, along with a summary of the TrimmedLogs on the website. We recognized that engineers, who are more concerned with underlying causes, may require more than a simple result. Therefore, the keywords and summary were designed to facilitate a rapid understanding of the entire incident process. The inclusion of similar incidents not only provided evidence supporting the correctness of our predicted teams but also offered valuable references for effective mitigation measures.

To evaluate the effectiveness of our proposed approach, we first conducted offline experiments (the results of which are described in section V). After COMET was effectively validated, we proceeded with online deployment and verification. Specifically, we collected incidents that occurred within a one-month timeframe on the services, all of which were successfully resolved. We compared the teams predicted by COMET with the actual teams to derive accuracy metrics. Additionally, we assessed changes in the TTM for the incidents before and after deploying COMET. The results obtained are presented in Table IV. The baseline method refers to the rule-based method previously deployed online. It does not have an ACC@5 metric because it only recommends a single team at most. COMET not only outperformed the baseline in terms of accuracy but also in terms of TTM.

TABLE IV
ONLINE RESULTS, TU MEANS TIME UNIT, FTT MEANS FIRST TRIAGE TIME.

	ACC@1	ACC@5	FTT	TTM
Baseline	0.47	-	7.85TU	-
COMET	0.61	0.88	1.30TU	-35%

In addition to accuracy, temporal efficiency is also a crucial aspect in incident triage. The timely initiation of incident triage directly impacts the prompt resolution of incidents. However, the currently deployed baseline methods, which rely on rules, are vulnerable to situations where the rules may not cover all cases, resulting in prolonged triage times. This observation is supported by the cumulative distribution function (CDF) plot of triage time for the baseline method (Fig. 7). The average triage time for services even reaches 7.85 time units. In contrast, our proposed method achieves an average triage time of only 1.30 time units, with a significant portion of the time attributed to the latency incurred by invoking the LLM.

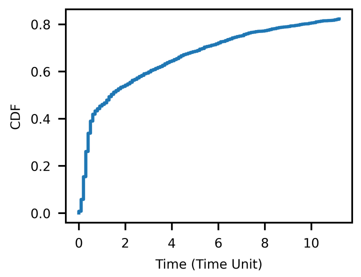
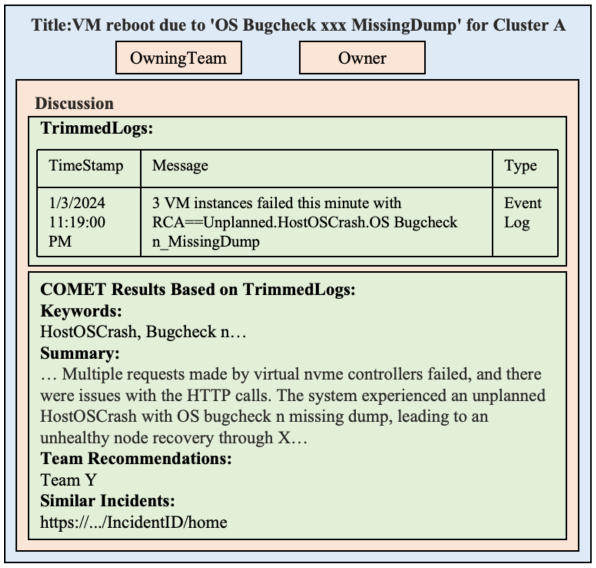


Fig. 7. CDF of baseline method TTT, where TTT means Time to Triage.

Implementation Details. We deploy COMET on two large-scale cloud services within Microsoft, leveraging the Azure web app framework. These services primarily serve users by provisioning virtual machines. Upon the generation of a new incident, the pipeline triggers the invocation of the web app, facilitating the delivery of the pertinent outcomes. In COMET, we use GPT-3.5 because it performs similarly to GPT-4 but at a significantly lower price. We make two calls to the LLM during the generation of keywords and summary. This practice is motivated by the consideration that when the input text is lengthy and multiple tasks are expected from the LLM, there is an inherent risk of misinterpretation.

Safety of Online Status. Security is a crucial component of any system. COMET merely adds elements such as predicted teams to the incident website as discussions, without making any structural changes to the underlying system. Even in the event of prediction errors, subsequent engineers can utilize domain knowledge to determine whether to utilize the recommended team.


Title:VM reboot due to 'OS Bugcheck xxx MissingDump' for Cluster A

OwningTeam | Owner

Discussion

TrimmedLogs:

TimeStamp	Message	Type
1/3/2024 11:19:00 PM	3 VM instances failed this minute with RCA==Unplanned.HostOSCrash.OS Bugcheck n_MissingDump	Event Log

COMET Results Based on TrimmedLogs:
Keywords:
HostOSCrash, Bugcheck n…
Summary:
… Multiple requests made by virtual nvme controllers failed, and there were issues with the HTTP calls. The system experienced an unplanned HostOSCrash with OS bugcheck n missing dump, leading to an unhealthy node recovery through X…
Team Recommendations:
Team Y
Similar Incidents:
https://.../IncidentID/home

Fig. 8. Overview of an incident's website

Case Study. Fig. 8 illustrates an incident website, where the occurrence of an OS crash in the TrimmedLogs is of particular interest to engineers dealing with incidents. By incorporating domain knowledge into the prompt, COMET successfully extracts this information in both the keywords and summary. The summary also provides a certain level of explanation regarding the causes and consequences of the OS crash, which proves highly beneficial to engineers. They no longer need to manually search through logs one by one, saving significant amounts of time. While the title generated by AutoAnalysis contains some useful information, it does not capture the complete details of the anomaly. In normal circumstances, engineers would have to consult relevant documentation to mitigate and resolve the incident. However, COMET simplifies this process by providing similar incidents, allowing engineers to gain insights into how similar incidents were resolved with just a click, thereby greatly reducing TTM.

V. Offline Evaluation

A. Experiment Settings

1) Datasets and Metrics: We collected a dataset spanning one year from two large service systems within Microsoft (same as in empirical study) to validate the effectiveness of COMET. We chose these two services because they serve millions of people, making them highly representative and able to demonstrate the generality of COMET. During the one-year period, thousands of incidents were resolved in these systems, involving over 70 distinct teams. The most recent 25% of incidents were used for testing. We evaluated our system using two metrics: ACC@1 and ACC@5, which are calculated as shown in (1).

2) Baseline Methods:

•**DeepCT [17]** DeepCT leverages gated recurrent unit (GRU) to capture temporal relationships within discussions while employing an attention mechanism to extract pertinent information from discussions using title and summary.

•**DeepTriage (DT) [19]** DeepTriage ensembles several submodels, including a multiple additive regression tree (MART) model, an light gradient boosting machine (LGBM) [41] model, an Inverted Index (II) model, a locality-sensitive hashing (SI) model, and a deep neural network (DNN) model.

B. RQ1: Overall Performance

The results presented in Table V demonstrate that COMET(GPT-4) outperforms the SOTA methods by 5% and 4% in ACC@1, as well as 10% and 6% in ACC@5, for the two evaluated systems. In the case of system A, where incidents are distributed across a diverse range of teams, introducing variability and complexity to the triage task, our method exhibits notable effectiveness. Similarly, system B, although relatively simpler, still benefits significantly from our approach. It is worth noting that our method achieves close to 100% ACC@5 for system B, indicating our ability to rapidly and accurately resolve each incident—a critical factor for system stability.

TABLE V
Top 1 and Top 5 accuracy of different approaches.

	System A		System B		All	
Approach	ACC@1	ACC@5	ACC@1	ACC@5	ACC@1	ACC@5
DeepCT [17]	0.17	0.47	0.81	0.90	0.56	0.74
MART	0.25	0.45	0.66	0.89	0.51	0.72
LGBM	0.18	0.45	0.64	0.89	0.47	0.72
II	0.01	0.17	0.11	0.12	0.04	0.15
SI	0.16	0.38	0.52	0.82	0.39	0.66
DNN	0.04	0.32	0.15	0.43	0.11	0.39
DT [19]	0.25	0.41	0.66	0.88	0.51	0.70
COMET(GPT-3.5)	**0.26**	**0.54**	**0.84**	**0.93**	**0.62**	**0.78**
COMET(GPT-4)	**0.30**	**0.57**	**0.85**	**0.96**	**0.65**	**0.82**

While DeepCT performs reasonably well, our experiments indicate that its effectiveness largely relies on the influence of incident titles. Despite the seemingly rational proposal of an attention structure based on discussions and GRU, the high noise level within discussions hampers the extraction of meaningful information. Additionally, the relationships among discussions often exhibit weak correlations, rendering GRU less effective. Although DeepTriage integrates multiple models, its performance is constrained by the limitations of its submodels. Moreover, the fusion of extensive textual information into a single vector introduces inherent biases.

TABLE VI
Ablation study, # 5 is COMET

#	Components				Metric	
	Embedding	Domain Knowledge	Data Augment	Title	ACC@1	ACC@5
1	-	-	-	-	0.36	0.70
2	✓	-	-	-	0.48	0.74
3	✓	✓	-	-	0.55	0.76
4	✓	✓	✓	-	0.60	0.78
5	✓	✓	✓	✓	**0.65**	**0.82**

C. RQ2: Ablation Study

The results of ablation study are presented in Table VI.

1) Effectiveness of Embedding Model: In method 1, no embedding model is utilized. Instead, keywords are extracted through LLMs, and a rule-based matching approach is employed. This method recalls incidents based on the maximum number of matched keywords and selects the teams of these incidents as the final predicted teams. On the other hand, in method 2, FastText is employed as the embedding method. Incidents are recalled based on the similarity of embeddings. By comparing the accuracy of method 1 and method 2, we observe that the use of FastText significantly improves accuracy, with increases of 12% and 4% for ACC@1 and ACC@5, respectively. This indicates that while the keywords extracted by LLMs may vary slightly in form, many of them carry similar meanings. For example,“error” and “fail”, although different, have closely related meanings. Basic matching algorithms struggle to capture such relationships, whereas embedding model, given our pre-trained model, can recognize these connections. Additionally, the embedding model has undergone fine-tuning in the current domain, providing embeddings with significant distinctiveness. Hence, embedding model proves to be indispensable.

2) Effectiveness of Domain Knowledge: In method 2, we employ a basic prompt for keyword extraction without incorporating any domain knowledge. However, in method 3, we integrate domain knowledge (detailed in section III-A3) to guide LLMs in better extracting keywords. The results presented in Table VI demonstrate that the inclusion of domain knowledge leads to an improvement of 7% for ACC@1 and 2% for ACC@5. This indicates that the input domain knowledge assists LLMs in understanding which words are relevant to our specific domain and ultimately aid in extracting more accurate keywords.

3) Effectiveness of Data Augmentation: In method 3, we don’t utilize data augmentation, whereas in method 4, we employ the data augmentation technique mentioned before. The results presented in Table VI indicate that ACC@1 and

ACC@5 improved by 5% and 2%, respectively. Data augmentation is an intrinsic capability of LLMs, which possesses the ability to introduce randomness while ensuring a certain level of accuracy. This feature aligns well with the requirements of many domains. Traditional data augmentation techniques, such as sampling, are mostly deterministic and may not introduce new random elements. In contrast, leveraging LLMs for data augmentation represents a significant trend, as they allow for the generation of diverse and realistic data samples.

4) Effectiveness of Title: In method 4, we do not utilize the title, whereas in COMET, the title is taken into consideration. The results presented in Table VI demonstrate significant improvements, with ACC@1 and ACC@5 increasing by 5% and 4%, respectively. These improvements clearly indicate the effectiveness of incorporating the title.

VI. Discussion

A. Noise in Keywords

1) Discussion about TF-IDF and Vocabulary: As discussed in section III-A2, our objective in generating TrimmedLogs is to select log categories with lower frequencies, aiming to avoid frequently occurring categories. However, it is inevitable that certain frequently occurring log categories still end up in the TrimmedLogs. These categories are spread across various teams and lack representativeness, introducing significant noise when extracting keywords from this subset of logs. To address this issue, we propose using TF-IDF to filter out noise-inducing keywords. Specifically, we organize the keywords from each team in the training samples into categories and rank them based on their TF-IDF values. We then eliminate keywords with lower rankings, which helps mitigate noise during training. However, this strategy poses challenges during real-time processing, as the same operations cannot be performed. To overcome this challenge, we adopt a practical solution. We assume that the keywords extracted during training are sufficiently comprehensive and represent a majority of the domain knowledge. In this context, we utilize all words from the training phase as the vocabulary. During online processing, we only extract keywords from this predefined vocabulary. However, this method may sacrifice diversity by potentially excluding terms that are crucial for team assignment but not present in the vocabulary. Therefore, caution must be exercised when using the vocabulary approach to avoid sacrificing the richness and diversity of the generated TrimmedLogs.

2) Experiment Results: To evaluate the effectiveness of TF-IDF and the vocabulary approach, we conducted detailed experiments divided into three sets with the same testing data used in section V. In the first set, we discarded different numbers of words during TF-IDF processing. In the second set, we used the vocabulary generated during training to select keywords during online processing, under the same conditions as the first set. In the third set, we incorporated title information into the experiments, again under the same conditions as the first set. The results of these experiments are presented in Fig. 9. The results of the first set of experiments

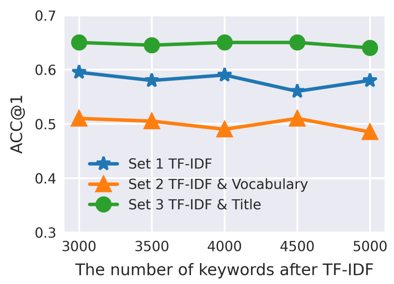


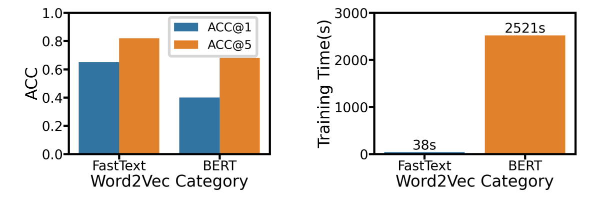
Fig. 9. ACC@1 with different settings

indicate that discarding a certain number of keywords through TF-IDF has a limited impact on the outcomes. This suggests that our approach of discarding frequently occurring categories in TrimmedLogs generation is effective. Additionally, when extracting incident embeddings, we employ average pooling across all keywords. Although some noise may be present in the keywords, the majority of them remain useful for team assessment, and the averaging process minimizes their overall impact. The results of the second set of experiments are noticeably worse than those of the first set, which reinforces the inference drawn from the first set. Since the first set already demonstrated minimal noise in the keywords, forcefully incorporating the vocabulary to eliminate a small portion of noise would result in an unacceptable loss of diversity. Furthermore, the superior performance of the third set of experiments compared to the first set provides additional evidence of the effectiveness of title information.

B. Embedding Models Comparison

As described in section III, we utilized FastText as our word embedding method. While there are more advanced embedding methods available, such as the widely used BERT, we found that BERT is not suitable for our specific use case. There are two main reasons for this. Firstly, BERT has a large number of model parameters, which results in significant memory and training costs. This can have a negative impact on online scenarios where efficiency is crucial. Secondly, there is a fundamental difference in how BERT and FastText handle relationships between words in text. BERT considers the relationships between individual words in the text, making it more effective for longer texts where the context is crucial. On the other hand, FastText applies n-gram operations to all texts and feeds the results into the model without explicitly focusing on the relationships between words. In our scenario, each of our keywords contains a limited number of words, and there is no need to consider intricate relationships between words. Therefore, FastText is more suitable for our specific use case. Using FastText not only saves time and memory but also avoids introducing noise from overly complex models, which verified by the experiment.

The results, as illustrated in Fig. 10, indicate a noticeable superiority of FastText over BERT in both ACC@1 and ACC@5 metrics. Moreover, the considerably longer training time of

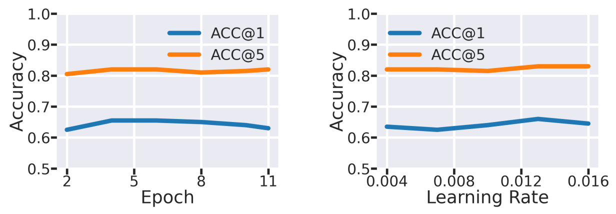


(a) Accuracy of different embedding models (b) Training time of different embedding models

Fig. 10. Comparison with FastText and BERT

BERT, attributed to its intricate network structure, renders it impractical for deployment in cloud services. Consequently, we opt for FastText as the embedding method.

C. Parameter Sensitivity



(a) Accuracy of different training epoches (b) Accuracy of different learning rate

Fig. 11. Accuracy for different parameters

A robust method should possess strong generalization capabilities, enabling it to adapt seamlessly to various domains. Parameter sensitivity is a critical aspect for evaluating robustness. To assess the robustness of our approach, we conducted sensitivity tests on different parameters using the same data used in V. The results of these tests are depicted in Fig. 11. The outcomes of the sensitivity tests validate the versatility of our approach.

VII. Related Work

Classifier-based methods. Classifier-based methods [15]–[19] typically involve constructing various classifier models by extracting textual and non-textual features to predict the corresponding team for incident triage. By constructing classifiers based on textual information, bugs can be accurately triaged to the correct team. For instance, SG [15] employs an ensemble of multiple base classifiers such as Support Vector Machines (SVM) to categorize bugs. CNN Triager [16] introduces Convolutional Neural Networks (CNN) for further precise processing of textual information. DeepCT [17], building upon CNN, considers temporal relationships among incident discussions. DeepTriage [19] integrates various classifier models, including popular ones like LGBM and MART.

Clustering-based methods. Clustering-based recall methods [19] posit that incidents or bugs with similar topics should be assigned to the same team. These methods initially cluster incidents with similar topics based on incident reports and then assign new incidents to the respective teams according to their topics. SI [19] converts incident text information into embeddings and identifies the incident with the closest embedding distance to the new incident.

LLMs in incident management. Recently, an increasing number of methods have endeavored to employ LLMs to address various issues in incident handling, such as RCACopilot [22], RCAgent [42], React Agent [43], MonitorCopilot [44], and Xpert [45].

VIII. Conclusion

In this paper, we propose a novel incident triage system called COMET, which is based on the extraction of domain-specific keywords using LLMs. We employ AutoExtractor to process logs. Then, we extract keywords using LLMs and enhance the keyword extraction process by incorporating domain knowledge into the LLM model. When online, we retrieve the most relevant results based on the similarity between embeddings generated by finetuned FastText model. Online evaluations have demonstrated that COMET improves accuracy by 30% and reduces the TTM by 35%.

IX. Acknowledgement

This work was supported by the National Key Research and Development Program of China (No. 2021YFE0111500), in part by the Chinese Academy of Sciences (No. 241711KYSB20200023), in part by the National Natural Science Foundation of China (No. 62202445), and in part by the State Key Program of the National Natural Science Foundation of China under Grant 62321166652.

References

[1] N. Zhao, J. Zhu, Y. Wang, M. Ma, W. Zhang, D. Liu, M. Zhang, and D. Pei, "Automatic and generic periodicity adaptation for kpi anomaly detection," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1170–1183, 2019.

[2] M. Ma, Y. Liu, Y. Tong, H. Li, P. Zhao, Y. Xu, H. Zhang, S. He, L. Wang, Y. Dang *et al.*, "An empirical investigation of missing data handling in cloud node failure prediction," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1453–1464.

[3] Y. Liu, H. Yang, P. Zhao, M. Ma, C. Wen, H. Zhang, C. Luo, Q. Lin, C. Yi, J. Wang *et al.*, "Multi-task hierarchical classification for disk failure prediction in online service systems," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3438–3446.

[4] L. Li, X. Zhang, S. He, Y. Kang, H. Zhang, M. Ma, Y. Dang, Z. Xu, S. Rajmohan, Q. Lin *et al.*, "Conan: Diagnosing batch failures for cloud systems," 2023.

[5] P. Dogga, C. Bansal, R. Costleigh, G. Jayagopal, S. Nath, and X. Zhang, "Autoarts: Taxonomy, insights and tools for root cause labelling of incidents in microsoft azure," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 359–372.

[6] M. H. Tong, R. L. Grossman, and H. S. Gunawi, "Experiences in managing the performance and reliability of a large-scale genomics cloud platform," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 973–988.

[7] M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai, "Robust and rapid adaption for concept drift in software system anomaly detection," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018, pp. 13–24.

[8] Q. Lin, T. Li, P. Zhao, Y. Liu, M. Ma, L. Zheng, M. Chintalapati, B. Liu, P. Wang, H. Zhang *et al.*, "Edits: An easy-to-difficult training strategy for cloud failure prediction," in *Companion Proceedings of the ACM Web Conference 2023*, 2023, pp. 371–375.

[9] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, and D. Pei, "Jump-starting multivariate time series anomaly detection for online service systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 413–426.

[10] Z. Zeng, Y. Zhang, Y. Xu, M. Ma, B. Qiao, W. Zou *et al.*, "Traceark: Towards actionable performance anomaly alerting for online service systems," in *45th ICSE-SEIP*. IEEE, 2023, pp. 258–269.

[11] Z. Wang, C. Pei, M. Ma, X. Wang, Z. Li, D. Pei, S. Rajmohan, D. Zhang, Q. Lin, H. Zhang, J. Li, and G. Xie, "Revisiting VAE for unsupervised time series anomaly detection: A frequency perspective," in *Proceedings of the ACM on Web Conference 2024*. ACM, 2024, pp. 3096–3105.

[12] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, and D. Pei, "Jump-starting multivariate time series anomaly detection for online service systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 413–426.

[13] Y. Chen, C. Zhang, M. Ma, Y. Liu, R. Ding, B. Li, S. He, S. Rajmohan, Q. Lin, and D. Zhang, "Imdiffusion: Imputed diffusion models for multivariate time series anomaly detection," *Proc. VLDB Endow.*, vol. 17, no. 3, pp. 359–372, 2023.

[14] X. Yan, K. Hsieh, Y. Liyanage, M. Ma, M. Chintalapati, Q. Lin, Y. Dang, and D. Zhang, "Aegis: Attribution of control plane change impact across layers and components for cloud systems," in *45th ICSE-SEIP*. IEEE, 2023, pp. 222–233.

[15] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, vol. 21, pp. 1533–1578, 2016.

[16] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proceedings of the 2017 11th Joint Meeting on foundations of software engineering*, 2017, pp. 926–931.

[17] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 364–375.

[18] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.

[19] P. Pham, V. Jain, L. Dauterman, J. Ormont, and N. Jain, "Deeptriage: Automated transfer assistance for incidents in cloud services," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3281–3289.

[20] G. Bortis and A. Van Der Hoek, "Porchlight: A tag-based approach to bug triaging," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 342–351.

[21] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *TOSEM*, vol. 20, no. 3, pp. 1–35, 2011.

[22] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu, "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems*. ACM, 2024, pp. 674–688.

[23] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv:1607.01759*, 2016.

[24] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[25] S. He, X. Zhang, P. He, Y. Xu, L. Li, Y. Kang, M. Ma, Y. Wei, Y. Dang, S. Rajmohan, and Q. Lin, "An empirical study of log analysis at microsoft," in *Proceedings of the 30th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2022, pp. 1465–1476.

[26] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[27] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[28] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin *et al.*, "Uniparser: A unified log parser for heterogeneous log data," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1893–1901.

[29] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin, D. Zhang, Z. Zhu, and D. Pei, "Robust failure diagnosis of microservice system through multimodal data," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 3851–3864, 2023.

[30] J. Cui, Z. Li, Y. Yan, B. Chen, and L. Yuan, "Chatlaw: Open-source legal large language model with integrated external knowledge bases," *arXiv:2306.16092*, 2023.

[31] Z. Li, Q. Cheng, K. Hsieh, Y. Dang, P. Huang, P. Singh, X. Yang, Q. Lin, Y. Wu, S. Levy *et al.*, "Gandalf: An intelligent, end-to-end analytics service for safe deployment in large-scale cloud infrastructure," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 389–402.

[32] H. Dai, H. Li, C.-S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using n n-gram dictionaries," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 879–892, 2020.

[33] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *2008 The Eighth International Conference on Quality Software*. IEEE, 2008, pp. 181–186.

[34] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv:1611.03213*, 2016.

[35] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[36] X. Wang, X. Zhang, L. Li, S. He, H. Zhang, Y. Liu, L. Zheng, Y. Kang, Q. Lin, Y. Dang *et al.*, "Spine: a scalable log parser with feedback guidance," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1198–1208.

[37] P. Jin, S. Zhang, M. Ma, H. Li, Y. Kang, L. Li, Y. Liu, B. Qiao, C. Zhang, P. Zhao *et al.*, "Assess and summarize: Improve outage understanding with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1657–1668.

[38] Y. He, S. Zheng, Y. Tay, J. Gupta, Y. Du, V. Aribandi, Z. Zhao, Y. Li, Z. Chen, D. Metzler *et al.*, "Hyperprompt: Prompt-based task-conditioning of transformers," in *International Conference on Machine Learning*. PMLR, 2022, pp. 8678–8690.

[39] T. Sun, Z. He, H. Qian, Y. Zhou, X.-J. Huang, and X. Qiu, "Bbtv2: towards a gradient-free future with large language models," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 3916–3930.

[40] Q. He, J. Pei, D. Kifer, P. Mitra, and L. Giles, "Context-aware citation recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 421–430.

[41] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[42] Z. Wang, Z. Liu, Y. Zhang, A. Zhong, L. Fan, L. Wu, and Q. Wen, "Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models," *arXiv:2310.16340*, 2023.

[43] D. Roy, X. Zhang, R. Bhave, C. Bansal, P. Las-Casas, R. Fonseca, and S. Rajmohan, "Exploring llm-based agents for root cause analysis," *arXiv:2403.04123*, 2024.

[44] Z. Yu, M. Ma, C. Zhang, S. Qin, Y. Kang, C. Bansal, S. Rajmohan, Y. Dang, C. Pei, D. Pei, Q. Lin, and D. Zhang, "Monitorassistant: Simplifying cloud service monitoring via large language models," in *Companion Proceedings of the 32nd International Conference on the Foundations of Software Engineering*. ACM, 2024, pp. 38–49.

[45] Y. Jiang, C. Zhang, S. He, Z. Yang, M. Ma, S. Qin, Y. Kang, Y. Dang, S. Rajmohan, Q. Lin, and D. Zhang, "Xpert: Empowering incident management with query recommendations via large language models," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. ACM, 2024, pp. 92:1–92:13.