Collaborative Knowledge Distillation and Reinforcement Learning for Automated Ticket Triage in Large-Scale Production Systems

RUOWEI FU, Nankai University, China
YANG ZHANG, ByteDance Inc., China
SHENGLIN ZHANG, Nankai University, China
XIN WU, ByteDance Inc., China
WENWEI GU, Nankai University, China
FENG WANG, ByteDance Inc., China
ZEYU CHE, Nankai University, China
XIAOZHOU LIU, ByteDance Inc., China
YONGQIAN SUN, Nankai University, China
YU ZHANG, ByteDance Inc., China

In large-scale enterprise environments, growing system complexity makes failures inevitable, threatening business continuity and customer satisfaction. To maintain system stability, efficient ticket triage is crucial for timely incident resolution. However, it remains a knowledge-intensive task requiring substantial domain expertise and detailed analysis, revealing the limitations of traditional rule-based and text-based methods. Recent advances in large language models (LLMs) offer new possibilities for automating triage through their remarkable reasoning and language capabilities. Yet, in industrial settings, LLMs often struggle to leverage domain-knowledge essential for accurate triage. We present *CoTriage*, a practical and scalable end-to-end automated ticket triage system, designed and deployed for real-world industrial scenarios. *CoTriage* leverages novel collaboration between large and small models: a small set of labeled tickets is used to distill high-quality LLM reasoning into a lightweight model, which is further optimized through a self-reinforcement mechanism. The refined triager then acts as a reward model to fine-tune a ticket summarizer using reinforcement learning. Comprehensive experiments conducted in the production environment of ByteDance, a leading global online video service provider, demonstrate the effectiveness of CoTriage, reducing the average triage time to 15.0 seconds and significantly enhancing operational efficiency and accelerating incident resolution in practice.

CCS Concepts: • **Software and its engineering → Software maintenance tools**.

Additional Key Words and Phrases: Ticket Triage, Large-small Model Collaboration, Reinforcement Learning

Authors' Contact Information: Ruowei Fu, furuowei@mail.nankai.edu.cn, Nankai University, Tianjin, China; Yang Zhang, ByteDance Inc., Beijing, China, zhangyang.329@bytedance.com; Shenglin Zhang, Nankai University, Tianjin, China, zhangsl@nankai.edu.cn; Xin Wu, ByteDance Inc., Shanghai, China, wuxin.29@bytedance.com; Wenwei Gu, Nankai University, Tianjin, China, wwgu@nankai.edu.cn; Feng Wang, ByteDance Inc., Hangzhou, China, wangfeng.ai@bytedance.com; Zeyu Che, Nankai University, Tianjin, China, chezeyu@mail.nankai.edu.cn; Xiaozhou Liu, ByteDance Inc., Beijing, China, wangding.01@bytedance.com; Yongqian Sun, Nankai University, Tianjin, China, sunyongqian@nankai.edu.cn; Yu Zhang, ByteDance Inc., Beijing, China, felix.zhang@bytedance.com.

Manuscript submitted to ACM

ACM Reference Format:
Ruowei Fu, Yang Zhang, Shenglin Zhang, Xin Wu, Wenwei Gu, Feng Wang, Zeyu Che, Xiaozhou Liu, Yongqian Sun, and Yu Zhang. 2018. Collaborative Knowledge Distillation and Reinforcement Learning for Automated Ticket Triage in Large-Scale Production Systems. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 27 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

As enterprise systems continue to scale and evolve in complexity, their IT infrastructures have become increasingly intricate, incorporating thousands of microservice components, cross-regional data centers, cloud-native architectures, and heterogeneous technology stacks. This high degree of distribution and heterogeneity introduces intricate dependencies and operational uncertainties, making the occurrence of incidents unavoidable [1, 7, 35, 52]. Typical examples include kernel failures, hardware performance degradation, and network packet loss, all of which can severely disrupt the operation of mission-critical business services. For example, a one-hour service disruption on Amazon.com can result in direct revenue losses exceeding $100 million [6]. Consequently, when such incidents arise, rapid response and effective troubleshooting by responsible teams are essential to prevent escalation and mitigate immediate financial and operational impacts.

To ensure timely responses to incidents, enterprises widely implement ticket management systems aimed at minimizing service disruptions and mitigating operational risks [9], as shown in Figure 1. Traditional ticket triage processes typically rely on a combination of manual operations and predefined rules. In this workflow, triage engineers are required to carefully review user-submitted tickets and accurately assign the ticket to the appropriate responsible team. However, there is a significant gap between the limited processing capacity of manual triage and the overwhelming volume of tickets. On the one hand, this process is time-consuming and labor-intensive, as engineers must conduct a comprehensive analysis of the ticket before assignment. In ByteDance, an triage engineer requires approximately 200 seconds on average to process a ticket. Furthermore, the efficiency and accuracy of manual triage are highly contingent upon the engineer's professional expertise, which typically requires months or even years to cultivate. Once a ticket is misassigned, the incident cannot be resolved promptly and must be reassigned based on feedback until the correct responsible team is identified. Such inefficiencies significantly prolong the incident resolution cycle, increase the risk of issue propagation, and can even escalate to critical incidents, resulting in considerable operational and financial losses [6]. Consequently, to improve operational efficiency and alleviate manual workload, the development of a rapid and accurate automated ticket triage system has become an urgent necessity for enterprises.

In recent years, numerous automated ticket triage methods have been proposed [7, 35, 44]. However, effective ticket triage requires substantial domain knowledge and complex logical reasoning. Existing approaches, which primarily rely on ticket textual information, often lack mechanisms for modeling and integrating domain-specific knowledge, thereby imposing significant limitations on their text comprehension capabilities. For example, such methods may fail to recognize critical domain-specific terminology or implicit contextual cues essential for accurate triage, ultimately resulting in degraded overall performance. Thus, their adaptability to large-scale production environments remains limited.

Recent advancements in large language models (LLMs) have demonstrated remarkable capabilities in language understanding and reasoning across various natural language processing tasks [20, 34, 55], thereby providing a robust foundation for automated ticket triage. Building upon these insights, we introduce an LLM-driven ticket triage

framework designed to address the inherent limitations of traditional approaches in effectively leveraging domain-specific knowledge and executing complex reasoning. However, it still faces two domain-specific challenges (detailed in Section 3).

Challenge 1: Class Imbalance. In real-world production environments, ticket teams are highly imbalanced, with common classes having abundant samples while rare ones remain severely underrepresented. Furthermore, evolving business requirements continuously introduce new teams with a few samples, worsening this imbalance. Such disparities hinder LLM models from learning distinctive features of minority classes, ultimately degrading overall triage performance.

Challenge 2: Noisy Ticket Content. In ticket triage, textual information (including titles and discussions) plays a critical role in assigning tickets to the appropriate teams. However, the quality of these discussions is inconsistent, often containing ambiguous phrases, images, or hyperlinks, which pose significant challenges for LLM models to accurately comprehend and utilize the content.

To address these challenges, we present ***CoTriage***, an automated ticket triage framework that leverages multi-LLM-agents to achieve efficient and accurate triage. For Challenge 1, we propose a collaborative ticket triage approach that integrates knowledge distillation to enable effective learning from limited labeled tickets, together with a self-reinforcement mechanism that mitigates class imbalance by generating and leveraging synthetic samples of rare teams, thereby enabling continuous performance improvement. For Challenge 2, we design a joint learning strategy for ticket summarization optimization, which accurately extracts critical ticket information and thereby improves the model's capability to comprehend noisy and unstructured content.

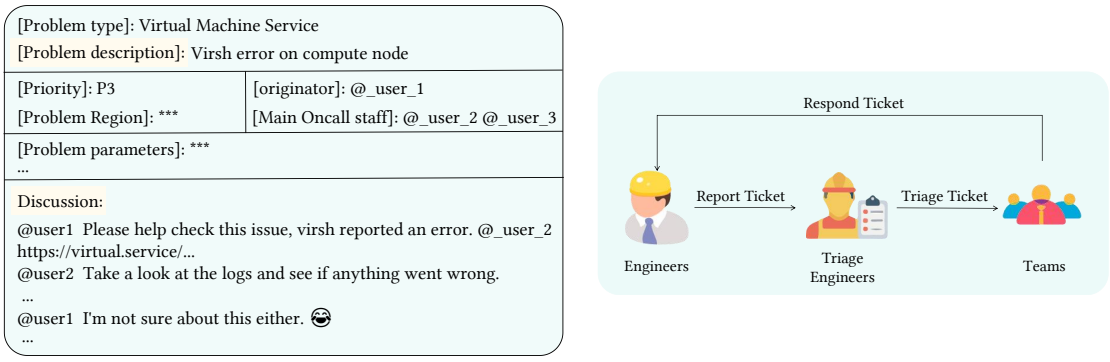
The main contributions of this work are as follows:

- We propose *CoTriage*, an end-to-end automated ticket triage framework that achieves accurate and efficient triage through large-small model collaboration.
- To achieve accurate ticket triage, we: (1) distill the reasoning capabilities of LLMs into small language models (SLMs) and continuously refine them using a self-reinforcement mechanism (Section 4.2), and (2) introduce a collaboratively optimized ticket summarization strategy to avoid the suboptimality of manual summaries (Section 4.3).
- To demonstrate the practical effectiveness of *CoTriage*, we conduct a comprehensive evaluation using a dataset collected from a real-world system technology and engineering (STE) scenario of a leading global online video service provider, ByteDance. The results demonstrate *CoTriage*'s effectiveness. Furthermore, it substantially reduces the average ticket triage time to 15 seconds, achieving a 13-fold improvement in efficiency over traditional manual workflows and demonstrating significant operational gains.

2 BACKGROUND

2.1 Incident Tickets

In real-world production environment of large-scale enterprise, system architectures are highly complex, comprising numerous interconnected service components and dependencies. During system operation and maintenance, engineers frequently encounter a wide range of incidents. These incidents can span systemic failures—such as service unavailability, abnormal requests, and severe performance degradation—to more common operational queries, including requests for device information associated with a particular IP address, clarification of technical concepts, or instructions for executing particular commands.



(a) Example of incident ticket

(b) Human triage workflow

Fig. 1. Illustration of the incident ticket and the overview of the human on-call workflow.

When issues arise, engineers typically submit tickets to the ticket management system. As illustrated in Figure 1a, each ticket serves as a detailed record that captures all the information necessary for triage, diagnosis, and resolution. Standard ticket fields generally include the problem type, problem description (a brief description of the encountered issue), priority (often categorized as low, medium, high, or critical, based on potential customer impact), originator details, associated problem parameters, and a discussion section that logs prior troubleshooting dialogues among engineers. These elements collectively provide essential context for ensuring timely and effective incident management.

2.2 Ticket Triage

To maintain system stability, enterprises widely deploy robust ticket management systems to efficiently handle tickets reported by engineers in production environments. Figure 1b illustrates a typical manual ticket triage workflow: when an incident occurs, engineers create and submit a ticket detailing the issue. A triage engineer then carefully reviews the detailed information in the tickets and classifies it based on the nature of the issue, thereby enabling its assignment to the appropriate owner or specialized team for resolution [6, 7, 35]. Each assignment typically incorporates contextual information, attempted troubleshooting steps, and relevant discussions to facilitate rapid understanding and resolution by the responsible teams. In practice, many organizations employ automated escalation mechanisms based on incident severity to ensure timely responses [32], particularly for high-impact or critical issues. However, due to the inherent complexity and interdependencies of enterprise IT systems, tickets are often misassigned to inappropriate teams. These misassignments substantially extend diagnosis and mitigation times, resulting in prolonged service downtime and elevated operational risks.

2.3 LLM-based Agents

In recent years, LLMs have achieved remarkable progress in natural language processing, demonstrating strong capabilities in language understanding and reasoning. These advancements highlight the transformative potential of Agentic AI in automating various software engineering tasks. Traditional software engineering workflows often involve lengthy and repetitive processes, such as requirements analysis, code review, and debugging, all of which demand substantial time and effort. In contrast, LLM-based agents extend the capabilities of language models to plan, reason, and interact with external tools and resources, effectively emulating diverse roles within a software development team

(e.g., requirement analyst, architect, developer, and tester). By leveraging such agents, software engineering tasks can be increasingly automated across multiple stages, reducing manual workload and improving overall work efficiency.

In the domain of software development [17, 36], MetaGPT [17] integrates Standardized Operating Procedures (SOPs) and assigns specialized roles to various LLM-based agents, enabling effective multi-agent collaboration across the software development lifecycle and facilitating the generation of high-quality solutions. In the field of code generation [3, 26], CodePlan [3] innovatively integrates an incremental dependency analysis, a change may-impact analysis and an adaptive planning algorithm with LLMs to synthesize multi-step editing chains, thereby achieving efficient, repository-level code generation. In software testing [40, 54], TestPilot [40] constructs detailed prompts that include the function signature, implementation, documentation comment, and usage examples to generate tests. When a generated test fails, the failed test along with the corresponding error messages are fed back into the LLM to iteratively refine the prompts and produce corrected tests.

Collectively, these advancements demonstrate the potential of LLM-based agents as autonomous problem-solvers, providing valuable insights and methodological inspiration for the design of automated ticket triage system.

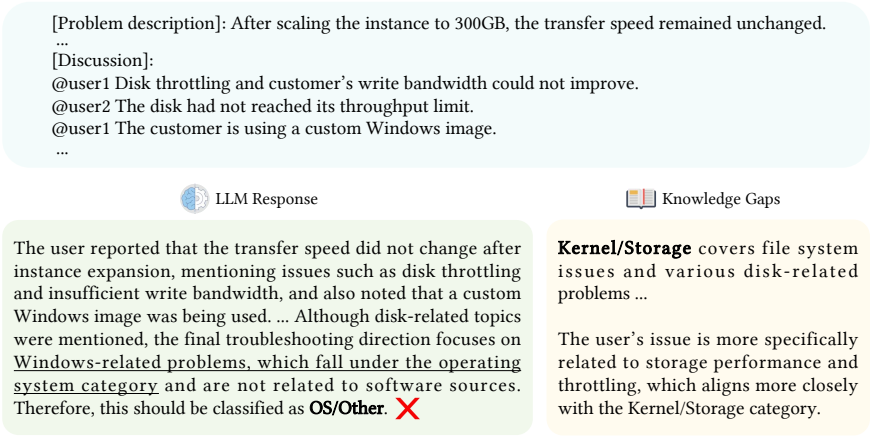


Fig. 2. Limitations of naive LLMs in ticket triage.

3 MOTIVATION

The STE team of ByteDance is responsible for core system technologies, including operating system kernels, virtualization, foundational system software, and the overall stability of large-scale data centers. Their ticket management system handles a substantial number of system failure-related tickets each day, which must be assigned to highly specialized infrastructure teams. These teams are defined by fine-grained domain knowledge rather than explicit textual patterns, making accurate triage a knowledge-intensive task.

In recent years, LLMs have demonstrated strong performance across various tasks without task-specific fine-tuning. However, ticket data often embeds rich domain knowledge, which may not be fully captured by naive LLMs, leading to incomplete understanding and misassignments. As illustrated in Figure 2, the LLM relied primarily on surface-level keywords (e.g., “Windows") rather than core indicators (e.g., “disk throttling", “bandwidth") and incorrectly

conflated troubleshooting actions with the actual root cause. Moreover, its lack of domain knowledge related to storage performance led to the ticket being misclassified as OS/Other instead of the correct Kernel/Storage team.

Therefore, to alleviate the workload of triage engineers and improve triage efficiency, we propose an end-to-end automated ticket triage framework that leverages collaborative optimization between large and small models to achieve accurate and efficient triage. However, it confronts the following two challenges: class imbalance and noisy ticket content.

3.1 Challenge 1: Class Imbalance

In real-world production environments, tickets often exhibits severe class imbalance: common fault types have abundant samples, whereas long-tail teams are sparse. This imbalance constrains the generalization capability of supervised models, resulting in overfitting to frequent classes and poor recognition of rare ones. Moreover, as business processes evolve, new teams continually emerge with only a few samples, further exacerbating the imbalance problem. Consequently, existing automated triage methods rely heavily on large annotated datasets, which is impractical under conditions of data imbalance.

To address these challenges, we propose a ticket triage framework that integrates knowledge distillation with a self-reinforcement mechanism to improve generalization under class imbalance. Specifically, we first distill the reasoning capabilities of LLM into a lightweight student model using a limited set of labeled tickets. Subsequently, through a self-reinforcement strategy, the student model iteratively improves without additional labeled tickets by leveraging its prediction confidence and consistency as guidance during learning. Crucially, during this process, the student model actively generates inputs corresponding to underrepresented classes, effectively augmenting rare-class samples. This approach reduces dependence on extensive annotations while improving the model's generalization and adaptability in imbalanced, resource-limited settings, enabling more efficient and intelligent ticket triage.

3.2 Challenge 2: Noisy Ticket Content

In ticket triage, textual information (e.g., titles and discussion content) plays a critical role in assigning tickets to the appropriate team. However, the quality of discussion content in real-world tickets is often highly variable, often containing noise such as disorganized expressions, ambiguous phrases, images or hyperlinks. Directly feeding raw tickets into LLMs can lead to hallucinations caused by noisy information and a tendency to over-rely on the beginning and end of the text while overlooking critical content in the middle, ultimately degrading triage performance. Consequently, a fundamental challenge for accurate triage is to identify and extract critical information from noisy ticket content.

Existing approaches [8] have attempted to leverage general-purpose language models without fine-tuning to generate summaries. However, tickets often contain rich domain knowledge, and LLMs trained on generic corpora lack the necessary domain awareness [4, 21, 37], resulting in summaries with significant randomness and irrelevant terms. In addition, constructing large-scale, high-quality summaries for training is highly labor-intensive, and model performance is extremely sensitive to the structure of these descriptions, while manually crafted templates are often suboptimal.

To address these challenges, we propose a collaborative optimization-based ticket summarization method that enhances domain awareness and reduces reliance on costly manual annotations by employing an end-to-end training strategy to jointly optimize accurate summary generation and effective ticket triage. Specifically, the triage model serves as a reward function to construct preference pairs, and the summary generation model is optimized via Direct Preference Optimization (DPO) [38], thereby effectively aligning summary quality with triage performance.

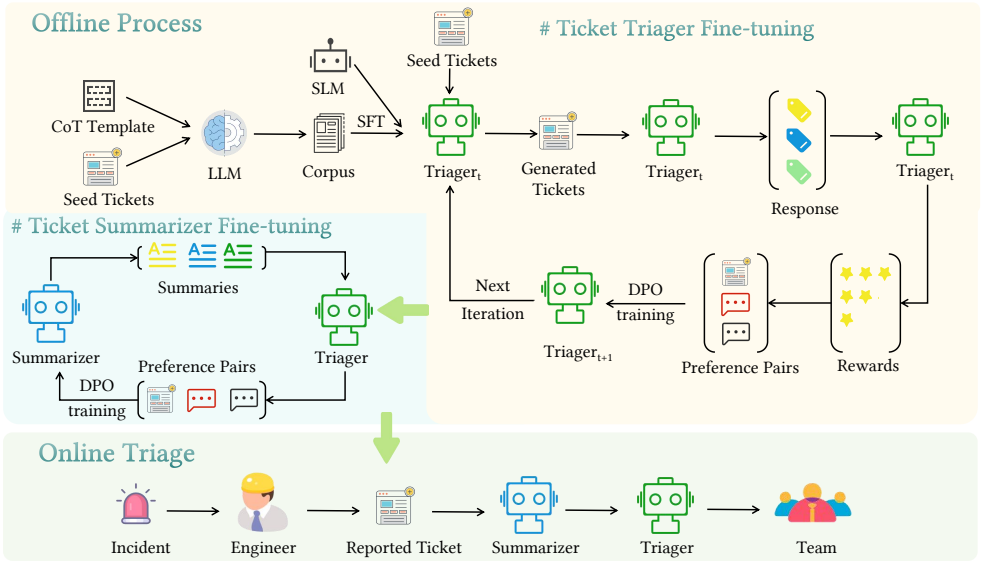


Fig. 3. The framework of *CoTriage*.

4 METHODOLOGY

4.1 Overview

As illustrated in Figure 3, the offline stage of *CoTriage* is designed to transform naive SLMs into high-quality ticket triage experts through two core modules. *Module 1, Ticket Triager Fine-tuning*, leverages a small amount of labeled ticket data to perform knowledge distillation, transferring the reasoning capability of a large, powerful teacher model to a lightweight student model. Subsequently, the preliminary fine-tuned Triager constructs preference pairs using a self-reinforcement mechanism, and iteratively improves its triage performance via DPO tuning. *Module 2, Ticket Summarizer Fine-tuning*, the fine-tuned Triager serves as a reward model to synthesize preference pairs for ticket summaries. The Summarizer is subsequently tuned through DPO to generate accurate and concise ticket summaries, which in turn facilitate more accurate triage.

For online ticket triage, when processing incident tickets, *CoTriage* first employs the Summarizer to extract key discriminative information essential for accurate triage. Subsequently, the Triager assigns tickets to the appropriate teams based on the summarized content.

4.2 Ticket Triager Fine-tuning

CoTriage employs a collaborative approach that combines large and small models, where a domain-specific SLM, Triager, is trained to compensate for the naive LLMs' limitations in domain knowledge.

4.2.1 Knowledge distillation from large models. LLMs endowed with strong capabilities in natural language understanding and reasoning, can not only predict labels but also generate rationales that explain and justify their predictions [24, 45]. Compared with simple labels, high-quality Chains of Thought (CoT) provide richer and more detailed reasoning information, establishing explicit mappings from problems to labels and enabling smaller models to better comprehend complex tasks. Prior studies have demonstrated that such CoTs can substantially enhance the reasoning ability and

```
## Instruction ##
You are a ticket triage expert with extensive knowledge in system and technical domain categorization.
The category of the text is already known. Please thoroughly understand each category based on the fo-
llowing [Category Descriptions], analyze the [User Issue Description] step by step, and provide a reaso-
ning process within 200 words.
...
## Category Descriptions ##
[Relevant category description here]
## User Issue Description ##
[Ticket content here]
## Category ##
[Ticket category ]
## Reasoning ##
[Step-by-step explanation ]
```

Fig. 4. Illustration of CoT prompt template.

learning efficiency of smaller models [16, 18, 30]. Building upon this insight, we employ LLMs as teacher models to effectively leverage their reasoning capabilities for efficient fine-tuning of smaller models.

Specifically, given an annotated ticket dataset $\mathcal{T}$, where each ticket is labeled with its corresponding team, we design a CoT prompt template p, as illustrated in Figure 4, to guide the LLM in generating detailed triage rationales. Each prompt explicitly specifies the problem-solving logic and can be represented as a quadruple (t_i, k_i, y_i, r_i), where t_i denotes the ticket content, k_i represents the relevant domain knowledge, describing the typical issues addressed by each team. These team descriptions were manually authored by engineers to concisely summarize possible scenarios for each team (e.g., the Compile/Java is responsible for handling JVM coredumps and Java exceptions), which involves a modest manual effort. Here, y_i is the ground-truth team , and r_i is the rationale explaining why t_i should be classified as y_i. For each ticket $t_i \in \mathcal{T}$, the corresponding triplet (t_i, k_i, y_i) is integrated into the prompt p and then submitted to the LLM to generate the triage rationale r_i.

After obtaining these reasoning-augmented annotations, we construct the distilled dataset $\mathcal{D} = \{(t_i, k_i, r_i, y_i)\}_{i=1}^{N}$. This enriched dataset is subsequently used to perform supervised fine-tuning on the smaller triage model, Triager. By incorporating rationales during tuning, the model is guided not only to predict the correct team but also to capture intermediate reasoning processes, fostering a deeper understanding of the underlying decision logic. This design facilitates the generation of predictions that are both more accurate and interpretable.

4.2.2 Iterative self-reinforcement. Due to the inherent imbalance in responsible team, the Triager obtained through initial supervised fine-tuning often exhibits limited generalization capability: it tends to overfit frequent teams while underperforming on rare ones. Inspired by [13, 27, 48, 53], the AI feedback demonstrates outstanding performance, enabling LLMs to continuously optimize and improve themselves using unlabeled data through AI feedback. *CoTriage* adopts a iterative self-reinforcement strategy to further enhance the Triager's triage performance, where the Triager acts both as a triage model to generate predictions and as a reward model to evaluate its own outputs. Through preference-based optimization, the Triager iteratively refines its triage performance.

The entire process is carried out in an iterative manner, with each iteration consisting of the following four main steps.

Step1: Augmented Ticket Generation. First, a subset of tickets are sampled as seed instances, and the Triager's inherent language generation capability is leveraged to synthesize new ticket examples based on these seeds. This step primarily addresses the issue of class imbalance. During generation, responsible team consistency is strictly maintained, while semantic diversity is deliberately introduced to ensure that the augmented data remains coherent and enhances the Triager's robustness to linguistic variations.

Step2: Triage Prediction Generation. The tickets generated in the previous step are then fed into the Triager, which functions as a standard triage model to generate corresponding predictions. Alongside the predicted responsible teams, the Triager produces intermediate reasoning steps, which provide interpretability and serve as critical input for subsequent reward scoring process. These rationales capture the model's internal logic and help ensure that the predictions are not only correct but also justifiable.

Step3: Self-Evaluation and Reward Scoring. The generated predictions are returned to the Triager along with an evaluation prompt that guides it to assess the outputs across multiple dimensions, including accuracy, consistency, and completeness. Specifically, the evaluation begins by verifying the correctness of the predicted team, followed by an examination of the reasoning component to ensure that it is logically coherent, sufficiently justified, and comprehensive. Based on these criteria, the Triager assigns a scalar reward score $R \in [0, 5]$, which reflects the overall quality of the prediction.

Step4: Preference Pair Construction and DPO Tuning. For each ticket sample, preference pairs $\mathcal{P}$ are constructed based on the evaluation scores, where the higher-scored prediction is designated as the "preferred option" and the lower-scored prediction as the "less preferred option". Using these pairs, we form a preference dataset $D = \{(X, Y_{\text{acc}}, Y_{\text{rej}}) \mid (Y_{\text{acc}}, Y_{\text{rej}}) \in \mathcal{P}\}$, and apply DPO to fine-tune Triager, guiding it to favor outputs with higher reward scores in future predictions.

Through this iterative refinement process, Triager incrementally mitigates performance degradation caused by class imbalance and substantially enhancing both triage accuracy and robustness. Importantly, this approach enables effective self-improvement even in scenarios where large-scale annotated data is scarce or unavailable, making it highly practical for real-world ticket triage applications.

4.3 Ticket Summarizer Fine-tuning

Incident tickets typically consist of a title accompanied by a series of discussions. However, these discussions are often highly unstructured and exhibit substantial variability in quality. In practice, they frequently contain redundant descriptions or even irrelevant information, that can hinder the decision-making capabilities of Triager and ultimately degrade triage accuracy.

To address this issue, *CoTriage* introduces a specialized ticket summarization module, referred to as Summarizer, built upon a SLM. Rather than merely producing a condensed version of the ticket, the Summarizer aims to strategically extract and emphasize information that is most salient to the final responsible team.By filtering out noise and preserving discriminative signals, the Summarizer provides the downstream Triager with cleaner and more structured textual input, thereby enhancing overall triage performance

Nevertheless, directly training such a summarization model using human annotations presents practical challenges. Generating high-quality summaries requires expert domain knowledge, is both costly and labor-intensive. Moreover, suboptimal or poorly structured summaries may misguide the Triager, potentially exacerbating performance degradation. To overcome these limitations, we propose a collaborative ticket summarization fine-tuning approach that jointly optimizes summary generation and triage effectiveness. This strategy encourages the Summarizer to produce outputs

that are not only fluent and concise but also maximally informative and utility-aligned with downstream triage objectives.

Specifically, this approach consists of three key steps: (1) generating diverse candidate summaries, (2) assigning rewards based on triage performance, and (3) aligning summary generation with reward-driven preferences through DPO fine-tuning.

4.3.1 Ticket Summarization. For each input ticket $t_i \in \mathcal{T}$, the Summarizer generates a set of candidate summaries $\{s_1^i, s_2^i, \dots\}$ based on the ticket title and discussion content. To enhance the diversity of candidate summaries, we adopt temperature-controlled decoding. Increasing the decoding temperature smooths the token probability distribution, reducing the dominance of high-probability tokens and encouraging lexical and structural variation in the generated summaries. Such a diverse pool of candidate summaries provides a rich foundation for subsequent preference-based optimization, making it possible to identify and reward those summaries that are most beneficial for accurate and robust triage.

4.3.2 Reward Assignment Based on Triage Performance. To effectively guide the Summarizer toward generating summaries that enhance triage accuracy, we leverage the previously fine-tuned Triager as a reward model. Specifically, for each candidate summary s_n^i, we concatenate it with the corresponding domain-specific knowledge k_i and the predefined set of team labels. This combined input is then passed to the Triager, which outputs a confidence distribution over all possible teams. The confidence score corresponding to the ground-truth label y_i is extracted and used as the reward signal:

$$R(k_i, s_n^i) = \text{confidence}(y_i \mid t_i, s_n^i, k_i). \tag{1}$$

This reward formulation provides task-aligned and model-driven feedback without relying on human annotations. Summaries that enable the Triager to produce more confident and accurate predictions are assigned higher rewards, whereas those that introduce ambiguity or omit critical information are penalized. Consequently, the Summarizer is implicitly optimized to capture salient and discriminative information that directly enhances triage accuracy.

4.3.3 Summarizer Optimization via DPO Tuning. With reward signals available for each candidate summary, we further construct pairwise preference relations within the candidate set corresponding to the same ticket. Specifically, for each ticket, candidate summaries with higher reward scores are considered preferable to those with lower scores, forming a set of preference pairs. We further employ the DPO framework to fine-tune the Summarizer. DPO explicitly models the preference distribution between candidate summaries, thereby effectively aligning the output distribution of the Summarizer with reward-driven preferences. This approach enables the consistent optimization of summary generation and triage performance, without requiring explicit human annotations.

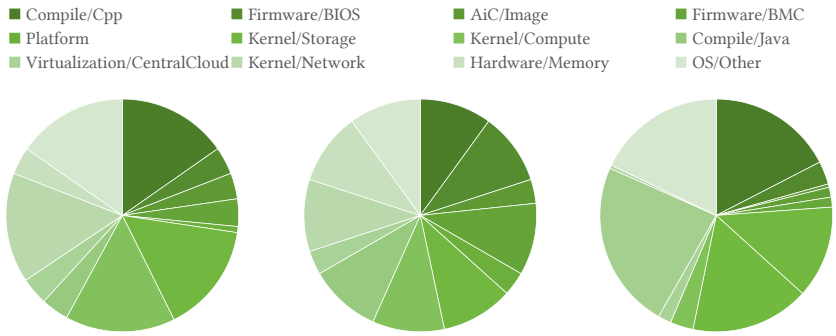
5 EVALUATION

In this section, we conduct a comprehensive evaluation of *CoTriage* to answer the subsequent research questions (RQs):

- **RQ1:** How does *CoTriage* perform in ticket triage?
- **RQ2:** How do model scale and reasoning capability affect *CoTriage*'s performance?
- **RQ3:** Does each component of *CoTriage* contribute significantly to *CoTriage*'s performance?
- **RQ4:** How efficient is *CoTriage* in production environment?
- **RQ5:** How does the decoding temperature affect summarization diversity and downstream triage performance?
- **RQ6:** How robust is *CoTriage* under data drift?

- **RQ7:** How the quality of generated summaries?
- **RQ8:** How does the general capability of *CoTriage* perform?
- **RQ9:** How effective is ticket synthesis?

5.1 Experiment Setup



(a) Training Dataset (b) Validation Dataset (b) Test Dataset

Fig. 5. Team distribution across training, validation, and test sets

5.1.1 Dataset. To evaluate the accuracy of ticket triage, we collect a total of 1,678 tickets from the production environment of ByteDance from Q1 2023 to Q3 2025, involving 12 distinct teams. To better reflect real-world deployment scenarios, we partition the dataset chronologically. Specifically, the earliest 1,308 tickets are used for training, followed by 150 tickets form a validation set for hyperparameter tuning (e.g., temperature selection). The 220 most recent tickets from the latest months are reserved as the test set to evaluate triage performance. In addition, we collect 30 tickets from two newly introduced teams to assess the robustness of *CoTriage* under data drift and previously unseen teams.

As illustrated in Figure 5, the dataset is constructed to reflect the real-world distribution of ticket teams observed in the production environment. Specifically, both the training and test sets follow a naturally imbalanced, long-tailed distribution derived from ticket records collected over recent months. Teams such as Kernel/Network and OS/Other occur more frequently, whereas others (e.g., Platform) are comparatively rare. Such a configuration enables a more rigorous evaluation of whether *CoTriage* can effectively mitigate the adverse effects of class imbalance during the training process.

5.1.2 Implementation Details. We conduct all the experiments with two NVIDIA A6000 GPU. Considering the trade-off between resource overhead and Chinese language processing capability, we experiment on three SLMs: DeepSeek-R1-Distill-Qwen-7B [10], Llama-3.1-8B-Instruct [12], Qwen3-8B and Qwen3-4B [49]. Additionally, GPT-3.5-Turbo-16k was employed as the teacher model for the knowledge distillation. Within our framework, the Triager was fine-tuned through three rounds of self-reinforcement.

5.1.3 Evaluation Metrics. Following prior work [7, 44], we adopt the widely used accuracy metric to evaluate the performance of ticket triage. Under this evaluation scheme, a prediction is considered correct if the actual responsible

team is ranked among the top N teams by the classifier based on their predicted probabilities. Specifically, we report two metrics, ACC@1 and ACC@3, which are calculated as follows.

$$\text{ACC@N} = \frac{\text{Sum(Correct Team in Top N Teams)}}{\text{Test Size}} \tag{2}$$

To further assess the performance of different methods under class-imbalanced conditions, we additionally employ Macro-Precision, Macro-Recall and Macro-F1 score, which assign equal importance to each team regardless of its frequency. Furthermore, we report Precision (P), Recall (R), and F1-score (F1) to specifically evaluate the performance on head and tail classes.

5.1.4 Baseline Approaches. To evaluate ticket triage performance of *CoTriage*, we adopt the following methods.

- **DeepCT [7]:** DeepCT introduces a gated recurrent unit (GRU) model with attention-based mask strategy and a revised loss function, enabling it to incrementally learn from discussions and dynamically update triage results.
- **DeepTriage [35]:** DeepTriage ensembles several submodels, including a simple multiple additive regression tree (MART) model, a light gradient boosting machine (LGBM) model, an inverted index model, a locality-sensitive hashing model (SI), and a deep neural network (DNN) model.
- **COMET [44]:** COMET filters out irrelevant logs, leverages a domain-informed LLM for keyword extraction, and performs similarity-based triage using embeddings from a fine-tuned FastText model.
- **TickIt [31]:** TickIt adopts a Chain-of-Thought (CoT) reasoning paradigm integrated with retrieval-augmented In-Context Learning (ICL), leveraging LLM to enable accurate multi-class ticket escalation.
- **Triangle [51]:** Triangle is a multi-agent-based incident triage framework that integrates semantic distillation with multi-role agent negotiation.
- **SetFit [42]:** SetFit is a prompt-free and efficient few-shot learning method based on Sentence Transformer.
- **w/CoT:** LLM with the CoT prompting [46], encouraging the model to generate intermediate reasoning steps before producing the final triage decision.
- **w/Few-Shot:** LLM with the few-shot prompting [5], where a small set of labeled ticket examples is provided in the prompt to guide prediction.
- **w/RAG:** LLM with Retrieval-Augmented Generation (RAG) framework, where historically labeled tickets serve as the knowledge base for retrieval during inference.

5.2 RQ1: The Overall Performance

The comparative results presented in Table 1 demonstrate the promising performance of *CoTriage*, which achieves an ACC@1 of 0.618 and a Macro-F1 score of 0.506. Compared to non-LLM baselines such as DeepTriage, DeepCT, and SetFit, *CoTriage* demonstrates a substantial performance improvement. Specifically, it achieves Macro-F1 gains ranging from 28.2% to 40.4%, highlighting the advantage of LLMs in understanding complex, unstructured ticket data. Ticket triage remains an inherently knowledge-intensive task, requiring models not only to grasp category distinctions but also to perform multi-step reasoning. When compared against various LLM-based approaches, under the same backbone (Qwen3-8B), *CoTriage* consistently achieves the best overall performance, with ACC@1 improvements ranging from 1.1% to 8.6% and Macro-F1 gains of 0.6% to 17.8%. These results indicate that the improvements do not merely stem from the backbone model itself, but from the *CoTriage* framework. Furthermore, *CoTriage* progressively approach, and for certain metrics even narrow, the gap with approaches built on more powerful proprietary LLMs such as GPT-3.5. A key advantage of *CoTriage* lies in its superior performance–efficiency trade-off. While Triangle incurs a substantial

Table 1. Performance comparison of triage methods. (Statistical significance is determined by a Wilcoxon signed-rank test on per-class F1-scores across 12 teams, † indicates significant statistical differences ($p \leq 0.05$) between a baseline and ours.)

Seed LLM	Method	ACC@1	ACC@3	Macro-Precision	Macro-Recall	Macro-F1	Avg.Tokens
	DeepTriage†	0.336	0.632	0.174	0.172	0.165	-
	DeepCT†	0.214	0.491	0.089	0.114	0.070	-
	SetFit†	0.236	0.441	0.250	0.176	0.178	-
GPT-3.5	w/CoT	0.635	0.854	0.540	0.553	0.505	2.91k
	w/Few-Shot	0.591	**0.891**	0.481	0.481	0.439	2.85k
	w/RAG	0.636	0.866	0.435	0.524	0.435	5.43k
	COMET†	0.318	0.855	0.254	0.268	0.211	2.35k
	TickIt	0.653	0.875	**0.559**	**0.564**	**0.539**	6.13k
	Triangle	**0.665**	0.872	0.501	0.493	0.449	15.62k
Qwen3-8B	w/CoT	0.594	0.845	0.413	0.453	0.396	3.70k
	w/Few-Shot†	0.568	0.836	0.477	0.480	0.447	3.80k
	w/RAG†	0.532	0.775	0.345	0.361	0.328	6.13k
	TickIt	0.607	0.826	0.515	0.532	0.500	5.11k
	CoTriage	0.618	0.823	0.515	0.560	0.506	4.24k

Table 2. Performance comparison on head and tail teams

Seed LLM	Method	Head Teams						Tail Teams					
		Kernel/Network			OS/Other			Virtualization/CentralCloud			Platform		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
	DeepTriage	0.353	0.471	0.403	0.308	0.205	0.246	0	0	0	0	0	0
	DeepCT	0.364	0.078	0.129	0.289	0.333	0.310	0.077	**0.500**	0.133	0	0	0
	SetFit	0.778	0.137	0.233	0.232	0.333	0.274	0.056	0.250	0.091	0.167	0.333	0.222
GPT-3.5	w/CoT	0.816	0.608	0.697	0.491	**0.737**	**0.589**	**0.143**	0.250	0.182	**0.667**	0.667	**0.667**
	w/Few-Shot	0.865	0.627	0.727	0.423	0.564	0.483	0.125	0.250	0.167	0.500	0.667	0.571
	w/RAG	**0.894**	0.667	0.764	0.453	0.649	0.533	0	0	0	0.286	0.667	0.400
	COMET	0.714	0.392	0.506	0.342	0.333	0.338	0.040	0.250	0.069	0.118	0.667	0.200
	TickIt	0.786	0.673	0.725	0.471	0.632	0.539	0.167	0.250	0.200	0.500	0.667	0.571
	Triangle	0.837	0.804	0.820	0.552	0.421	0.478	1.000	0.250	**0.400**	0.400	0.667	0.500
Qwen3-8B	w/CoT	0.795	0.620	0.697	0.440	0.564	0.494	0.053	0.250	0.087	0.333	0.667	0.444
	w/Few-Shot	0.780	0.627	0.696	0.382	0.538	0.447	0.045	0.250	0.077	0.500	0.667	0.571
	w/RAG	0.808	**0.745**	**0.776**	**0.519**	0.378	0.438	0.034	0.250	0.061	0.200	0.667	0.308
	TickIt	0.783	0.706	0.742	0.391	0.462	0.424	0.100	0.250	0.143	**0.667**	0.667	**0.667**
	CoTriage	0.731	**0.745**	0.738	0.462	0.462	0.462	0.053	0.250	0.087	**0.667**	0.667	**0.667**

overhead of 15.62k tokens per request due to intensive multi-agent interactions, CoTriage achieves higher F1 scores using only 4.24k tokens, reducing token consumption by approximately 72.6%. This efficiency is particularly important in real-world industrial deployment, where inference latency and token costs directly impact scalability. Overall, the results confirm that CoTriage achieves a strong balance between accuracy, reasoning capability, and deployment efficiency.

To address the extreme class imbalance in real-world production environment, the results in Table 2 demonstrate the robustness of *CoTriage*. For highly scarce tail classes, such as the Platform, *CoTriage* achieves an F1 score of 0.667.

However, performance varies across different tail classes. For instance, in Virtualization/CentralCloud, the F1 score drops to 0.087. This is primarily because incidents in this team often exhibit surface symptoms such as disk recognition anomalies, increased CPU load, or system performance fluctuations at the operating system level, while their root causes lie in virtual machine resource scheduling or abnormalities in the underlying virtualization components. This leads to semantic overlap with OS/Other team and blurred class boundaries, making fine-grained discrimination more difficult under limited training data. For head classes, such as Kernel/Network, *CoTriage* maintains competitive performance comparable to GPT-3.5-based methods, ensuring stability on common tasks. By leveraging an effective fine-tuning strategy, our approach mitigates the challenges of few-shot learning for most tail classes, enabling accurate identification of rare teams. Overall, these results highlight *CoTriage*'s capability in addressing class imbalance, while also indicating potential improvement space for extremely semantically ambiguous teams.

CoTriage significantly improves ticket triage performance by incorporating a ticket Summarizer to extract the most relevant information from ticket content and by employing a fine-tuned Triager that integrates effective reasoning with domain-specific knowledge.

5.3 RQ2: Impact of Model Scale and Reasoning Capability on *CoTriage*'s Performance

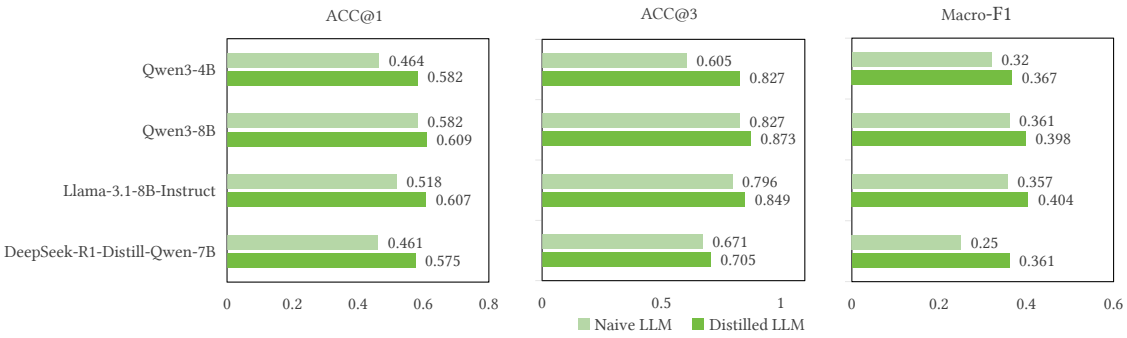


Fig. 6. Effect of distillation on different SLMs.

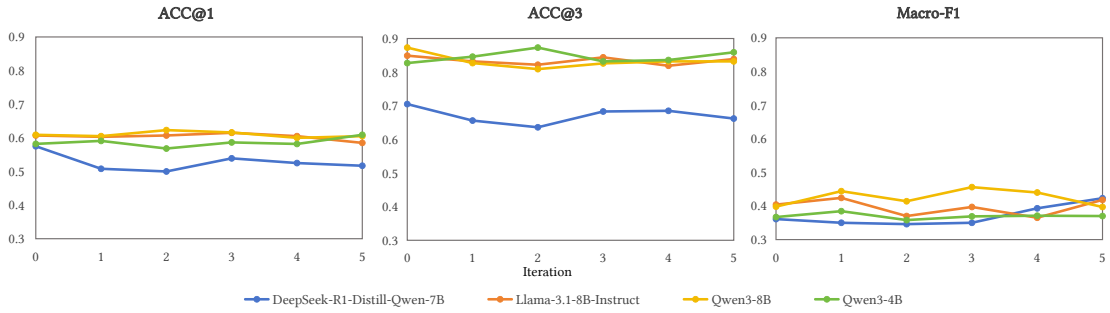


Fig. 7. The performance of *CoTriage* across different backbone SLMs under different iterations.

To assess the impact of different SLMs scale and reasoning capability on the performance of *CoTriage* , we conduct a comprehensive analysis across four representative models: DeepSeek-R1-Distill-Qwen-7B [10], Llama-3.1-8B-Instruct [12], Qwen3-8B and Qwen3-4B [49]. We evaluate each model using three metrics: ACC@1, ACC@3, Macro-F1 Score.

Effect of Distillation. As illustrated in Figure 6, the knowledge distillation mechanism consistently improves performance across all SLM backbones. Notably, we observe that the smaller backbone, Qwen3-4B, demonstrates the highest sensitivity to the refinement process, with ACC@1 increased from 0.464 to 0.582. This observation suggests that smaller or less advanced models benefit more substantially from the refinement process, indicating that *CoTriage* does not merely rely on strong SLMs but can effectively enhance weaker SLMs.

Performance Across Iterations. As illustrated in Figure 7, model performance across multiple self-refinement iterations demonstrates the model's stability and convergence. We observe that both ACC@1 and ACC@3 improve or stabilize within 2–3 iterations, suggesting that the self-refinement mechanism converges without amplifying noise. This trend suggests that early iterations progressively correct model errors, after which performance reaches a stable equilibrium.

In conclusion, Qwen3-8B demonstrates superior performance. Consequently, we adopt it as the default SLM backbone in all subsequent experiments, providing a stable and high-performance foundation for evaluating the effectiveness of *CoTriage*.

5.4 RQ3: Ablation Study

Table 3. The evaluation results of ablation study

Seed LLM	Method	ACC@1	ACC@3	Macro-Precision	Macro-Recall	Macro-F1
Qwen3-8B	*CoTriage*	0.618	0.823	0.515	0.560	0.506
	A1: w/o Triager	0.568	0.832	0.407	0.402	0.374
	A2: w/o Summarizer	0.600	0.832	0.451	0.492	0.440

To comprehensively validate the contribution of each core component in *CoTriage*, we conduct an ablation study under different conditions: A1: replace the fine-tuned Triager with a naive SLM (Qwen3-8B) to generate triage results; A2: remove the fine-tuned Summarizer and directly use the Triager for ticket triage.

5.4.1 Effectiveness of the Triager Module. The results in Table 3 demonstrate that the Triager significantly improves the triage performance. Specifically, the ACC@1 increased from 0.568 to 0.618, while Macro-F1 score achieves an improvement of 13.2%. These improvements can be mainly attributed to two factors: (i) the knowledge distillation mechanism, which integrates labeled tickets with LLM-generated reasoning information, enabling SLMs to capture not only ticket–label mappings but also the underlying logical cues of triage decisions; and (ii) the self-reinforcement strategy, which iteratively refines model behavior by reinforcing correct reasoning patterns and generating dense, model-driven signals, thereby mitigating class imbalance and data sparsity. Together, these mechanisms endow the Triager module with stronger generalization and robustness in ticket triage tasks, leading to substantial gains in triage accuracy.

5.4.2 Effectiveness of the Summarizer Module. As illustrated in Table 3, incorporating the Summarizer leads to further improvements in triage accuracy. Specifically, the ACC@1 increases by 1.8%, while Macro-F1 score achieves a gain of

6.6%. These gains can be attributed to the collaborative optimization strategy adopted by the Summarizer, in which a fine-tuned Triager serves as a reward model to guide the generation of summaries that better support accurate triage decisions. By jointly optimizing the summarization and triage objectives, the Summarizer learns to produce semantically focused summaries that are closely aligned with the requirements of downstream triage tasks. These structured summaries reduce noise and irrelevant information, providing the Triager with concise and discriminative input representations. This design is particularly effective for handling tickets with lengthy or highly unstructured descriptions, enhancing the robustness and accuracy of the overall triage process.

5.5 RQ4: Efficiency of *CoTriage*

Previously, manual ticket triage typically required over 200 seconds per ticket. Such latency primarily arose from two factors: (i) the heavy workload that often delayed timely processing, and (ii) the knowledge-intensive nature of the task, which demanded thorough contextual comprehension, and domain-specific reasoning to ensure accurate triage.

With the adoption of *CoTriage*, the average triage time per ticket is reduced to 15.0 seconds, achieving a 13 times improvement in efficiency compared to manual triage. Specifically, the Summarizer takes an average of 7.8 seconds to generate a concise and informative summary of the ticket, and the Triager spends approximately 7.2 seconds to determine the ticket responsible team. This remarkable reduction not only lowers human and time costs but also substantially enhances overall operational efficiency.

5.6 RQ5: Temperature Sensitivity in Summarization

Table 4. Comparison of temperature configurations.

Temperature	ACC@1	ACC@3	Macro-Precision	Macro-Recall	Macro-F1
0.3	0.607	0.800	0.640	0.548	0.548
0.8	0.624	0.832	0.690	0.603	0.608
1.2	0.613	0.800	0.650	0.544	0.549

To examine the impact of decoding temperature on the effectiveness of *CoTriage*, we conduct a sensitivity analysis on this hyperparameter in the summarization module. Table 4 reports the ACC@1 and Macro-F1 scores under different temperature settings. The results indicate that temperature has a impact on balancing summary diversity and semantic fidelity. When the temperature is set to a low value (e.g., $\tau = 0.3$), generation becomes nearly deterministic, leading to limited lexical variation and reduced diversity among candidate summaries. This leads to relatively low performance (ACC@1 = 0.607). As the temperature increases to $\tau = 0.8$, *CoTriage* achieves the best performance (ACC@1 = 0.624 , Macro-F1 = 0.608), suggesting that moderate stochasticity effectively enhances summary diversity while preserving key diagnostic signals. However, further increasing the temperature to $\tau = 1.2$ degrades performance (ACC@1 = 0.613), as excessive randomness introduces noise, negatively affecting the Triager's decision-making process. These findings indicate that a moderate temperature strikes the optimal balance between diversity and informativeness. Based on this observation, we adopt $\tau = 0.8$ as the default setting in summary generation.

5.7 RQ6: Robustness of *CoTriage* Under Data Drift

Table 5. Performance of *CoTriage* on novel responsible team.

Seed LLM	Method	P	R	F1
GPT-3.5	w/CoT	0.882	0.500	0.638
	w/Few-Shot	1.000	0.300	0.462
	w/RAG	0.483	0.414	0.446
	TickIt	1.000	0.467	0.636
	Triangle	1.000	0.679	0.805
Qwen3-8B	w/CoT	0.940	0.379	0.525
	w/Few-Shot	0.900	0.233	0.367
	w/RAG	1.000	0.433	0.507
	TickIt	0.944	0.333	0.451
	CoTriage	0.938	0.400	0.554

In real-world production environments, ticket teams may change due to organizational adjustments. To evaluate the robustness of *CoTriage* under data drift and unseen teams, we conducted experiments on 30 tickets from two new teams. Traditional deep learning methods such as DeepTriage and DeepCT rely on supervised fine-tuning with a fixed output space and therefore cannot predict previously unseen teams. Similarly, COMET depends on historical incident retrieval and is unable to identify novel responsible team. Consequently, we focus on LLM-based baselines for comparison. As shown in Table 5, *CoTriage* achieves an F1-score of 0.554 on the new responsible teams, demonstrating its adaptability to unseen ticket types.

5.8 RQ7: Analysis of Generated Summary Quality

Table 6. Human evaluation of generated summaries.

	Factual Consistency	Relevance	Informativeness
Qwen3-8B (Naive)	4.08	4.41	4.25
Summarizer(Qwen3-8B)	4.12	4.54	4.32

To evaluate the quality of the generated summaries, we randomly sampled 50 tickets from the test set while ensuring coverage across diverse ticket teams. The generated summaries were assessed along three dimensions: Factual Consistency, Relevance, and Informativeness. Factual Consistency measures whether the summary faithfully reflects the content of the original ticket without introducing unsupported information. Relevance evaluates the degree to which the summary focuses on fault-related information critical for downstream triage. Informativeness measures whether the summary retains sufficient key information to enable accurate triage.

The evaluation was independently conducted by two domain experts with extensive experience in incident management. Each dimension was scored on a 1-5 scale, where higher scores indicate better summary quality. To ensure the reliability of the evaluation results, we further measured inter-annotator agreement using weighted Cohen's Kappa, achieving a score of 0.588, indicating moderate agreement between the evaluators.

Table 6 reports the average quality scores for summaries generated by the naive LLM (Qwen3-8B) and our reward-optimized Summarizer. The evaluation results indicate that both methods produce summaries with high factual consistency. Furthermore, the reward-optimized Summarizer achieves higher Relevance scores, generating summaries

that are more closely aligned with the fault content. These findings suggest that reward-optimized summaries provide more relevant and informative information for downstream ticket triage, while maintaining high factual fidelity.

5.9 RQ8: The General Capability of *CoTriage*

Table 7. Bug triage performance comparison of triage methods on the MSR 2013 bug dataset.

Seed LLM	Method	ACC@1	ACC@3	Macro-Precision	Macro-Recall	Macro-F1
	DeepTriage	0.310	0.480	0.263	0.202	0.200
	DeepCT	0.190	0.345	0.010	0.050	0.016
	SetFit	0.225	0.405	0.243	0.239	0.210
	COMET	0.220	0.555	0.185	0.212	0.183
GPT-3.5	TickIt	**0.435**	0.625	0.430	**0.460**	**0.415**
	Triangle	0.402	**0.628**	**0.492**	0.423	0.378
	w/CoT	0.133	0.331	0.215	0.139	0.124
	w/Few-Shot	0.209	0.429	0.168	0.164	0.129
Qwen3-8B	w/RAG	0.250	0.440	0.295	0.242	0.208
	TickIt	0.315	0.444	0.337	0.314	0.281
	CoTriage	0.335	0.510	0.307	0.318	0.279

To evaluate the generalization capability of *CoTriage*, we conducted comprehensive experiments on the widely used MSR 2013 Bug dataset[7, 25, 51], which is based on the defect tracking records of the open-source software system Eclipse. Since the original dataset lacks team-related information required by LLM-based approaches, we randomly selected 20 developers and summarized key team functional information based on existing metadata and historical bug resolution records. This approach balanced manual effort while ensuring adequate coverage across 200 assembled bug cases. As shown in Table 7, under the same experimental settings, *CoTriage* achieved an ACC@1 of 0.335 and a Macro-F1 score of 0.279, outperforming other methods based on the same backbone model. Although all methods performed lower than conventional ticket triage, likely due to the relatively limited textual information in bug reports, *CoTriage* consistently maintained a clear advantage, providing partial evidence of its generalization capability.

5.10 RQ9: Qualitative Analysis of Synthetic Tickets

To examine whether synthetic tickets introduce meaningful diversity rather than mere surface-level paraphrasing, we present representative example in Table 8. As shown, by using original ticket titles as generative seeds, our method enriches the operational context. While strictly preserving the core semantic integrity of labels (e.g., “connection error”), the synthesized tickets organically incorporate fine-grained, discriminative features, such as CLOSE_WAIT kernel states and gRPC protocol latencies. This transition from sparse intent to rich operational scenarios shows that our synthetic tickets go beyond surface-level paraphrasing, helping to address long-tail class sparsity and providing the model with a stronger basis for accurate ticket triage.

6 Industrial Deployment

6.1 Online Deployment

CoTriage has been deployed in the ticket management system of ByteDance’s STE team. For each engineer-submitted ticket, *CoTriage* first generates a concise summary by extracting key information from the title and discussion content.

Table 8. Qualitative examples of synthetic tickets.

Type	Illustrative Examples
Seed Ticket	Some instances in the TCE service frequently encounter TCP connection errors when accessing external networks.
Synthetic Ticket 1	In the TCE cluster, some application instances frequently encounter TCP connection timeouts when accessing external APIs. The errors are concentrated during specific time periods. Restarting the instances temporarily mitigates the issue, but it reoccurs afterward. The network connection state of the instances is observed as **CLOSE_WAIT**, but the root cause remains unclear. An investigation is needed to determine whether abnormal network configurations or service dependencies are causing connections to remain open for extended periods, and whether any underlying network devices or firewall policies are imposing restrictions. Assistance is requested to analyze relevant logs and network paths to identify the root cause and resolve the issue.
Synthetic Ticket 2	On the TCE cluster, a microservice intermittently experiences connection timeouts when invoking external APIs via the **gRPC protocol**. Network monitoring indicates connection establishment failures and significant request latency fluctuations. Investigation is needed to determine whether the issue stems from network configuration or service dependency problems.

Based on this summary, it predicts the appropriate team and assigns the ticket to the appropriate team, where engineers further analyze and resolve the incident.

Training Cost. The system runs on a high-performance Linux server equipped with an Intel(R) Xeon(R) Gold 5416S CPU and two NVIDIA A6000 GPU with 48 GB VRAM. The training pipeline includes 11 hours of knowledge distillation fine-tuning, followed by 15 hours of self-reinforcement optimization to transform a naive LLM into a specialized ticket Triager, and an additional 4 hours of DPO tuning for the ticket Summarizer.

6.2 Case Study

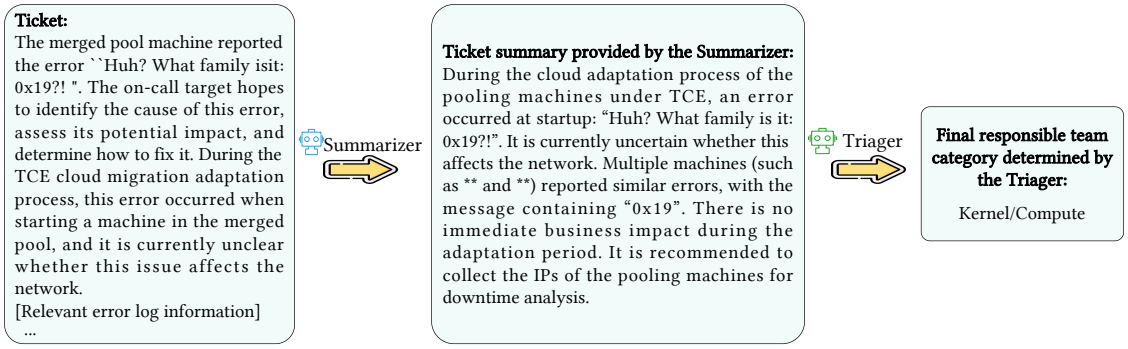


Fig. 8. The workflow of an example.

To comprehensively demonstrate the effectiveness of *CoTriage*, we present a detailed case study. To maintain strict corporate data privacy standards, all sensitive information has been excluded from the description.

As illustrated in Figure 8, we analyze a real-world ticket from a TCE cloud migration scenario, where a startup error occurred in a merged pool environment with the message: "Huh? What family is it: 0x19?!" We propose a collaborative strategy. Firstly, the Summarizer filters system noise and extracts the most discriminative information, specifically the startup error occurring during the cloud adaptation process in a merged pool environment. Based on this distilled summary, the Triager infers that the hexadecimal code "0x19" indicates potential kernel resource conflicts. By mapping these symptoms to the Kernel/Compute team, which covers soft lockups, reboot issues, and kernel panics, the system identifies the ticket as kernel-related. Ultimately, the Triager accurately assigns the ticket, demonstrating the collaborative advantage of *CoTriage* in enabling precise reasoning and reliable triage.

6.3 Generalizability and Flexibility

6.3.1 Generalizability. This study is based on data collected from the production environment of ByteDance's STE team, ensuring the representativeness and generalizability of our methodologies and findings. Although the framework was initially designed for the specific scenarios in this study, it can be effectively extended to other tasks involving complex long-text classification, such as ticket triage in customer service systems. This demonstrates *CoTriage*'s strong adaptability and potential for cross-scenario applications in diverse long-text processing contexts.

6.3.2 Flexibility. *CoTriage* adopts a modular design, endowing the framework with high flexibility and enabling fine-grained customization according to specific application requirements without compromising overall triage performance. For instance, users can adjust the sampling strategy of the summarizer or the scoring mechanism to accommodate different application contexts. This configurability is particularly critical in dynamic and diverse application scenarios, allowing the framework to adapt to various demands and challenges, thereby significantly enhancing its applicability and practical value in production environments.

6.4 Lessons Learned

6.4.1 Collaborative Optimization Enhances Summary Generation Quality. Although LLMs perform well in summarization, they often fail to capture critical information when processing real-world tickets, which are typically lengthy, noisy, and rich in domain knowledge. This limitation undermines their effectiveness in supporting downstream triage tasks. Our study shows that collaborative optimization, using the Triager as a reward model for end-to-end training, effectively guides the summarizer to focus on discriminative information closely related to triage, improving the alignment between summary quality and triage accuracy.

7 DISCUSSION

7.1 Limitations

Our study explores the potential of collaboration between large and small models in ticket triage scenarios and identifies two major limitations.

7.1.1 Self-Reinforcement Induced Bias. The self-reinforcement mechanism relies on the model's internal cognition, which may inadvertently amplify existing biases during reinforcement learning optimization if the base model has absorbed undesirable biases during training. The model may rationalize its predictions through seemingly coherent reasoning steps and reward itself for outputs that conform to learned patterns. We observed some cases where the reasoning chain appeared plausible, yet the final classification was incorrect. To mitigate this issue, we first fine-tuned

the model via knowledge distillation, equipping it with an initial high-quality triage capability. Building on this improved foundation, we then applied self-reinforcement to further enhance performance. Future work should explore hybrid strategies, such as using endogenous rewards as the primary signal while incorporating high-quality human feedback or guidance from more capable models to periodically correct model biases.

7.1.2 Limitations in Handling Image-Based or Multimodal Data. Our current approach focuses exclusively on text-based tickets and does not incorporate image-based or multimodal information, such as screenshots or visual logs, which are prevalent in real-world ticket management systems. This limitation may reduce the model's ability to fully capture the contextual cues present in complex tickets, potentially affecting triage accuracy. To address this limitation, future work should explore integrating visual representations or joint text-image modeling, possibly combining pretrained vision-language models with our current framework to enhance performance on tickets containing rich visual information.

7.2 Threats to Validity

7.2.1 Internal threats. The triage performance of *CoTriage* can be influenced by the selection of SLMs. To systematically examine this potential sensitivity, we conducted comparative evaluations across several widely used SLMs, including DeepSeek-R1-Distill-Qwen-7B, Llama-3.1-8B-Instruct, Qwen3-8B and Qwen3-4B. The experimental results demonstrate that, while slight performance variations exist among different model , *CoTriage* consistently achieves high triage accuracy across diverse configurations. These findings highlight the robustness and generalizability of our approach, indicating that its effectiveness is not overly sensitive to the particular choice of backbone models.

In addition, the human evaluation of the generated summaries may introduce potential threats to internal validity. Although the 50 evaluation samples were randomly selected to cover diverse ticket teams, they may not fully represent all real-world incident scenarios. Furthermore, the evaluation process inherently involves subjective judgment from human experts. To mitigate this threat, each summary was independently assessed by two domain experts with extensive experience in incident management. The final scores were computed by averaging the ratings across evaluators, thereby reducing potential individual bias. Moreover, inter-annotator agreement was assessed using weighted Cohen's Kappa, with a score of 0.588, indicating moderate agreement among the evaluators.

7.2.2 External threats. The threat to external validity lies in the extent to which our experimental results can be generalized. We evaluated *CoTriage* only on ticket data collected from the STE team in ByteDance's real-world production environment. Nevertheless, we believe *CoTriage*'s method is highly transferable to triage scenarios in other organizations, owing to its modular design and flexible fine-tuning strategy for SLMs, which together offer strong adaptability. With a limited amount of labeled data from a target domain, *CoTriage* can be efficiently transferred and customized, enabling rapid deployment and improved performance in new application scenarios. Future work will focus on further validating and optimizing *CoTriage*'s generalizability and robustness across industries, languages, and diverse operation and maintenance environments, promoting its application in broader real-world production environments.

7.3 Broader Potentials of Agentic AI in AIOps

Although our study primarily focuses on ticket triage, the potential of agentic AI in AIOps extends well beyond this domain. By leveraging the inherent strengths of AI agents, specifically their capabilities in contextual reasoning, autonomous decision-making, and tool invocation, AIOps can evolve toward more intelligent operational paradigms.

For example, in root cause analysis, multiple agents can collaborate to integrate insights from various data sources including logs, metrics, and traces, thereby enhancing both diagnostic accuracy and interpretability. By synthesizing

signals from different types of data, agents can uncover issues that a single agent might overlook, leading to more reliable anomaly detection and root cause analysis. Furthermore, in incident mitigation, equipping agents with the necessary tools enables the agent-based system to autonomously invoke remediation scripts and validate recovery outcomes through iterative feedback loops. This dynamic approach ensures more efficient incident mitigation, significantly reducing the time to mitigate (TTM) and minimizing the need for manual intervention.

Collectively, these directions highlight the transformative potential of agentic AI to drive end-to-end automation across the AIOps lifecycle.

7.4 Deployment Challenges

While our approach demonstrates promising performance in offline experiments, deploying it in real-world production environment introduces additional challenges due to the gap between offline testing and online operation. Offline experiments enable rapid trial-and-error validation, identifying effective strategies at relatively low cost. Once initial results are obtained, online deployment provides fast feedback from real usage, allowing iterative refinement and performance optimization. This offline–online iterative workflow underscores the value of combining controlled experimentation with real-world validation to ensure both effectiveness and adaptability in enterprise settings.

8 RELATED WORK

8.1 Triage in software engineering

In software engineering, triage is generally categorized into incident triage and bug triage, based on the types of objects being addressed, application scenarios, and core objectives [33, 50, 58]. Incident triage focuses on handling incident tickets reported for unexpected operational incidents in production environments. Its primary goals are to rapidly identify the scope of impact, assess the urgency of incidents, and triage them to the appropriate responsible teams, thereby minimizing service downtime and business loss. In contrast, bug triage targets the systematic processing of bug reports submitted during the software development and maintenance phases. Its core objective is to efficiently assign bugs to the most suitable developers or teams through a series of analytical processes, ensuring that defects are prioritized and accurately resolved under limited development resources.

Incident ticket triage. Ticket triage aims to build classifier models by leveraging both textual and non-textual features to achieve efficient and accurate ticket triage. Existing research has made notable progress in this area. For example, [23] improves triage accuracy through ensemble methods combining Naive Bayes, SVM, KNN, and Decision Tree. CNN Triager [28] employs Convolutional Neural Network and word embeddings for automated ticket triage, enhancing triage efficiency. DeepCT [7] utilizes GRU model to capture temporal dependencies in discussions, incorporates attention mechanisms to mitigate noise, and optimizes stage-wise predictions using a continuous loss function. This enables incremental knowledge learning, allowing accurate incident triage with fewer interactions and significantly reducing incident mitigation time. Additionally, [6] adopts deep learning to further improve the performance of ticket triage. DeepTriage [35] integrates multiple machine learning techniques to efficiently and automatically assign incidents to the appropriate teams. While Goel et al. [14] combines retrieval-based association mining with LLM reasoning. It integrates X-lifecycle data into GPT-4–based reasoning pipelines for root cause recommendation and monitoring classification, leveraging event semantics and dependency metadata to improve LLM-based ticket triage. COMET [44] combines domain knowledge with LLMs to extract keywords and predicts the responsible team based on the similarity between new and historical incident embeddings, thereby improving accuracy and reducing mitigation latency.

TickIt [31] introduces an LLM-based online ticket escalation framework that continuously understands dialogue content, performs topic-aware multi-class escalation, and exploits inter-ticket relationships, achieving efficient automated ticket escalation through feedback-driven fine-tuning. Triangle [51] proposes a multi-agent collaborative ticket triage method, which integrates semantic distillation, candidate team generation, and negotiation-based decision-making to automatically fuse multi-team domain knowledge and achieve high-accuracy ticket triage. Despite these advancements, existing ticket triage methods often rely on large models making them difficult to deploy under resource-constrained settings. Our approach, *CoTriage*, overcomes these limitations by transferring a large model's reasoning capability to a lightweight student model via knowledge distillation, enhanced with self-reinforcement and collaborative tuning, achieving efficient deployment and expert-level triage.

Bug triage. Recent research on bug triage is extensive and can generally be divided into two main categories: machine learning–based methods and information retrieval-based methods. Machine learning–based approaches typically formulate bug triage as a supervised classification problem. By learning patterns from historical bug reports, these methods build models to automatically complete the triage task. For example, Lee et al. [29] employed a convolutional neural network combined with Word2Vec embeddings to achieve automated bug triage. Dipongkor [11] proposed an ensemble method that integrates six Transformer-based LLMs, combined with voting and stacking techniques, and demonstrated that the ensemble outperforms individual models. In contrast, information retrieval–based approaches treat bug triage as a text matching task, assigning developers by measuring the similarity between new bug reports and historical reports or developer-related documents. For instance, Bugzie [41] combines fuzzy sets with caching mechanisms to model developers' fixing expertise, incrementally updates membership scores and aggregates fuzzy sets for new bugs, then ranks developers for triaging. Dretom [47] constructs topic models from historical bug resolution records to capture developers' interests and expertise in different bug types, and ranks developers accordingly for new bug reports. BugFixer [19] builds a Developer–Component–Bug network to model the relationships among them and ranks developers based on their connectivity and similarity to historical bug-fixing information. However, the inherent scale, complexity, and diverse failure patterns of cloud systems make incident triage significantly more challenging in real-world industrial environments.

8.2 LLMs in cloud system maintenance

In the realm of cloud system maintenance, LLMs have been widely applied to address various challenges, including incident management [15, 43], incident reporting [22, 52], root cause analysis [1, 8, 56, 57] and incident mitigation [2, 39]. NetAssistant [43] is a dialogue-based network diagnosis system that leverages natural language processing to understand user queries and performs diagnosis using predefined workflows built from network engineers' expertise. OASIS [22] generates human-readable system outage summaries to help maintenance engineers quickly grasp the context and severity of issues. mABC [56] is a collaborative framework based on multi-agent systems and a blockchain voting mechanism, which achieves efficient root cause analysis by standardizing task decomposition and collaboration through agent workflow. [39] leverages LLMs to perform contextual learning using a customized Ansible-based remediation dataset, automatically generating and executing Ansible playbooks to resolve various issues in microservices environments. Unlike previous approaches, we propose for the first time a collaborative method combining large and small models to achieve efficient ticket triage.

9 CONCLUSION

To improve the accuracy and efficiency of ticket triage, we propose *CoTriage*, an end-to-end automated ticket triage framework powered by LLMs. *CoTriage* leverages CoT knowledge distillation to transfer the advanced reasoning capabilities of LLMs to SLMs. Building on this foundation, *CoTriage* incorporates a self-reinforcement mechanism that iteratively refines the model's reasoning ability, further enhancing its triage performance. Additionally, to effectively extract key information from noisy ticket content, *CoTriage* utilizes the previously trained triage model as a reward model and employs DPO to fine-tune a ticket summarizer , thereby assisting in improving triage accuracy. Extensive experiments on a real-world ticket dataset from the production environment of a top-tier global online video service provider, ByteDance, demonstrate *CoTriage*'s effectiveness. Moreover, *CoTriage* has been successfully deployed in the STE team's ticket management system at ByteDance, significantly improving ticket triage efficiency. The average triage time per ticket has been reduced to 15.0 seconds, achieving a 13-fold efficiency gain compared to manual triage, and substantially lowering both labor and time costs.

References

[1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 1737–1749.

[2] Kaikai An, Fangkai Yang, Junting Lu, Liqun Li, Zhixing Ren, Hao Huang, Lu Wang, Pu Zhao, Yu Kang, Hua Ding, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024. Nissist: An Incident Mitigation Copilot based on Troubleshooting Guides. In *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024) (Frontiers in Artificial Intelligence and Applications, Vol. 392)*, Ulle Endriss, Francisco S. Melo, Kerstin Bach, Alberto José Bugarín Diz, Jose Maria Alonso-Moral, Senén Barro, and Fredrik Heintz (Eds.). IOS Press, 4471–4474.

[3] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram K. Rajamani, Balasubramanyan Ashok, and Shashank Shet. 2024. CodePlan: Repository-Level Coding using LLMs and Planning. *Proc. ACM Softw. Eng.* 1, FSE (2024), 675–698.

[4] Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. 2024. Augmenting large language models with chemistry tools. *Nat. Mac. Intell.* 6, 5 (2024), 525–535. doi:10.1038/S42256-024-00832-8

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[6] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, Helen Sharp and Mike Whalen (Eds.). IEEE / ACM, 111–120. doi:10.1109/ICSE-SEIP.2019.00020

[7] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 364–375. doi:10.1109/ASE.2019.00042

[8] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 674–688. doi:10.1145/3627703.3629553

[9] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, Yingnong Dang, Feng Gao, Pu Zhao, Bo Qiao, Qingwei Lin, Dongmei Zhang, and Michael R. Lyu. 2020. Towards intelligent incident management: why we need it and how we make it. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1487–1497. doi:10.1145/3368089.3417055

[10] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *CoRR* abs/2501.12948 (2025). arXiv:2501.12948

[11] Atish Kumar Dipongkor. 2024. An Ensemble Method for Bug Triaging using Large Language Models. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 438–440. doi:10.1145/3639478.3641228

[12] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and et al. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024). arXiv:2407.21783

[13] Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2024. GPTScore: Evaluate as You Desire. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (Eds.). Association for Computational Linguistics, 6556–6576.

[14] Drishti Goel, Fiza Husain, Aditya Singh, Supriyo Ghosh, Anjaly Parayil, Chetan Bansal, Xuchao Zhang, and Saravan Rajmohan. 2024. X-Lifecycle Learning for Cloud Incident Management using LLMs. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, Marcelo d'Amorim (Ed.). ACM, 417–428. doi:10.1145/3663529.3663861

[15] Pouya Hamadanian, Behnaz Arzani, Sadjad Fouladi, Siva Kesava Reddy Kakarla, Rodrigo Fonseca, Denizcan Billor, Ahmad Cheema, Edet Nkposong, and Ranveer Chandra. 2023. A Holistic View of AI-driven Network Incident Management. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks, HotNets 2023, Cambridge, MA, USA, November 28-29, 2023*. ACM, 180–188.

[16] Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large Language Models Are Reasoning Teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 14852–14882.

[17] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

[18] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 8003–8017.

[19] Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. 2014. Effective Bug Triage Based on Historical Bug-Fix Information. In *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014, Naples, Italy, November 3-6, 2014*. IEEE Computer Society, 122–132. doi:10.1109/ISSRE.2014.17

[20] Kung-Hsiang Huang, Philippe Laban, Alexander R. Fabbri, Prafulla Kumar Choubey, Shafiq Joty, Caiming Xiong, and Chien-Sheng Wu. 2024. Embrace Divergence for Richer Insights: A Multi-document Summarization Benchmark and a Case Study on Summarizing Diverse Information from News Articles. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (Eds.). Association for Computational Linguistics, 570–593. doi:10.18653/V1/2024.NAACL-LONG.32

[21] Tatsuro Inaba, Hirokazu Kiyomaru, Fei Cheng, and Sadao Kurohashi. 2023. MultiTool-CoT: GPT-3 Can Use Multiple External Tools with Chain of Thought Prompting. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 1522–1532. doi:10.18653/V1/2023.ACL-SHORT.130

[22] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, Shilin He, Federica Sarro, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Assess and Summarize: Improve Outage Understanding with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, Satish Chandra, Kelly Blincoe, and Paolo Tonella (Eds.). ACM, 1657–1668. doi:10.1145/3611643.3613891

[23] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. 2016. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empir. Softw. Eng.* 21, 4 (2016), 1533–1578. doi:10.1007/S10664-015-9401-9

[24] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.).

[25] Ahmed Lamkanfi, Javier Pérez, and Serge Demeyer. 2013. The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 203–206.

[26] Hung Le, Hailin Chen, Amrita Saha, Akash Gokul, Doyen Sahoo, and Shafiq Joty. 2024. CodeChain: Towards Modular Code Generation Through Chain of Self-revisions with Representative Sub-modules. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=vYhglxSj8j

[27] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. 2024. RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

[28] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 926–931. doi:10.1145/3106237.3117776

[29] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 926–931. doi:10.1145/3106237.3117776

[30] Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. 2023. Symbolic Chain-of-Thought Distillation: Small Models Can Also "Think" Step-by-Step. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 2665–2679.

[31] Fengrui Liu, Xiao He, Tieying Zhang, Jianjun Chen, Yi Li, Lihua Yi, Haipeng Zhang, Gang Wu, and Rui Shi. 2025. TickIt: Leveraging Large Language Models for Automated Ticket Escalation. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, FSE Companion 2025, Clarion Hotel Trondheim, Trondheim, Norway, June 23-28, 2025*, Leonardo Montecchi, Jingyue Li, Denys Poshyvanyk, and Dongmei Zhang (Eds.). ACM, 343–354. doi:10.1145/3696630.3728558

[32] N.R. Murphy, B. Beyer, C. Jones, and J. Petoff. 2016. *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media.

[33] Naresh Kumar Nagwani and Jasjit S. Suri. 2023. An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review. *Int. J. Inf. Manag. Data Insights* 3, 1 (2023), 100153. doi:10.1016/J.JJIMEI.2022.100153

[34] Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolò Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, Renqian Luo, Scott Mayer McKinney, Robert Osazuwa Ness, Hoifung Poon, Tao Qin, Naoto Usuyama, Christopher M. White, and Eric Horvitz. 2023. Can Generalist Foundation Models Outcompete Special-Purpose Tuning? Case Study in Medicine. *CoRR* abs/2311.16452 (2023). arXiv:2311.16452 doi:10.48550/ARXIV.2311.16452

[35] Phuong Pham, Vivek Jain, Lukas Dauterman, Justin Ormont, and Navendu Jain. 2020. DeepTriage: Automated Transfer Assistance for Incidents in Cloud Services. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 3281–3289. doi:10.1145/3394486.3403380

[36] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 15174–15186. doi:10.18653/V1/2024.ACL-LONG.810

[37] Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. Is ChatGPT a General-Purpose Natural Language Processing Task Solver?. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 1339–1384. doi:10.18653/V1/2023.EMNLP-MAIN.85

[38] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.).

[39] Komal Sarda, Zakeya Namrud, Marin Litoiu, Larisa Shwartz, and Ian Watts. 2024. Leveraging Large Language Models for the Auto-remediation of Microservice Applications: An Experimental Study. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, Marcelo d'Amorim (Ed.). ACM, 358–369.

[40] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2024. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. *IEEE Trans. Software Eng.* 50, 1 (2024), 85–105.

[41] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2011. Fuzzy set and cache-based approach for bug triaging. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, Tibor Gyimóthy and Andreas Zeller (Eds.). ACM, 365–375. doi:10.1145/2025113.2025163

[42] Lewis Tunstall, Nils Reimers, Unso Eun Seo Jo, Luke Bates, Daniel Korat, Moshe Wasserblat, and Oren Pereg. 2022. Efficient Few-Shot Learning Without Prompts. *CoRR* abs/2209.11055 (2022). arXiv:2209.11055 doi:10.48550/ARXIV.2209.11055

[43] Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin, Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu, Wei Zhou, Yongbin Dong, Weirong Jiang, and Yi Wang. 2024. NetAssistant: Dialogue Based Network Diagnosis in Data Center Networks. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*, Laurent Vanbever and Irene Zhang (Eds.). USENIX Association.

[44] Zexin Wang, Jianhui Li, Minghua Ma, Ze Li, Yu Kang, Chaoyun Zhang, Chetan Bansal, Murali Chintalapati, Saravan Rajmohan, Qingwei Lin, Dongmei Zhang, Changhua Pei, and Gaogang Xie. 2024. Large Language Models Can Provide Accurate and Interpretable Incident Triage. In *35th IEEE International Symposium on Software Reliability Engineering, ISSRE 2024, Tsukuba, Japan, October 28-31, 2024.* IEEE, 523–534. doi:10.1109/ISSRE62328.2024.00056

[45] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.).

[46] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[47] Xihao Xie, Wen Zhang, Ye Yang, and Qing Wang. 2012. DRETOM: developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering, PROMISE '12, Lund, Sweden, September 21-22, 2012*, Stefan

Wagner (Ed.). ACM, 19–28. doi:10.1145/2365324.2365329

[48] Canwen Xu, Daya Guo, Nan Duan, and Julian J. McAuley. 2023. Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 6268–6278.

[49] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[50] Qingyang Yu, Nengwen Zhao, Mingjie Li, Zeyan Li, Honglin Wang, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2024. A survey on intelligent management of alerts and incidents in IT services. *J. Netw. Comput. Appl.* 224 (2024), 103842. doi:10.1016/J.JNCA.2024.103842

[51] Zhaoyang Yu, Aoyang Fang, Minghua Ma, Jaskaran Singh Walia, Chaoyun Zhang, Shu Chi, Ze Li, Murali Chintalapati, Xuchao Zhang, Rujia Wang, Chetan Bansal, Saravan Rajmohan, Qingwei Lin, Shenglin Zhang, Dan Pei, and Pinjia He. 2025. Triangle: Empowering Incident Triage with Multi-Agent. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 674–686. doi:10.1109/ASE63991.2025.00062

[52] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. 2024. MonitorAssistant: Simplifying Cloud Service Monitoring via Large Language Models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, Marcelo d'Amorim (Ed.). ACM, 38–49. doi:10.1145/3663529.3663826

[53] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-Rewarding Language Models. *CoRR* abs/2401.10020 (2024). arXiv:2401.10020

[54] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. 2023. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. *CoRR* abs/2305.04207 (2023). arXiv:2305.04207

[55] Biao Zhang, Barry Haddow, and Alexandra Birch. 2023. Prompting Large Language Model for Machine Translation: A Case Study. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 41092–41110.

[56] Wei Zhang, Hongcheng Guo, Jian Yang, Zhoujin Tian, Yi Zhang, Chaoran Yan, Zhoujun Li, Tongliang Li, Xu Shi, Liangfan Zheng, and Bo Zhang. 2024. mABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, 4017–4033.

[57] Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. 2024. Automated Root Causing of Cloud Incidents using In-Context Learning with GPT-4. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, Marcelo d'Amorim (Ed.). ACM, 266–277. doi:10.1145/3663529.3663846

[58] Yongxin Zhao, Shenglin Zhang, Yujia Wu, Yuxin Sun, Yongqian Sun, Dan Pei, Chetan Bansal, and Minghua Ma. 2025. Triage in Software Engineering: A Systematic Review of Research and Practice. *CoRR* abs/2511.08607 (2025). arXiv:2511.08607 doi:10.48550/ARXIV.2511.08607

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009