LLM-Powered Multi-Agent Collaboration for Intelligent Industrial On-Call Automation

Ruowei Fu[†], Yang Zhang[‡], Zeyu Che[†], Xin Wu[‡], Zhenyu Zhong[†], Zhiqiang Ren[‡],
Shenglin Zhang[†§*], Feng Wang[‡], Yongqian Sun[†¶], Xiaozhou Liu[‡], Kexin Liu[†], and Yu Zhang[‡]
[†]Nankai University, {furuowei, chezeyu, zyzhong, liukx}@mail.nankai.edu.cn, {zhangsl, sunyongqian}@nankai.edu.cn
[‡]ByteDance Inc., {zhangyang.329, wuxin.29, renzhiqiang.marvin, wangfeng.ai, wangding.01, felix.zhang}@bytedance.com
[§]Key Laboratory of Data and Intelligent System Security, Ministry of Education, China
[¶]Tianjin Key Laboratory of Software Experience and Human Computer Interaction

***Abstract*—In large-scale enterprises, on-call engineers (OCEs) are critical for ensuring service availability and reliability. However, as incidents grow in volume and complexity, traditional manual on-call processes are becoming increasingly inadequate. Recent advances in large language models (LLMs) have demonstrated remarkable capabilities in reasoning and multi-agent collaboration, presenting new opportunities for automation. We propose OncallX, an end-to-end automated on-call system designed for real-world industrial scenarios that integrates LLMs with multi-agent cooperation to enable intelligent and efficient incident management. OncallX first enhances user queries by leveraging external knowledge bases and multi-turn dialogue interactions. Subsequently, multiple expert agents collaborate through tree-search-based mechanisms to generate effective responses and solutions. When incidents cannot be resolved automatically, OncallX accurately assigns them to the most appropriate teams. Comprehensive experiments conducted in the real-world production environment of a top-tier global online video service provider demonstrate that OncallX efficiently responds to incidents and accurately triages tickets, significantly outperforming existing methods in both automated metrics and human evaluations. Furthermore, OncallX has been successfully deployed in production for two months, during which it has substantially enhanced on-call efficiency, reducing average incident response time to just 21 seconds and average triage time to 4 seconds—representing a transformative improvement in operational excellence.**

***Index Terms*—On-Call, Large Language Model, Multi-Agent**

I. Introduction

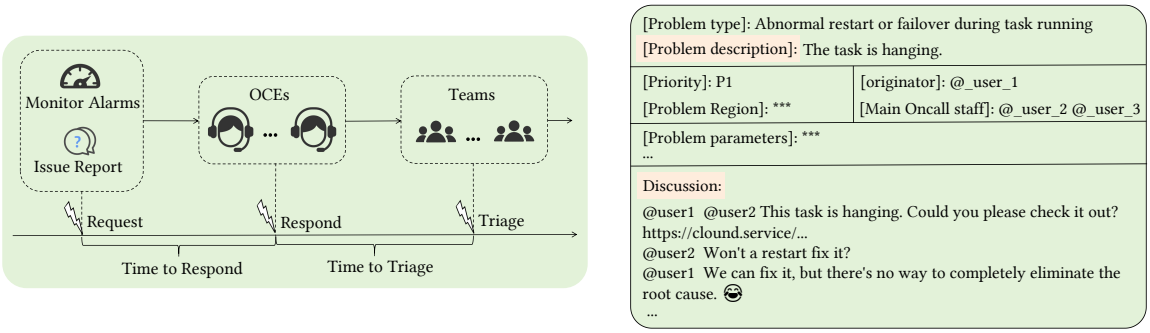
On-call engineers (OCEs) are responsible for handling urgent technical issues and emergencies, including system alerts and troubleshooting critical problems [1]. They serve as the frontline defenders of system reliability, ensuring timely incident management during exceptions to minimize business disruptions and mitigate operational risks. As enterprise IT infrastructure continues to expand in scale and complexity, modern systems now encompass thousands of services, distributed architectures, cross-regional data centers, and diverse technology stacks [2]–[4]. In such environments, OCEs must rapidly respond to a wide spectrum of emergencies—from kernel failures and network delays to database anomalies—while providing accurate and effective solutions [5].

* Shenglin Zhang is the corresponding author.

However, a significant gap exists between the limited capabilities of manual on-call processes and the increasingly demanding operational requirements. First, manual on-call response speed heavily depends on individual OCE experience and expertise. Developing a competent OCE typically requires months or even years of intensive training, encompassing both systematic technical documentation study and extensive hands-on incident handling experience [6]. Second, manual processes are constrained by human workload limitations, potentially preventing timely responses during critical situations. Such delays can escalate minor issues into major incidents, resulting in substantial business losses. For instance, a single hour of Amazon.com downtime could translate to direct revenue losses exceeding $100 million [7]. Consequently, on-call automation has emerged as an urgent enterprise need to enhance operational efficiency while alleviating the burden on human operators.

Recent advances in large language models (LLMs), including GPT-4-Turbo [8] and LLaMA [9], have demonstrated remarkable capabilities across various natural language processing (NLP) tasks. However, individual LLMs still struggle with complex tasks requiring intensive domain knowledge and sophisticated reasoning [10]–[12]. In contrast, human teams effectively address on-call challenges through collaborative problem-solving, where domain specialists complement each other's knowledge limitations, tackle cross-domain issues through coordinated expertise, and enhance solution quality through diverse perspectives and specialized knowledge integration.

Inspired by this collaborative paradigm, we propose **OncallX**, a multi-agent collaborative framework that automates two critical tasks spanning the complete on-call lifecycle: **(1) Efficient incident response**, where the framework dynamically orchestrates domain expert agents to collaboratively address user queries, enabling rapid and accurate incident resolution; and **(2) Accurate ticket triage**, which serves as the final safeguard by assigning incidents that cannot be automatically resolved to appropriate specialized teams for timely human intervention. Together, these components address our core research objective of enabling efficient on-call incident management. However, realizing this vision presents three core



(a) Human on-call workflow

(b) Example of incident ticket

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

challenges (detailed in Section III).

Challenge 1: How to enhance LLMs' understanding of ambiguous on-call queries? Due to limited domain-specific knowledge and the inherent ambiguity in on-call incident descriptions, LLMs often struggle to accurately discern user intent, leading to imprecise and inefficient incident responses. On-call queries frequently contain technical jargon, incomplete information, and context-dependent references that require domain expertise to interpret correctly.

Challenge 2: How to improve the accuracy of LLMs' responses to complex on-call incidents? The complexity and diversity of on-call scenarios often exceed the capabilities of individual LLMs. Unlike many conventional LLM applications [13], on-call incidents are highly unpredictable, lack clear domain boundaries, and resist decomposition into predefined modules. Effective resolution requires coordinated expertise from multiple domains, necessitating sophisticated collaboration mechanisms among specialized LLM agents.

Challenge 3: How to enable efficient and accurate LLM-based ticket triage? While LLMs demonstrate strong language understanding capabilities, their direct application to ticket triage faces significant challenges. The presence of textual noise (e.g., formatting artifacts, irrelevant metadata) and the large category space of potential teams/domains create both accuracy and efficiency bottlenecks that hinder practical deployment.

To address these challenges, this paper presents **OncallX**, an end-to-end automated on-call system that utilizes multi-agent LLM collaboration. For Challenge 1, OncallX combines knowledge base retrieval with multi-turn dialogue to systematically refine and clarify user intent. For Challenge 2, we employ a tree-search-based multi-agent collaboration mechanism that enables specialized agents to coordinate their expertise effectively. For Challenge 3, we develop a knowledge graph-enhanced approach that filters textual noise and provides structured domain guidance for accurate and efficient ticket triage.

The main contributions of this work are as follows:

(1) Novel end-to-end multi-agent framework for on-call automation. To the best of our knowledge, OncallX represents the first comprehensive on-call system designed around a multi-LLM-agent architecture that automates the complete on-call workflow, from initial incident response to final ticket triage.

(2) Comprehensive solution addressing key automation challenges. We develop three innovative modules that collectively tackle the core challenges of automated on-call systems: (i) a user intent enhancement module that leverages domain knowledge and multi-turn dialogue to improve query understanding; (ii) a tree-search-based multi-agent orchestration mechanism that enables effective collaborative problem-solving; and (iii) a knowledge graph-enhanced ticket triage approach that achieves high accuracy and efficiency without requiring additional model training.

(3) Real-world validation and deployment success. We demonstrate OncallX's practical effectiveness through comprehensive experiments conducted on real System Technologies & Engineering (STE) scenarios at a top-tier global online video service provider *B*. The system has been successfully deployed in production for two months, achieving transformative performance improvements with average incident response times of 21 seconds and ticket triage times of 4 seconds, representing significant operational enhancements over traditional manual processes.

II. BACKGROUND

A. On-Call Workflow

On-call is a critical operational mechanism designed to ensure the stability of business systems and the timely resolution of issues. It generally comprises three key stages: incident reporting, incident response and ticket triage, as illustrated in Figure 1a.

1) Incident Reporting: In production environments, the continual evolution of business requirements, frequent system updates, and iterative improvements make incidents inevitable. When an issue arises, engineers can manually submit a ticket to the OCE teams. As shown in Figure 1b, each ticket comprehensively records the necessary information for triage, diagnosis, and resolution of the issue. Tickets typically include the following fields: problem type, problem description, priority (set according to its potential impact on business), originator, problem parameters, and discussion, among others.

2) Incident Response: Incident response refers to the process by which OCEs promptly investigate user-reported issues or system-generated alerts and strive to implement effective mitigation strategies. In practice, OCEs typically rely on alert data, system logs, and monitoring metrics to rapidly identify the root cause of an incident. Mitigation actions are then executed based on predefined runbooks or the engineers' operational expertise. To illustrate typical failure scenarios in this process, we provide manually labeled error analysis reports in Figure 2, which highlight two most frequent types of failures: fault-related issues and consultation requests. The primary objective of this phase is to resolve the incident as quickly as possible, thereby minimizing disruptions to system stability and user experience. In most cases, incidents are effectively mitigated at this stage, preventing unnecessary escalation.

Query: Server downtime.

Manually labeled error analysis report:
Root Cause: The fault location was observed in dmesg. It is preliminarily determined to be an issue with a GPU, but the specific GPU has not been identified.
Solution: 1. Try restarting the system to see if the GPU recovers after being reset. 2. Provide the kernel logs (preferably crash logs; if unavailable, provide the full dmesg output, at least including the logs around the reboot) for analysis of the anomaly. 3. Verify the driver version.

Query: How to evaluate a program's performance on a dual-NIC machine under single-NIC versus dual-NIC configurations?

Manually labeled error analysis report:
Root Cause: None.
Solution: 1. It is recommended to try namespace-isolated testing. For example, use 'ip netns add ns1' to add a namespace, 'ip link set eth1 netns ns1' to assign the network interface to the namespace, 'ip netns exec ns1 bash' to enter the namespace, 'ip route add default xxx' to add a route inside the namespace, and then ping the target address. 2. Propose different testing modes, such as: Mode 1: Bring down the 'eth2' interface, and set 'eth0' affinity on 'numa0'. Mode 2: Set 'eth0' affinity on the first 24 CPUs of 'numa0' and 'eth2' affinity on the last 24 CPUs of 'numa0'.

Fig. 2: Illustration of manually labeled error analysis reports.

3) Ticket Triage: Ticket triage refers to the process in which, when OCEs are unable to resolve an issue within their scope of responsibility, triage engineers review the ticket details and forward it to the appropriate team for further handling [2], [3], [7]. Each ticket typically contains the incident context and engineers' discussions to help the receiving team quickly understand and take over. In practice, triage mechanisms may support automatic escalation based on incident severity to ensure timely response and resolution. However, due to the high complexity and interdependencies of systems, tickets are frequently misrouted to incorrect teams, leading to prolonged service downtime.

B. Multi-Agent Collaboration

In recent years, LLMs have emerged as pervasive and foundational technologies in the field of NLP, owing to their remarkable performance. Nevertheless, single LLM possess inherent limitations in addressing complex tasks, chiefly due to their inability to effectively collaborate with other agents and to acquire knowledge through social interactions [14]. These constraints impede their capacity to leverage multi-turn feedback for continual learning and performance improvement.

To address these challenges, researchers have proposed multi-LLM-agent systems [13], [15], [16], which incorporate multiple autonomous agents. Each agent focuses on a specific subtask and is equipped with its own knowledge base, toolset, and behavioral strategy. By engaging in efficient information sharing and human-like collaboration, these agents collectively leverage their specialized expertise to solve complex problems, effectively overcoming the limitations of single-agent systems. This architecture significantly enhances the system's robustness, adaptability, and overall effectiveness in handling real-world complexity.

III. MOTIVATION

The STE team of *B* specializes in core system technologies, including operating system kernels, virtualization, foundational system software, and the stability of large-scale data centers. Its OCEs are responsible for handling a high volume of system failures and operating system related issues on a daily basis, with a commitment to delivering reliable and efficient technical support for upper-layer business services. The stark imbalance between the overwhelming workload and the limited number of OCEs presents a significant challenge to timely incident response. Moreover, the complexity and diversity of incidents often make it difficult for even highly experienced OCEs to respond accurately and promptly. Therefore, to alleviate the burden of manual on-call duties and improve response efficiency, the development of a more automated on-call system has become an urgent necessity.

To address this, we propose an end-to-end automated on-call system powered by LLMs and collaborative multi-agent interactions. This system integrates multiple expert agents that actively participate across the entire on-call lifecycle to ensure accurate and efficient incident response and ticket triage. However, it confronts the following three challenges.

A. Challenge 1: How to enhance LLMs' understanding of ambiguous on-call queries?

IT operations is inherently complex, involving a vast amount of specialized terminologies, intricate operational workflows, as well as subtle semantic nuances that depend heavily on specific contextual environments. LLMs trained on general-purpose corpora struggle to fully comprehend these domain-specific knowledge [17]–[19], resulting in limited performance in real-world IT operations scenarios.

This challenge is particularly evident in on-call scenarios, where users face intense time pressure and often report issues in brief, vague, or incomplete ways. For example, a user might simply input, a terse phrase like "GWPAsan related questions." Such expressions are inherently ambiguous and lack the necessary contextual information, making it difficult for LLMs to accurately interpret the specific meaning of "GWPAsan" and the user's intent. Due to insufficient domain knowledge and contextual understanding, LLMs cannot effectively determine which expert agent to invoke and what actions to take, often responding with failure messages such as "Unable to identify user intent. Please provide more information."

To address this, we propose a user intent enhancement module that leverages domain knowledge and employs a specially designed intent clarification agent. Through multi-turn dialogue, this agent assists users in refining and completing their problem descriptions.

B. Challenge 2: How to improve the accuracy of LLMs' responses to complex on-call incidents?

Multi-agent collaboration can generally be categorized into adversarial and cooperative types. Adversarial interaction achieves shared goals through agent competition [20]–[22], but often incurs prolonged debates, increased latency, and significant computational overhead. Moreover, agents may converge on an incorrect consensus, each confidently believing it to be correct [21]. In contrast, cooperative multi-agent systems are more suitable for scenarios that emphasize long-term collaboration, limited resources, and system stability. Cooperative interaction can be further divided into disordered [23], [24] and ordered [13], [15], [16] cooperation. Disordered cooperation refers to the introduction of a dedicated coordinating agent within a multi-agent system, which is responsible for aggregating and organizing the responses from all agents, thereby continuously updating and refining the final answer. In contrast, ordered cooperation involves agents following specific rules to engage in a highly structured and sequential discussion, where each agent expresses their opinion in turn to collaboratively produce the final answer.

In on-call scenarios, incidents are highly diverse and unpredictable, with vague responsibility boundaries and module definitions, making rule-based approaches inadequate. Therefore, when confronting complex on-call issues, agents must possess strong multi-turn interaction and dynamic reasoning abilities, enabling them to continuously track and comprehend contextual information while flexibly adjusting decision-making strategies in response to real-time environmental changes.

To address this challenge, we propose a tree-search-based multi-agent orchestration framework designed to guide LLMs in exhaustively exploring potential solution paths. Additionally, by incorporating a reflection mechanism, LLMs can backtrack to previous steps when the current action fails or yields no valuable information, enabling effective correction of the solution paths.

C. Challenge 3: How to enable efficient and accurate LLM-based ticket triage?

In real-world production environments, the continuous evolution of ticket categories resulting from ongoing business changes renders the training and maintenance of a static ticket classifier impractical. Therefore, we aim to leverage the powerful language understanding and reasoning capabilities of LLMs to support ticket triage. However, directly applying LLMs to this task poses challenges in two key aspects.

Textual Noise Interference. In ticket triage, textual information (such as titles and engineers' discussions) is crucial for identifying the appropriate team. However, discussion quality is inconsistent, often including disorganized expressions, ambiguous phrases, images, or links. Inputting raw tickets directly into an LLM may cause the model to hallucinate due to noisy inputs, overly relying on the beginning and end of the text while neglecting important information in the middle [25], [26]. Thus, effective noise filtering and key information extraction are essential for improving triage accuracy.

Large Category Space. In practice, the number of ticket categories is often very large. Asking an LLM to choose the correct category from the full set significantly increases reasoning complexity [27]. To enhance LLMs' classification performance, it is necessary to impose constraints on the candidate category space.

To address this, we propose a two-step strategy: first, a LLM is employed to summarize the row tickets, effectively reducing noise and irrelevant content, and then a knowledge graph is used to constrain the candidate category space. This strategy improves the triage accuracy of the LLM by directly leveraging discriminative information and reducing the number of possible categories.

IV. METHODOLOGY

A. Overview

As shown in Figure 3, *OncallX* consists of three modules. *Module 1*, upon receiving an incident ticket, the system first extracts key technical terms from the user's query and retrieves relevant knowledge from domain knowledge bases. This knowledge is provided to the ClarifyAgent, which determines whether the user's intent is sufficiently clear. If the intent is incomplete, the agent initiates follow-up questions to help the user fill in missing details. If the intent is deemed complete, the dialogue ends, resulting in a well-formed query for the OCEAgent. *Module 2*, the OCEAgent employs a tree-based search strategy to coordinate multiple domain expert agents and explore diverse solution paths. A reflection mechanism guides the LLM to backtrack previous steps when no useful information is found to improve search efficiency. *Module 3*, if the OCEAgent fails to resolve the incident, the TriageAgent assigns the ticket to the appropriate expert team, they will collaborate via group chat to further address the problem.

B. User Intent Enhancement Module

OCE teams are often approached by non-expert users seeking assistance. These queries typically lack sufficient context, are semantically vague, and involve domain terminology that general-purpose LLMs struggle to understand due to limited domain expertise. As a result, LLMs often fail to accurately grasp the user's true intent, which can result in suboptimal or erroneous solutions. To address this, *OncallX* enhances intent understanding through two key mechanisms at this stage.

Domain Knowledge Augmentation. *OncallX* employs a retrieval augmented generation approach that strategically combines the strengths of both knowledge retrieval with the language understanding capabilities of LLM. Specifically, when a query is received, *OncallX* first utilizes the LLM to extract domain terms (e.g., "GWPAsan", "kdump", "clang11"), which are then used as targeted search keys to query a

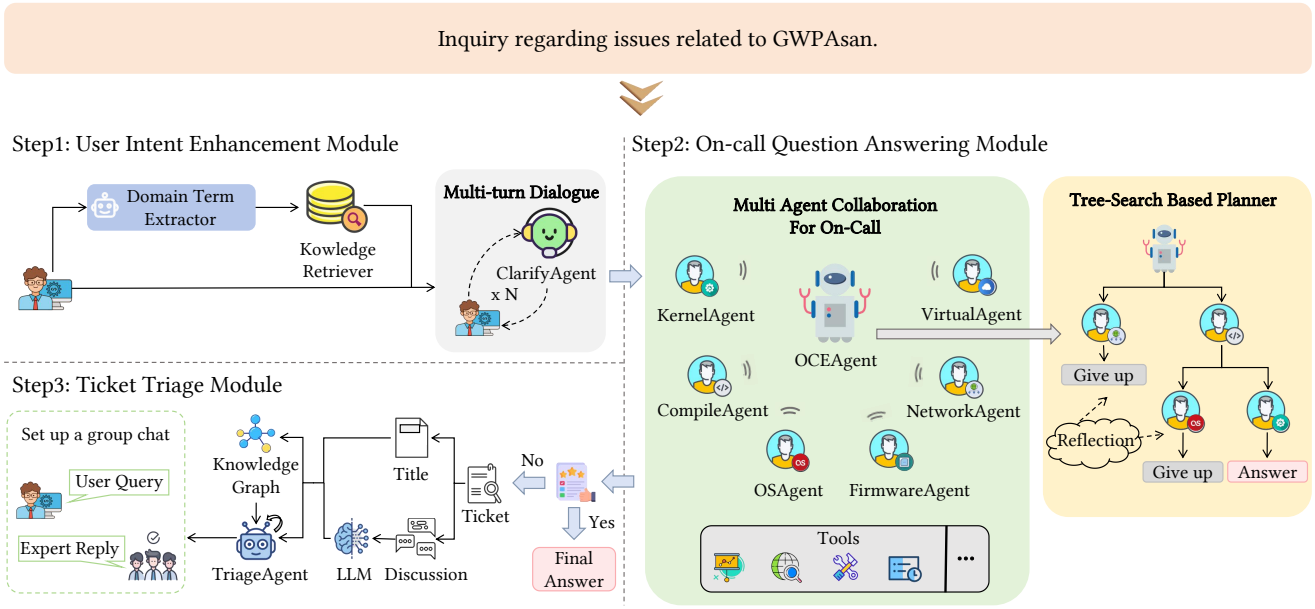


Fig. 3: The framework of *OncallX*.

curated professional knowledge base comprising authoritative documentation, historical incident reports, and expert insights. The knowledge base is dynamically updated through two mechanisms: resolved tickets are archived into the historical incident repository, and domain documents are periodically refreshed to ensure the timeliness and reliability of the content. The retrieved, contextually relevant information is seamlessly integrated with the original user query, effectively enriching the input data that is fed into the LLM for further processing.

Multi-Turn Dialogue for Intent Clarification. In real-world production environment, engineers frequently submit questions that are concise, ambiguous, or lacking crucial details, typically due to the intense time constraints. Such incomplete queries present significant challenges for LLM, which struggles to accurately interpret the user's true intent without sufficient contextual information. To address this, *OncallX* introduces a specialized agent called the ClarifyAgent. This agent leverages carefully designed, context-aware prompts to engage users in multi-turn interactive dialogue, with the goal of systematically eliciting essential information that may have been omitted, such as broader incident context, specific error messages, system states, and the user's core concerns. By iteratively refining the initial query in this manner, *OncallX* significantly enhances the LLM's ability to comprehend the user intent more precisely, thereby improving the accuracy of incident response.

C. On-Call Question Answer Module

To efficiently and accurately address user queries, we propose a multi-agent collaborative framework for on-call question answering. This framework employs a tree-based search strategy to orchestrate multiple expert agents, guiding the LLMs to comprehensively explore potential solution paths. A reflection mechanism is further introduced to enable LLMs to backtrack to previous steps when the current operation fails or yields no valuable information, thereby achieving effective correction and optimization of solution paths.

1) Design of Expert Agents: In designing expert agents, we employ a general agent construction strategy to enable efficient and scalable development [13], [28], [29]. These agents harness the robust capabilities of LLMs, displaying advanced competencies in language comprehension, planning, and tool use, enabling them to reason effectively in complex on-call scenarios. Engineers can rapidly assemble agents with well-defined roles and task scopes by modularly selecting and configuring a set of built-in components. The prompt design example for an expert agent is shown in Figure 4.

Each expert agent is assigned a clearly defined role description, which serves to constrain its operational boundaries and guide its task execution. This role-oriented design paradigm ensures behavioral consistency and maintains domain-specific expertise, both of which are essential for effective collaboration in multi-agent systems. To further support their specialized functions, expert agents are equipped with domain tools. For instance, the CompileAgent is provisioned with tools such as a compilation knowledge base retriever and a user self-diagnosis manual. These tools enable the agent to accurately and efficiently address user queries, improving its reliability in real-world applications.

2) Design of the Multi-Agent Planner: To effectively orchestrate multi-agent collaboration for on-call problem solving, we adopt a tree-search–based planning strategy. In *OncallX* framework, OCEAgent serves as a central planner, responsible for interpreting user query, dispatching expert agents, coordinating multi-step reasoning processes, and integrating responses to generate a final solution.

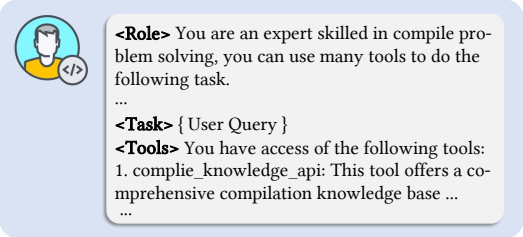


Fig. 4: Example prompt for expert agent.

Upon receiving a user query enhanced by ClarifyAgent, the OCEAgent leverages the language comprehension capabilities of LLM to infer user intent and select appropriate expert agents to handle corresponding subtasks. Each selected expert agent invokes its domain tools to execute the task and returns the results to the OCEAgent. The overall collaboration unfolds iteratively as a tree-structured search process, consisting of three key steps.

Step1: Planning. We have carefully designed and configured reflection-oriented prompts for OCEAgent (e.g., "if you can't handle the task, please give up and restart"). These prompts are intended to guide OCEAgent in actively evaluating the current context and history actions, enabling dynamic self-assessment and strategic adjustment. During the planning process, if the OCEAgent determines that a previous action has failed or produced uninformative results, it will automatically backtrack to an earlier decision point and replan accordingly. Conversely, if a useful outcome is obtained, the agent leverages the successful result to expand new search nodes, thereby further advancing the problem-solving process. This mechanism improves decision quality, reduces unnecessary computational overhead caused by unproductive paths, and enhances overall efficiency.

Step2: Execution. During the execution phase, OCEAgent distributes specific subtask instructions to the corresponding expert agents based on the task allocation determined in the planning stage. Upon receiving the instructions, the selected expert agent autonomously invokes its equipped tools (e.g., knowledge retriever, IP query tool) to perform the assigned operations and generate intermediate result. The result is subsequently returned to the OCEAgent and serve as guidance for subsequent planning decisions.

Step3: Integration. Upon receiving intermediate results from expert agents, the OCEAgent incorporates the information into its maintained list of historical messages, thereby continuously refining the system's understanding of the current problem state.

The process iterates until the OCEAgent identifies a satisfactory solution and outputs the final answer, or terminates when no viable solution can be found. To prevent infinite exploration, the search space is constrained by predefined depth and width limits, ensuring that the tree-search-based planning process remains bounded and computationally tractable.

D. Ticket Triage Module

Tickets comprise both title and discussions. In this stage, *OncallX* filters out noise from tickets and constructs a knowledge graph (KG) to assist LLM in achieving more accurate ticket triage.

Noise Reduction. The title and discussions of a ticket usually provide a textual description of the incident's symptoms, which play a critical role in determining the appropriate team for handling the incident. However, the quality of these discussions varies greatly, and they often include irrelevant or redundant information that acts as noise, hindering effective triage. LLM have demonstrated strong capabilities in summarizing and refining textual information. Therefore, TriageAgent feeds the raw discussion content into a LLM and prompts it to generate a concise summary, enhancing readability and facilitating accurate decision-making. Finally, the title and summarized discussions are concatenated to serve as input for triage.

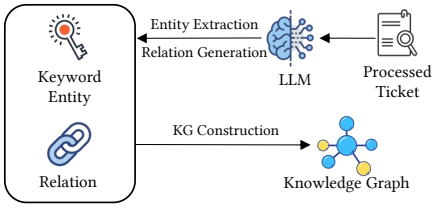


Fig. 5: The KG construction framework of *OncallX*.

Knowledge Graph Construction. As shown in Figure 5, the construction of the KG involves three key stages: entity extraction, relation generation, and graph database storage. To construct the KG, we leverage an LLM to extract key entities (e.g., "jemalloc", "velinux", "coredump") from historical tickets and to generate semantic relations between these entities (e.g., "belongs to", "occurs", "leads to") based on contextual understanding. Specifically, to enhance the relevance of the KG for ticket triage, we extract entities and relations that are semantically associated with the ticket's category. To achieve this, we adopt a label-informed prompting strategy, where the ticket category label is used as a weak supervisory signal during the LLM prompting process. This guides the LLM to focus on extracting entities and relations most pertinent to triage. Furthermore, as illustrated in Figure 6, to improve the quality of entity–relation extraction, we incorporate a few-shot prompting design, enabling the LLM to learn the mapping pattern from unstructured ticket data to structured triples. Subsequently, to support the efficient storage and querying of large-scale and complex relational information, we adopted Neo4j [30], a graph-based database. It supports efficient query execution through the Cypher Query Language (CQL), enabling fast graph traversal.

Online Multi-Round Ticket Triage. In the online ticket triage, we propose an efficient multi-round semantic-driven triage approach that integrates LLM with KG. The online process is mainly divided into three rounds. Specifically, (1) we design domain-knowledge-enhanced prompts to guide

Instruction
You are a system technology expert with extensive experience in extracting key entities from ticket texts and generating relations among them. I will provide a text along with its corresponding category. Based on this category, please extract the key entities most relevant to it and generate the relations between these entities. Return the output in JSON format.

Example
Please refer to the following examples.
Example 1:
Ticket Text: An error occurred with RapidJSON when compiling the codebase locally.
Text Category: Compilation/C++
Output:
[{"head": "RapidJSON", "relation": "belongs to", "tail":"Compilation/C++"}, {"head": "Local compilation", "relation": "involves", "tail": "Codebase"}]

Example 2:
...

Now extract from the following:
Ticket Text: [New text here]
Text Category: [New category label]
Output:

Fig. 6: Illustration of few-shot prompting designs.

the LLM in automatically extracting key entities $E^{(k)} = \{e_1^{(k)}, e_2^{(k)}, \ldots, e_N^{(k)}\}$ from the incoming ticket. The extracted entities serve as initial query points within the pre-constructed KG and retrieved via CQL. For each entity, the retrieval process explores directly connected entities as well as multi-hop relations, effectively traversing the graph to identify all entities potentially related to the ticket. From this traversal, entities associated with predefined categories are extracted and aggregated, forming a set of candidate categories $C^{(k)} = \{c_1^{(k)} : num_1, c_2^{(k)} : num_2, \ldots, c_N^{(k)} : num_N\}$. The candidate categories are ranked by their frequency of association with the ticket, and the top N most relevant categories as the final candidate set. The candidate set, together with the preprocessed ticket content, is fed into TriageAgent. Additionally, background knowledge related to the ticket and detailed descriptions of each candidate category are provided to TriageAgent. Based on these inputs, TriageAgent infers the most appropriate category label from the restricted candidate set by leveraging semantic information. (2) To further improve the triage accuracy, we incorporate a reflection mechanism that prompts TriageAgent to reflect on the initially generated label, evaluate its correctness, and revise it if necessary to produce the final triage result. (3) To address the potential incompleteness of the KG, we incorporate an LLM-enhanced strategy: when TriageAgent determines that the correct label is not present in the provided candidate set, it is granted access to the full set of categories for a more comprehensive selection. This enables synergistic augmentation between LLM and the KG. To more clearly illustrate the multi-round ticket triage, we use the following equations. Here, *C* represents the set of categories along with their descriptions, *R* represents the predictive result.

$$\text{LLM(query, knowledge, } C_{\text{candidate}}) \rightarrow R_{\text{temp}} \quad (1)$$

$$\text{LLM(query, knowledge, } C_{\text{candidate}}, R_{\text{temp}}) \rightarrow R_{\text{final}} \quad (2)$$

$$\text{LLM(query, knowledge, } C_{\text{all}}) \rightarrow R_{\text{final}} \quad (3)$$

V. EVALUATION

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

- **RQ1:** How does *OncallX* perform in incident response?
- **RQ2:** How does *OncallX* perform in ticket triage?
- **RQ3:** Does each component contribute to *OncallX*?
- **RQ4:** How efficient is *OncallX* in production environment?

A. Experiment Setup

1) Dataset: To evaluate the accuracy of incident response, we constructed a test set of 70 tickets from the full dataset of 1,285 tickets collected over several months. To minimize selection bias and ensure representativeness, the subset was drawn from the first month after ticket volumes had stabilized. The selection criteria ensured both (i) balanced ticket priorities to evaluate performance across different urgency levels, with a distribution of low : medium : high = 51 : 15 : 4, and (ii) varied complexity to assess performance across tasks of differing difficulty, with a distribution of simple : medium : complex = 1 : 3 : 3. These statistics demonstrate that the sampled tickets are both high-quality and representative of the full dataset. To evaluate the accuracy of ticket triage, we collect a total of 8,662 tickets from the production environment of *B*. Among these, 8,517 tickets from Q1 2023 to Q4 2024 are used for knowledge graph construction and model training, whereas the remaining 145 tickets are utilized to evaluate triage performance.

2) Implementation Details: We conduct all the experiments with one NVIDIA A6000 GPU. We use gpt-3.5-turbo-16k and doubao-1.5-pro-32k [31] as the backbone models. Notably, gpt-3.5-turbo-16k provides performance comparable to GPT-4 at a significantly lower price [32].

3) Evaluation Metrics: In order to assess the performance of *OncallX* in incident response, we conduct three types of evaluations [16], [33]: (1) Human evaluation, (2) GPT4 evaluation and (3) Pass Rate.

- **Human Evaluation.** For this evaluation, we present both our solution and the baseline solution side-by-side to human participants, without revealing their identities. Participants are then asked to evaluate which solution is more effective according to two objective criteria: Accuracy and Readability. The codebook employed in our human evaluation is shown in Figure 7.
- **GPT4 Evaluation.** We employ GPT-4 to assess the performance of Model 1 (ours) and Model 2 (baseline) on various tasks by prompting it to score and determine the superior solution for each task.
- **Pass Rate.** We directly utilize GPT-4 to assess the effectiveness of the solutions proposed by different approachs in resolving the incident. The pass rate is calculated using the formula:

$$\text{Pass Rate} = \frac{\text{\# (Solved)}}{\text{\# (Solved)} + \text{\# (Unsolved)}} \quad (4)$$

TABLE I: Performance comparison of incident response

Model	Method	Evaluator	Baseline Wins (%)	*OncallX* Wins (%)	Pass Rate (%)	Avg.Tokens
GPT-3.5	Direct	GPT-4	35.48	64.52	72.46	0.30K
		Human	19.35	80.65	-	-
	ReAct	GPT-4	39.13	60.87	71.01	9.00K
		Human	26.09	73.91	-	-
	OncallX	GPT-4	-	-	78.26	12.51K
Doubao	Direct	GPT-4	30.65	69.35	73.91	0.92K
		Human	16.13	83.87	-	-
	ReAct	GPT-4	26.47	73.53	69.57	10.57K
		Human	19.70	80.30	-	-
	OncallX	GPT-4	-	-	75.36	9.75K

In this evaluation task, you will be given pairs of solutions to certain problems. Your task is to rate the quality of the solutions on three levels based on the following two fundamental and objective dimensions:
1. Accuracy: Whether the response is correct and effectively addresses the user's query.
2. Readability: Whether the response is concise, clear, and easy to understand, as engineers in real-world production environments prefer solutions that are straightforward and actionable.

Now, please read the following evaluation criteria and rate the given problem–solution pairs accordingly.

Evaluation Criteria
1. Accuracy:
- 3 points: The response is completely correct, fully addresses the user's query, and effectively resolves the user's needs.
- 2 points: The response is mostly correct and partially addresses the user's query, but contains minor omissions or ambiguities.
- 1 point: The response is incorrect or deviates from the user's query, failing to provide an effective solution.

2. Readability:
- 3 points: The response is concise and clear, logically structured, and easy to understand and apply.
- 2 points: The response is generally clear but somewhat redundant or not fully fluent, requiring additional effort to comprehend.
- 1 point: The response is lengthy, obscure, or poorly structured, making it difficult for engineers to quickly understand and use.

Fig. 7: The codebook for human evaluation.

Following existing methods [2], [32], we adopt the widely-used accuracy metric to evaluate the effectiveness of our ticket triage method. This metric means that a prediction is considered correct if the actual responsible team appears within the top N teams ranked by the classifier's predicted probabilities. We report two metrics, ACC@1 and ACC@3, which are calculated as follows.

$$\text{ACC@N} = \frac{\text{Sum(Correct Team in Top N Teams)}}{\text{Test Size}} \quad (5)$$

4) Baseline Approaches: To evaluate the incident response of *OncallX*, we adopt the following evaluated methods. (1) *GPT-3.5* model that does not utilize the techniques, which receives a user query and outputs the solution; (2) *Doubao* model evaluated similarly without applying any additional techniques; (3) *GPT-3.5-ReAct*. GPT-3.5 model with the ReAct [34] framework, utilizing various external tools during reasoning; (4) *Doubao-ReAct*. Doubao model with the ReAct framework, also leveraging tools to support the inference process.

To evaluate ticket triage performance of *OncallX*, we adopt the following methods.

- **DeepCT [2]:** DeepCT proposed a gated recurrent unit (GRU) model with attention-based mask strategy and a revised loss function, enabling incremental learning from discussions and updating incident triage results.
- **DeepTriage [3]:** DeepTriage ensembles several sub-models, 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 [32]:** COMET filters non-critical logs, extracts keywords using LLM enhanced with domain knowledge, and uses a fine-tuned FastText model to generate embeddings for precise triage via similarity matching.

B. RQ1: The Performance of OncallX in Incident Response

The results presented in Table I clearly indicate that *OncallX* significantly outperforms other baselines across all evaluation metrics, highlighting its substantial performance advantages. Specifically, *OncallX* (GPT-3.5) achieves a pass rate of 78.26%, representing a 5.8% improvement over GPT-3.5-Direct and a 7.25% improvement over GPT-3.5-ReAct. While *OncallX* requires slightly more tokens due to its comprehensive reasoning for complex on-call tasks, this increase is marginal compared to the substantial gains in performance. This superior performance can be attributed to the following three key factors.

Firstly, *OncallX* leverages domain knowledge to help LLMs better understand specialized concepts, and uses multi-turn dialogue to progressively clarify the user's true intent. Experimental results show that when the query is incomplete, the baselines are unable to provide a definitive answer and instead returns a message such as, "Sorry, the information you provided is insufficient to determine the specific cause of the issue." Moreover, vague queries can mislead *GPT-3.5-ReAct* and *Doubao-ReAct* into incorrect reasoning paths, resulting in wrong answers.

Secondly, *OncallX* enables evidence-based reasoning through collaboration among expert agents equipped with domain tools. For instance, it can leverage the "System Software Center Data Steward" tool to accurately diagnose the cause of system configuration errors. In contrast, the baseline often fail to identify root causes, offering only generic and impractical troubleshooting suggestions. These responses are typically verbose and fall short of users' expectations for conciseness.

Thirdly, *OncallX* exhibits capabilities in tree-search-based planning and reflective reasoning. *OncallX* flexibly plans multiple solution paths using tree-search-based planning and performs dynamic reflection using contextual information. When a particular path fails or yields no effective output, the system can automatically backtrack and explore alternative directions, ensuring a robust and efficient reasoning process.

C. RQ2: The Performance of OncallX in Ticket Triage

TABLE II: Performance comparison of triage methods.

Method	ACC@1	ACC@3
MART	0.372	0.710
LGBM	0.359	0.655
InvertedIndex	0.186	0.331
SI	0.117	0.269
DNN	0.048	0.379
DeepTriage	0.359	0.641
DeepCT	0.207	0.455
COMET	0.262	0.455
Ours	**0.652**	**0.774**

As shown in Table II, TriageAgent consistently outperforms all baseline approaches in ticket triage. Specifically, TriageAgent outperforms the SOTA methods by 28.0% in ACC@1 and by 6.4% in ACC@3. Although DeepTriage exhibits suboptimal performance, its effectiveness is constrained by the limitations of its sub-models. These traditional machine learning methods use simple feature extraction techniques, which struggle to model complex contextual semantics. The poor performance of DeepCT can be attributed to its reliance on attention mechanisms to reduce noise by assessing the correlation between discussions and title. However, the high noise level within engineers' discussions leads to distorted correlation judgments, making it difficult for the model to accurately identify key information. As a result, noisy data may receive disproportionately high weights, weakening the model's discriminative ability and ultimately impairing its overall triage performance. While COMET effectively leverages the language understanding capabilities of LLMs, unlike logs, engineers' discussions are highly diverse, even incidents of the same category can differ significantly in expression, making keyword-based similarity unreliable.

D. RQ3: Ablation Study

To evaluate the contribution of each component in *OncallX*, we conducted ablation studies focusing on its performance in incident response and ticket triage.

1) Incident Response: To evaluate the contributions of each component to incident response, we focus on the user intent enhancement module and the multi-agent & tree-planner. (1) **User Intent Enhancement Module.** As shown in Table III, removing this module more than doubles token consumption and reduces the pass rate significantly from 75.36% to 65.22%, highlighting its essential role. By leveraging multi-turn dialogue and domain knowledge, this module clarifies vague or incomplete queries and restructures them into well-defined inputs. This improves the LLM's understanding of user intent, facilitates more accurate expert agents selection, and ultimately enhances response accuracy. (2) **Multi-Agent & Tree-Planner.** The results show that removing this module leads to a 13.04% performance drop, highlighting the importance of well-defined agent roles and effective coordination. The Multi-Agent architecture supports task decomposition and specialization, while the Tree-Planner allows the LLM to explore diverse solutions paths and enhance response efficiency through strategic pruning.

TABLE III: Component effectiveness in *OncallX* response.

Method	Pass Rate (%)	Avg.Tokens
OncallX	75.36	9.75K
- User Intent Enhancement Module	65.22	19.26K
- Multi-Agent & Tree-Planner	62.32	10.56K

2) Ticket Triage: Table IV shows that *OncallX* outperforms all variants in ticket triage. Removing the noise reduction module leads to a 7.3% drop in ACC@1, as excessive irrelevant information can cause LLMs hallucinations. Without the KG, ACC@1 drops by 20.9% due to the challenge LLMs face in reasoning over a large number of categories. Disabling multi-round triage causes an 8.9% drop, highlighting the role of reflection mechanism in correcting LLMs' inference errors. Moreover, LLMs can compensate for incomplete knowledge coverage, enabling effective synergy between LLMs and KG.

TABLE IV: Component effectiveness in *OncallX* triage.

Method	ACC@1	ACC@3
TriageAgent	0.652	0.774
- Noise Reduction	0.579	0.729
- KG	0.443	0.752
- Multi-Round Triage	0.563	0.714

E. RQ4: Efficiency of OncallX

Previously, resolving a ticket required an average of 0.58 person-days, equivalent to approximately 4.6 hours of work by an OCE. The process typically involved troubleshooting and developing mitigation plan. Additionally, the average time to triage a ticket was around 200 seconds. This prolonged duration of the on-call process was primarily due to two factors: (1) a heavy workload that delayed timely responses from OCEs, and (2) manual troubleshooting and ticket triage

relied heavily on individual expertise and often led to slow resolution. With *OncallX* deployed, the average response time per incident has been reduced to just 21 seconds and ticket triage time dropped to 4 seconds, leading to a significant reduction in both labor and time costs. Intuitively, *OncallX* reduced the response time by approximately 789 times and increased the ticket triage speed by 50 times, dramatically improving operational efficiency.

F. Case Study

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

As shown in Figure 8, the user query first undergoes preprocessing through the User Intent Enhancement Module. In this stage, domain term are extracted from the user's query and explained via retrieval from domain knowledge bases, thereby enhancing the LLMs' understanding of on-call issues. The enriched query, which includes both the original user query and the retrieved domain knowledge, is then passed to the ClarifyAgent. This agent interacts directly with the user by asking targeted clarification questions to resolve ambiguities in the initial query. The goal is to ensure that the final query is clear, complete, and actionable, making it easier for the LLMs to grasp task ideas.

Once the query is clarified, it is submitted to the OCEAgent, which serves as the central controller responsible for selecting and orchestrating collaboration among multiple expert agents. Each expert agent is equipped with specialized tools and domain knowledge tailored to handling specific aspects of the problem. The OCEAgent dynamically manages this collaboration by distributing subtasks, aggregating intermediate results, and synthesizing a final response.

If the expert agents collectively generate an accurate and reliable response, the incident response process automatically terminates with the proposed solution. However, if the response is assessed by the user as inaccurate or inadequate, the query along with all relevant contextual information is forwarded to TriageAgent, which assigns the issue to the appropriate team for manual intervention and resolution.

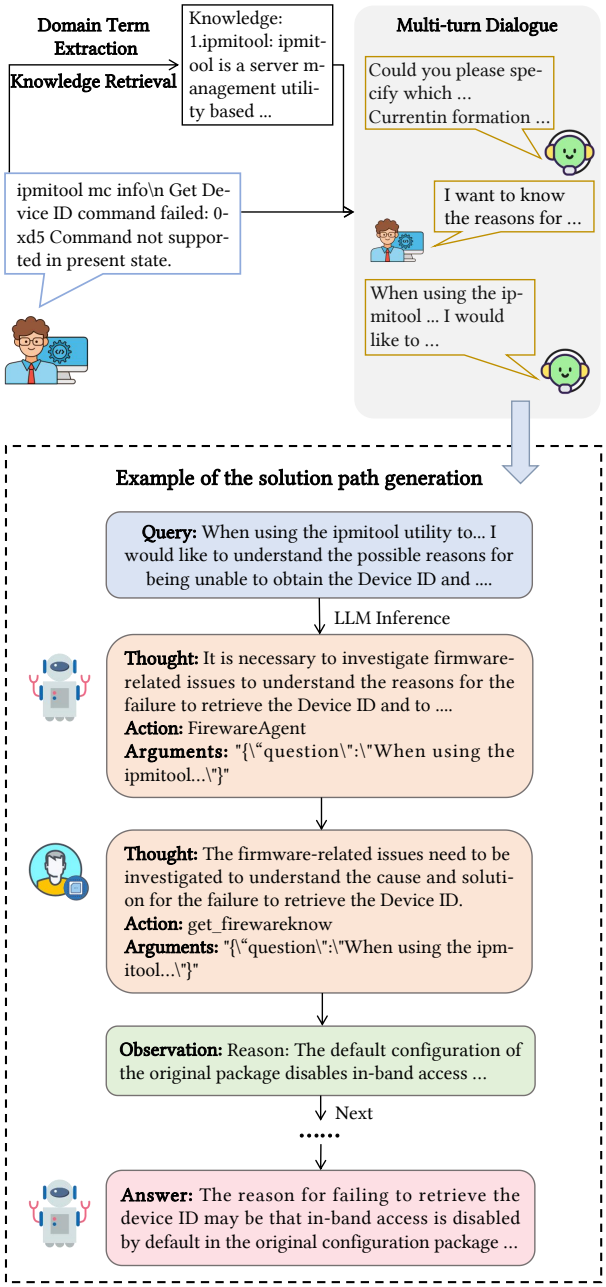


Fig. 8: The workflow of an example.

VI. DISCUSSION

A. Limitations

Our study explores the potential of multi-agent collaboration in on-call scenarios and identifies three major limitations. (1) Overestimated capabilities of autonomous agents. While they can significantly improve response efficiency and reduce human effort, their execution accuracy remains highly dependent on the quality and reliability of external knowledge bases and tool integrations. Therefore, future work should focus on equipping agents with more specialized and diverse capabilities, particularly in terms of domain tools and knowledge, to enhance their adaptability and effectiveness in complex on-call scenarios. (2) Performance limitations and resource overhead remain key practical challenges in multi-agent systems. The complexity of coordination and frequent inter-agent communication often introduce additional latency and computational burden. To mitigate these issues, future research should explore more efficient orchestration strategies aimed at reducing interaction overhead and improving overall system performance. (3) Limitations in contextual reasoning present another bottleneck. As the number of interaction rounds increases, LLMs' capacity to comprehend and retain contextual information rapidly degrades, hindering its ability to accurately track task objectives and follow historical instructions, ultimately compromising the success rate of prob-

lem resolution. This highlights an urgent need for effective conversation history compression mechanism that can distill essential information from lengthy interaction chains, thereby reducing LLMs' reasoning burden and enhancing its long-range understanding.

B. Threats to Validity

1) Internal Threats: OncallX relies on LLMs, which may cause hallucinations and introduce incorrect actions or triage. To mitigate this issue, we carefully designed the prompt templates used by *OncallX* and incorporated a reflection mechanism (detailed in Section IV-C and Section IV-D). Our experimental results indicate that this strategy is effective in reducing this threat.

2) External Threats: The threat to external validity lies in the extent to which our experimental results can be generalized. We evaluated *OncallX* only on on-call issues from the STE team in *B*'s real-world production environment. Nevertheless, we believe that the method proposed by *OncallX* possesses strong transferability and can be easily applied to on-call scenarios in other organizations. The framework is plug-and-play and does not require any module training. However, adjustments may be necessary based on the specific requirements of different application contexts. In future work, we plan to further validate and enhance the generality and adaptability of the system in more diverse operational environments.

VII. RELATED WORK

A. Ticket Triage

Ticket triage methods aim to accurately triage ticket to the appropriate team by extracting both textual and non-textual features to build various classifier models. For instance, [35] enhances classifier accuracy by ensembling multiple models such as Naive Bayes, SVM, KNN, and Decision Tree. CNN Triager [36] explored Convolutional Neural Network to further improve the handling of textual information. DeepCT [2] leverages a GRU model to capture temporal relationships in discussions and applies attention mechanisms to reduce noise in the triage process. Moreover, [7] employs deep learning to improve the accuracy and efficiency of incident triage. DeepTriage [3] combines multiple machine learning models for automated incident triage. COMET [32] utilizes LLMs to extract domain keywords and calculates embedding-based similarity between new and historical incidents for effective triage. In contrast, we adopt a fine-tuning-free approach that leverages KG to constrain the category space, thereby enhancing the effectiveness of ticket triage by LLMs.

B. LLM Agent

LLMs have been widely adopted as central controllers for AI agents, enabling them to accomplish given goals. For instance, Auto-GPT [37] leverages LLMs as an AI agent that utilizes a suite of tools to complete given tasks. To enhance the problem-solving capabilities of LLMs, recent studies have increasingly explored coordinating multiple LLM agents for collective intelligence [13], [15], [16], [38]–[41]. For example, BabyAGI [38] is an AI-powered task management system composed of several LLM-driven agents. These include agents responsible for creating new tasks based on previous objective and result, prioritizing the task lists, and completing tasks. Camel [16], a communicative agent framework, demonstrates how role playing can be used to enable chat agents to collaborate toward shared objectives. Moreover, traditional multi-agent systems often rely on handcrafted or user-specified role agents and typically lack support for automatic agent generation, thereby constraining their scalability and adaptability. To address this limitation, AgentVerse [39] and AutoAgent [41] introduce strategies for the automatic generation of an unlimited number of agents. Inspired by the demonstrated effectiveness of LLM-based multi-agent collaboration in various tasks, *OncallX* presents a LLM-powered automated on-call system in the multi-agent paradigm.

C. LLMs in Incident Management

In the realm of cloud system incident management, LLMs have been widely applied to improve various aspects of incident handling [4], [32], [42]–[48]. RCAgent [43] utilizes a tool-augmented multi-agent architecture to perform root cause analysis of cloud incidents, thereby improving the effectiveness of incident resolution. NetAssistant [44] is a dialogue-based network diagnostic system that leverages natural language processing to understand user queries and conducts diagnostics based on predefined workflows informed by the expertise of network engineers. To the best of our knowledge, *OncallX* is the first attempt to tackle the on-call problem through a multi-agent collaborative framework.

VIII. CONCLUSION

To effectively alleviate the burden of manual on-call operations, we propose *OncallX*, a system that leverages LLMs for end-to-end on-call automation. *OncallX* enhances LLMs' understanding of on-call problems through external knowledge bases and employs a multi-turn dialogue mechanism to guide users in refining their problem descriptions, thereby improving LLMs' intent comprehension. We design role-playing agents and introduce a tree-search-based multi-agent orchestration strategy to coordinate agent collaboration efficiently and improve response performance. For unresolved tickets, *OncallX* includes a dedicated ticket classifier that automatically assigns tasks to the appropriate expert teams, completing the full lifecycle of on-call handling. Extensive experiments on a real-world ticket dataset demonstrate that *OncallX* significantly outperforms state-of-the-art methods. Moreover, *OncallX* has been deployed in production environment for two months, significantly improving on-call efficiency, with an average incident response time of 21 seconds and ticket triage time of 4 seconds.

IX. ACKNOWLEDGEMENT

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

REFERENCES

[1] J. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software analytics for incident management of online services: An experience report," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, E. Denney, T. Bultan, and A. Zeller, Eds. IEEE, 2013, pp. 475–485.

[2] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 364–375.

[3] P. Pham, V. Jain, L. Dauterman, J. Ormont, and N. Jain, "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*, R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, Eds. ACM, 2020, pp. 3281–3289.

[4] Z. Yu, M. Ma, C. Zhang, S. Qin, Y. Kang, C. Bansal, S. Rajmohan, Y. Dang, C. Pei, D. Pei, Q. Lin, and D. Zhang, "Monitorassistant: Simplifying cloud service monitoring via large language models," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, M. d'Amorim, Ed. ACM, 2024, pp. 38–49.

[5] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, Y. Dang, F. Gao, P. Zhao, B. Qiao, Q. Lin, D. Zhang, and M. R. Lyu, "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*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 1487–1497.

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

[7] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "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*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 111–120.

[8] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023.

[9] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *CoRR*, vol. abs/2302.13971, 2023.

[10] T. Inaba, H. Kiyomaru, F. Cheng, and S. Kurohashi, "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*, A. Rogers, J. L. Boyd-Graber, and N. Okazaki, Eds. Association for Computational Linguistics, 2023, pp. 1522–1532.

[11] C. Qin, A. Zhang, Z. Zhang, J. Chen, M. Yasunaga, and D. Yang, "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*, H. Bouamor, J. Pino, and K. Bali, Eds. Association for Computational Linguistics, 2023, pp. 1339–1384.

[12] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller, "Augmenting large language models with chemistry tools," *Nat. Mac. Intell.*, vol. 6, no. 5, pp. 525–535, 2024.

[13] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "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*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024, pp. 15 174–15 186.

[14] R. Liu, R. Yang, C. Jia, G. Zhang, D. Yang, and S. Vosoughi, "Training socially aligned language models on simulated social interactions," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[15] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "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, 2024.

[16] G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "CAMEL: communicative agents for "mind" exploration of large scale language model society," *CoRR*, vol. abs/2303.17760, 2023.

[17] C. M. C. Nascimento and A. S. Pimentel, "Do large language models understand chemistry? A conversation with chatgpt," *J. Chem. Inf. Model.*, vol. 63, no. 6, pp. 1649–1655, 2023.

[18] P. Lee, S. Bubeck, and J. Petro, "Benefits, limits, and risks of gpt-4 as an ai chatbot for medicine," *New England Journal of Medicine*, vol. 388, no. 13, pp. 1233–1239, 2023.

[19] Y. Guo, Z. Xu, and Y. Yang, "Is chatgpt a financial expert? evaluating language models on financial natural language processing," in *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Association for Computational Linguistics, 2023, pp. 815–821.

[20] T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, S. Shi, and Z. Tu, "Encouraging divergent thinking in large language models through multi-agent debate," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, Y. Al-Onaizan, M. Bansal, and Y. Chen, Eds. Association for Computational Linguistics, 2024, pp. 17 889–17 904.

[21] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[22] C. Chan, W. Chen, Y. Su, J. Yu, W. Xue, S. Zhang, J. Fu, and Z. Liu, "Chateval: Towards better llm-based evaluators through multi-agent debate," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[23] R. Hao, L. Hu, W. Qi, Q. Wu, Y. Zhang, and L. Nie, "Chatllm network: More brains, more intelligence," *AI Open*, vol. 6, pp. 45–52, 2025.

[24] Z. Wang, S. Mao, W. Wu, T. Ge, F. Wei, and H. Ji, "Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration," 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*, K. Duh, H. Gómez-Adorno, and S. Bethard, Eds. Association for Computational Linguistics, 2024, pp. 257–279.

[25] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, L. Wang, A. T. Luu, W. Bi, F. Shi, and S. Shi, "Siren's song in the AI ocean: A survey on hallucination in large language models," *CoRR*, vol. abs/2309.01219, 2023.

[26] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 248:1–248:38, 2023.

[27] X. Sun, X. Li, J. Li, F. Wu, S. Guo, T. Zhang, and G. Wang, "Text classification via large language models," in *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Association for Computational Linguistics, 2023, pp. 8990–9005.

[28] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, S. Follmer, J. Han, J. Steimle, and N. H. Riche, Eds. ACM, 2023, pp. 2:1–2:22.

[29] R. Yang, L. Song, Y. Li, S. Zhao, Y. Ge, X. Li, and Y. Shan, "Gpt4tools: Teaching large language model to use tools via self-instruction," 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*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023.

[30] D. Fernandes and J. Bernardino, "Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb," in *Proceedings of the 7th International Conference on Data Science, Technology and Applications, DATA 2018, Porto, Portugal, July 26-28, 2018*, J. Bernardino and C. Quix, Eds. SciTePress, 2018, pp. 373–380.

[31] Doubao, https://seed.bytedance.com/en/special/doubao_1_5_pro.

[32] Z. Wang, J. Li, M. Ma, Z. Li, Y. Kang, C. Zhang, C. Bansal, M. Chintalapati, S. Rajmohan, Q. Lin, D. Zhang, C. Pei, and G. Xie, "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, 2024, pp. 523–534.

[33] Y. Du, F. Wei, and H. Zhang, "Anytool: Self-reflective, hierarchical agents for large-scale API calls," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[34] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[35] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empir. Softw. Eng.*, vol. 21, no. 4, pp. 1533–1578, 2016.

[36] S. Lee, M. Heo, C. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, E. Bodden, W. Schäfer, A. van Deursen, and A. Zisman, Eds. ACM, 2017, pp. 926–931.

[37] Significant Gravitas, "AutoGPT." [Online]. Available: https://github.com/Significant-Gravitas/AutoGPT

[38] Y Nakajima, "BabyAGI." [Online]. Available: https://github.com/yoheinakajima/babyagi

[39] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C. Qian, C. Chan, Y. Qin, Y. Lu, R. Xie, Z. Liu, M. Sun, and J. Zhou, "Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents," *CoRR*, vol. abs/2308.10848, 2023.

[40] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, "Autogen: Enabling next-gen LLM applications via multi-agent conversation framework," *CoRR*, vol. abs/2308.08155, 2023.

[41] G. Chen, S. Dong, Y. Shu, G. Zhang, J. Sesay, B. Karlsson, J. Fu, and Y. Shi, "Autoagents: A framework for automatic agent generation," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*. ijcai.org, 2024, pp. 22–30.

[42] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu, "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 2024, pp. 674–688.

[43] Z. Wang, Z. Liu, Y. Zhang, A. Zhong, J. Wang, F. Yin, L. Fan, L. Wu, and Q. Wen, "Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM 2024, Boise, ID, USA, October 21-25, 2024*, E. Serra and F. Spezzano, Eds. ACM, 2024, pp. 4966–4974.

[44] H. Wang, A. Abhashkumar, C. Lin, T. Zhang, X. Gu, N. Ma, C. Wu, S. Liu, W. Zhou, Y. Dong, W. Jiang, and Y. Wang, "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*, L. Vanbever and I. Zhang, Eds. USENIX Association, 2024.

[45] X. Zhang, S. Ghosh, C. Bansal, R. Wang, M. Ma, Y. Kang, and S. Rajmohan, "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*, M. d'Amorim, Ed. ACM, 2024, pp. 266–277.

[46] P. Jin, S. Zhang, M. Ma, H. Li, Y. Kang, L. Li, Y. Liu, B. Qiao, C. Zhang, P. Zhao, S. He, F. Sarro, Y. Dang, S. Rajmohan, Q. Lin, and D. Zhang, "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*, S. Chandra, K. Blincoe, and P. Tonella, Eds. ACM, 2023, pp. 1657–1668.

[47] Y. Jiang, C. Zhang, S. He, Z. Yang, M. Ma, S. Qin, Y. Kang, Y. Dang, S. Rajmohan, Q. Lin, and D. Zhang, "Xpert: Empowering incident management with query recommendations via large language models," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 92:1–92:13.

[48] J. Liu, C. Zhang, J. Qian, M. Ma, S. Qin, C. Bansal, Q. Lin, S. Rajmohan, and D. Zhang, "Large language models can deliver accurate and interpretable time series anomaly detection," *CoRR*, vol. abs/2405.15370, 2024.