iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation

Junjie Huang*, Yuedong Zhong†, Guangba Yu*, Zhihan Jiang*,
Minzhi Yan‡, Wenfei Luan‡, Tianyu Yang‡, Rui Ren‡, Michael R. Lyu*
*The Chinese University of Hong Kong, China, {jjhuang23, guangbayu, zhjiang22, lyu}@cse.cuhk.edu.hk
†Sun Yat-sen University, China, ‡Huawei Cloud Computing Technology Co., Ltd, China, yanminzhi@huawei.com

***Abstract*—Managing complex cloud services requires standard operational documentation, but its sheer volume often hinders cloud engineers from efficient knowledge acquisition. Retrieval-Augmented Generation (RAG) can streamline this process by retrieving relevant knowledge and generating concise, referenced answers. However, deploying a reliable RAG-based chatbot for cloud operation remains a challenge. In this experience paper, we analyze the development and deployment of RAG-based chatbots for operational question answering (OpsQA) at a large-scale cloud vendor. Through an empirical study of 2,000 real-world queries across three operational teams, we identify five unique OpsQA intent types (*e.g.*, symptom analysis and terminology explanation) and their corresponding requirements for a satisfactory answer, which differ from general software engineering queries. Our analysis further uncovers six root causes leading to chatbot failures—over half stem from query issues (*i.e.*, incompleteness, out-of-scope, or invalid queries), while others are from retrieval or generation issues. To address these issues, we propose iKnow, an intent-guided RAG-based chatbot that integrates intent detection, query rewriting tailored to each intent, and missing knowledge detection to enhance answer quality. In internal evaluations, iKnow improves average answer accuracy from 65.8% to 81.3% with only a modest increase in latency. iKnow has been deployed for six months at CloudA, supporting thousands of cloud engineers in daily operations. We discuss lessons learned from real-world deployment, providing valuable insights for future research and practical implementations in similar domains.**

***Index Terms*—software operation, chatbot, retrieval augmented retrieval, intent detection**

I. Introduction

Modern cloud platforms, such as Azure and AWS, provide over 200 distinct products, each with comprehensive documentation that elaborates on their features, usage procedures, and troubleshooting guides [1], [2]. However, software engineers who manage and operate these services encounter considerable challenges in accessing relevant information. For example, when a GPU *Xid 74 Error* occurs during training large language models (LLMs), engineers must sift through extensive pages of documents to determine that this error is related to an NVLink Error and follow subsequent mitigation steps.

As shown in Fig. 1, existing keyword-based retrieval systems [3], [4] offer limited capabilities, often returning extensive relevant documents that require manual filtering and interpretation, thus leading to reduced productivity and prolonged resolution times [5]–[7]. In addition, the inability to quickly

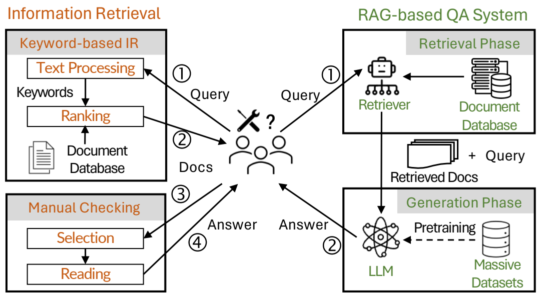

Fig. 1. Comparisons of information seeking methods.

access actionable information can result in suboptimal solutions or delays in critical decision-making, adversely affecting service quality and customer satisfaction. Recent studies have explored LLMs to answer cloud operation queries [8]–[10]. However, these models often suffer from hallucination, *i.e.*, generating plausible but incorrect responses, posing significant risks in high-stakes operational environments [11]. Additionally, much of the documentation and domain-specific knowledge in cloud operations is proprietary and confidential, often absent from the training data of general-purpose LLMs [9].

To overcome these limitations of "out-of-the-box" LLMs, Retrieval-Augmented Generation (RAG) is proposed for knowledge-intensive tasks [12]. As shown in Fig. 1, a RAG framework combines a *retrieval* component, which searches a designated document database for relevant knowledge, with a *generation* component, which synthesizes responses based on both retrieved documents and query context [13]. By incorporating relevant knowledge, a RAG-based chatbot can generate summarized answers with source reference, making them well-suited for cloud operations requiring verifiable information.

Despite RAG's potential, developing a reliable RAG-based chatbot for cloud operations presents significant challenges. In this experience paper, we analyze the development of RAG-based chatbots for operational question answering (OpsQA) at a large-scale cloud vendor, CloudA (§ III). Plenty of teams at CloudA maintain diverse operational documents, such as product manuals, FAQs, incident tickets, and failure libraries. We develop three RAG-based chatbots based on open-source solutions for three operational teams at CloudA and deploy them for two months. Through an empirical analysis of 2,000 data points (including user queries, retrieved documents, and LLM-generated responses), we identify five unique intent types in OpsQA, such as symptom analysis and terminology

*Zhihan Jiang is the corresponding author.

explanation. These types differ significantly from general software queries, primarily due to a greater focus on system analysis [14], [15]. We also outline the requirements to satisfy the information need in each intent. However, existing RAG-based chatbots exhibits limited performance in real-world OpsQA, especially those with particular intents (§ III-C).

To understand the weaknesses of the RAG-based chatbots based on open-source solutions, we investigate the 683 failed cases to investigate root causes (§ III-D). From the empirical analysis, we identify six root causes and their distributions spanning the lifecycle of a RAG pipeline, including query, retrieval, and generation. We find that more than half of failed queries are imperfect (*i.e.*, incomplete (32%), out-of-scope (10%), and invalid (9%)), requiring query enhancement for more precise retrieval and focused response. Lacking knowledge is another important cause (the second highest of 27%), which causes LLMs to respond more often with its intrinsic knowledge and have more chances of hallucination. Furthermore, we find that root causes have a correlation to query intents. For example, 58.8% failed queries for terminology explanation are due to incomplete query, suggesting the potential of intent-specific query enhancement.

To address the issues, we introduce iKnow, an intent-guided chatbot for OpsQA. iKnow follows a general RAG framework but enriched with a series of intent-specific improvements to handle operational queries of varying intents. First, to manage queries of different intents, we propose an efficient intent detection module based on prototypical network [16], [17]. The classified queries are then routed to a query rewriting module for intent-specific rewriting before retrieval. Guided by the detected intents, this module leverages an LLM with tailored prompts to reformulate the imperfect queries, ensuring they were semantically rich and properly structured, thereby improving retrieval accuracy and generation focus. Lastly, we introduce a missing knowledge detection module after retrieval, which applies LLM-based categorization on query-context sufficiency to detect the absence of relevant knowledge. With improved queries and relevant retrieved context, LLMs can generate more accurate responses to help operators.

We evaluate iKnow on three internal datasets with 2,000 queries. The results reveal that iKnow achieved a significant improvement in overall accuracy, from original 65.8% to 81.3%. Specifically, iKnow improves the response comprehensiveness to cover more aspects when explaining terminology (with 78% win-rate in coverage), while also provides more concrete details (such as commands and API examples) in answering queries that seeks summary (with 77% win-rate in specificity). We also conduct ablation study to evaluate the effectiveness of proposed components. Moreover, our intent-specific enhancements do not cause a significant increase in delay, achieving an end-to-end latency of 22.5 seconds and consuming 19.1% of the total time, underscoring its practical effectiveness. Till now, iKnow has been deployed at CloudA for six months, providing QA services to a range of operational teams with customized documents, including customer service, on-call engineers, DevOps engineers. Finally, we share our lessons learned throughout the deployment experience and discuss future directions.

Contribution. We summarize our contribution as follows.

- We present a comprehensive study of cloud operation question answering with RAG-based chatbots. We qualitatively and quantitatively characterize five operational query intents and six root causes, suggesting the room for improvement.
- We developed iKnow, an intent-guided RAG-based chatbot that addresses prevalent reliability challenges through intent detection, query rewriting, and missing knowledge detection. Deployed at CloudA, iKnow supports thousands of cloud operators in their daily tasks.
- We share our experiences and lessons learned from building and deploying a reliable chatbot for cloud operations, providing valuable insights for future research and practical implementations in similar domains.

II. Background

A. Cloud Operation and Chatbot

To improve service reliability and customer satisfaction, modern cloud companies often organize a large group of operational engineers to perform operational tasks and manage their cloud-based software systems [18]. These tasks, such as system maintenance, testing, and troubleshooting, are complex and diverse, necessitating high expertise to ensure precise operations [19], [20]. To ensure the safety and reliability of the operation works, engineers often need to follow documents to reduce misoperation [21]–[23]. These documents, including product manuals, troubleshooting guides, and historical issue tickets, are valuable resources for engineers providing knowledge of the service system and guidance of operation steps. However, manually retrieving knowledge from documents is time-consuming and prone to omission of important information, as engineers need to frequently switch among pages and continuously adjust queries to search numerous documents distributed in different platforms.

Recently, question answering systems (*aka*, chatbots), which accept a user's query and deliver a concise answer in natural language, are widely adopted to fulfill the information need and improve the efficiency of software engineers [24]–[26]. These chatbots, leveraging the text comprehension abilities of Large Language Models (LLMs) [27], have shown significant success in interpreting diverse questions and producing coherent answers and are widely adopted in cloud companies such as Azure [28] and AWS [29]. Nonetheless, relying solely on LLMs poses challenges such as the hallucination and the inability to access to timely and proprietary data [11], [12]. This is especially problematic in cloud operations, where sensitive documents, including failure logs, are retained exclusively within organizations and span multiple versions.

B. RAG-based Chatbot

To address these limitations of "out-of-the-box" LLMs, Retrieval-Augmented Generation (RAG) was proposed to improve chabot performance by delivering faithful responses tailored to documents of specific domains [12]. As depicted in

Fig. 6(a), a conventional RAG system consists of two stages: the *offline* and *online* stages. In the offline stage, a large set of raw documents are embedded into vectors and stored in vector databases for future retrieval. Specifically, documents are first segmented into fixed-length chunks (*e.g.*, 100 tokens). Chunk enrichment methods are often applied in this stage to improve retrieval accuracy, such as prefixing titles [30]. These chunks are subsequently encoded into vectors via an embedding model and stored in a vector database for efficient retrieval. In the online stage, an incoming query is first encoded into a vector with the same embedding model. This vector is then used to retrieve relevant chunks from the database using a similarity metric such as cosine similarity. In addition to similarity search, methods like reranking and term-based ranking, are frequently used together to increase the likelihood of answer coverage in top-k chunks [30], [31]. After retrieval, the query and associated chunks are composed into a prompt, which contains instructions for LLMs to answer based on the given context. Finally, an LLM processes the prompt and produces response answers, which are returned to the users.

III. Study of Preliminary RAG-based QA Systems

To satisfy the emerging information seeking need of cloud operators quickly, our team first developed and deployed three RAG-based chatbots at CloudA using open-source solutions [32], [33] . Having being deployed for two months, these systems, while effective and useful for many cases, still produces unsatisfactory or incorrect outputs as open-sourced frameworks might fail short in tackling specific issues in OpsQA. Therefore, we first conduct a thorough empirical study to qualitatively and quantitatively analyze task characteristics and answer failures in OpsQA. In the following section, we introduce our study setting and share our findings.

A. Study Design

To understand the characteristics of OpsQA and performance of RAG-based chatbots, we design three research questions (RQ) to answer through our analysis.

- **RQ1:** What questions do cloud operators frequently ask?
- **RQ2:** How well can existing RAG-based chatbot answer these questions?
- **RQ3:** Why does existing RAG-based chatbot fail?

Study System: The deployed chatbots follow the standard RAG framework introduced in § II-B. We adopt an open-source LangChain [32] framework and the FAISS [33] vector database. Specifically, we set the chunk size as 100 tokens and prepend document names and section titles to enrich semantics. The embedding model `BGE-M3` [34] is employed, while the reranker model is `bce-reranker-base_v1 [35]`, and `Qwen2.5-32B` [36] serves as the LLM. We retrieve the most similar five chunks to compose the prompt. Apart from the RAG framework, we have developed a user interface enabling users to submit queries and offer feedbacks. QA data is also stored to track system usage.

TABLE I
SUMMARY OF OPSQA DATASETS.

	# Query	Document Source	Data Type	# Words	User Group
$\mathcal{A}$	300	Product docs, group Wikis, FAQ	Text, PDF, HTML	~1.2 M	External Ops Team
$\mathcal{B}$	1,350	Product docs, group Wikis, FAQ	Text, PDF	~0.6 M	Internal Ops Team
$\mathcal{C}$	350	Incident ticket, fault library, FAQ	Text, PDF, Figure	~0.4 M	On Call Team

Study Dataset: To answer the three RQs, we manually analyze real-world QA data collected from deployed RAG-based QA systems over two months at CloudA, a prominent cloud vendors offering hundreds of services to global users. To enhance the generalizability of our findings, we developed three datasets by randomly sampling user queries and responses from three systems. These systems share the same architecture but are used by different teams, each with their customized documents. $\mathcal{A}$ contains 300 cases based on the product manuals of product A, used by an external operations team. $\mathcal{B}$ contains 1,350 cases based on the product manuals of product B, used by the internal operations teams. $\mathcal{C}$ includes 350 cases based on the incident tickets and failure logs of product C used by an On Call Engineers (OCEs) team. Each case includes a query, the top-10 retrieved documents, and the generated response. Table I shows the dataset statistics.

Study Methods: To answer the three RQs, we identified the key factors to capture for each RQ, *i.e.*, query intents, answer correctness, and root causes, respectively (as shown in Table II). To holistically study these factors, we employ a two-phase paradigm commonly used in empirical software engineering studies [22], [37], [38]. Initially, open coding [39], [40] is conducted to develop a taxonomy and annotation guidelines through iterative manual analysis, followed by consensus-based annotations for quantitative analysis at scale using the established taxonomy. The annotation was conducted using a custom annotation system, which managed task assignment and label collection due to the large number of cases and annotators involved. We elaborate our methods as follows.

TABLE II
STUDY FACTORS FOR ANALYZING OPSQA DATASETS.

RQ	Study Factor	Description
RQ1	Query Intent	What is the information needed?
RQ2	Answer Correctness	What are the requirements of answers? Does the answer satisfy the requirements?
RQ3	Root Cause	What causes the incorrect answer?

We first randomly split 2,000 cases into training, validation, and test sets in a 2:1:1 ratio. In phase one, 15 experts, including five doctoral students and ten engineers with over three years of cloud operations experience, used open coding [39], [40] to develop a taxonomy for each factor with the training set independently. Specifically, every case was coded by three different experts, resulting in three codes per case. The codes—free-form descriptions or keywords—were then discussed by all experts to develop a consensus taxonomy and annotation guidelines. Two team leaders facilitated the discussion and resolved disagreements on non-consensus codes. In phase two, 15 experts annotated the examples using the taxonomy and guideline via the annotation system. Each case was independently labeled by three experts. The labeling process began with the validation set to ensure no new categories emerged

TABLE III
INTENT TAXONOMY OF REAL-USER QUERIES IN OPSQA.

Intent	Query Manifestation	Query Examples
Symptom Analysis	Describes observable or ambiguous system behaviors and anomalies.	"No performance data for VM disk usage?" "App Orchestration error: Slave has no response."
Multi-facet Summary	Solicits a comprehensive or summarized overview of operational concepts or issues, often using high-level terms.	"Flink troubleshooting summary?" "FAQ for SystemA backup?" "Comprehensive guide to $\mathcal{S}$ production issues"
Terminology Explanation	Asks for the meaning of a technical term, usually short or abbreviated.	"ESN?" "SystemA private key?" "Operator login password policy?"
Fact Verification	Seeks specific factual details, often posed as yes/no or what questions.	"Does ServiceA support capacity statistics?" "Can roles of existing users be modified?"
Operational Guidance	Requests step-by-step instructions for an action, often as a "how-to" question or verb–noun phrase.	"Check tenant operation logs?" "Add MySQL to access whitelist?" "How to request quota for the current VDC?"

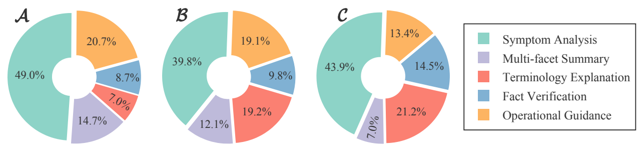

Fig. 2. The query distribution of different intents.

and to further refine the taxonomy if necessary. The testset was then annotated and the inter-annotator agreement (*i.e.*, Krippendorff's alpha [41]) was measured to ensure reliability, followed by annotation of the training set. For each case, the final label was determined by majority vote; any remaining disagreements were resolved through group discussion. After annotation, we observe a high concordance for all factors: Query Intent: 0.84, Answer Correctness: 0.86, Root Cause: 0.81. It is worth-noting that each case was not redundantly classified into multiple categories.

The annotation system ensured that each case was assigned to 3 different experts and prevented duplicate assignments. This partial coverage approach is widely used in crowdsourcing platforms like Amazon Mechanical Turk [42] for large-scale annotation. To accommodate varying time commitments among experts, a flexible annotation scheme was adopted, requiring each expert to annotate at least 100 cases per factor.

B. RQ1: What questions do cloud operators frequently ask?

1) Study Settings: To understand the information need of cloud operators, we first examine and categorize the intents of the queries they submit, following previous work on programmers' questions [14], [43]. To collect intent taxonomy and labels, we follow the study method introduced in § III-A. Table III outlines the intent categories with definitions and examples. Fig. 2 shows the intent distribution in three datasets. In the following, we highlight key findings from our analysis.

2) Analysis: **Operational queries manifest five unique intent types when operators search for answers.** These queries cover information need from two dimensions: general operational knowledge (*e.g.*, `terminology explanation` and `multi-facet summary`) and task-specific guidance (*e.g.*, `symptom analysis`). The former usually serves to enhance operator knowledge, whereas the latter provides detailed guidance to address complex issues in system performance during routine tasks. Furthermore, the intent types in OpsQA differ significantly from programmers' questions typically found on platforms (*e.g.*, Stack Overflow and GitHub), where intents related to API usage, conceptual understanding, discrepancies, and reviews are predominant [14], [15], [43].

Symptom analysis is the most common intent, occupying 40.6% on average across three operations teams, which differs from general SE questions. `Symptom analysis` dominates among the five intent categories, showing the highest frequency across all three datasets, specifically 45.4%, 39.1%, and 37.3% in $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, respectively. These queries literally describe an observable anomalous behavior, such as a failure phenomenon or an error message, reflecting operators' helplessness when facing complex real-time issues and their need for diagnostic insights from past cases. The large portion emphasizes the primary role of cloud operators in diagnosing and resolving unprecedented issues, which is also different from general programmers' questions that focus more on how-to questions [14], [15], [43], [44].

Operators across teams have similar information need, yet show moderate variations in their query preferences, highlighting the generalizability of our OpsQA intent taxonomy. As shown in Fig. 2, three datasets share the same five intent types. While `symptom analysis` is the most prevalent type in these datasets, the distributions of other intents show moderate differences. For example, queries concerning `fact verification` are more common among OCE teams (14.5% for $\mathcal{C}$) compared to the other two teams (8.7%/9.8% for $\mathcal{A}$/$\mathcal{B}$). Moreover, `terminology explanation` are posed more frequently by the internal team (19.2%/21.2% for $\mathcal{B}$/$\mathcal{C}$) than by the external team (7.0% for $\mathcal{A}$). The intent distribution suggests that our OpsQA intent taxonomy can be generalized across operational teams with diverse roles.

Summary of RQ1: There are five intent types in OpsQA. Cloud operators primarily seek information on symptom analysis (40.6%), highlighting unique characteristics in cloud operations compared to general SE questions. The intent distribution's similarity across datasets suggests the taxonomy generalizability.

C. RQ2: How well can existing RAG answer these questions?

1) Study Settings: Upon understanding the query information needs in OpsQA, our next focus is to evaluate the performance of existing RAG-based chatbots in handling these queries. As shown in RQ1, queries have distinct informational requirements; hence, responses need to be both factually accurate and aligned with specific demands. For example, responses to `symptom analysis` queries should analyze the issue thoroughly and identify its root cause. To collect intent-specific response criteria and correctness labels, we implement the study method introduced in § III-A. Table IV shows the criteria for correct responses and symptoms for failed responses of various query intents. Responses meeting the criteria are regarded as *correct*, while those displaying failure symptoms will be regarded as *incorrect*. Fig. 3 shows the answer accuracy across three datasets.

2) Analysis: **RAG-based chatbots exhibit varying capability in answering OpsQA queries, excelling at primary information needs (symptom analysis) but struggling with factual and terminology-related queries.** The

TABLE IV
INTENT-SPECIFIC ANSWER CRITERIA FOR RAG-BASED OPSQA.

Intent	Answer Requirement	Symptoms of Failed Answer
Symptom Analysis	Diagnoses the issue, identifies root cause, and provides actionable suggestions, grounded in the retrieved sources.	**Hallucination**: Fabricates causes or suggestions not in sources. **Intent-conflicting**: Misidentifies or ignores the issue. **Overly Generic**: Lacks concrete diagnosis or actionable steps. **Deficiency**: Provides misleading or incomplete analysis.
Multi-facet Summary	Aggregates and synthesizes relevant aspects to deliver a comprehensive overview, highlighting key perspectives.	**Hallucination**: Fabricates aspects not supported by sources. **Intent-conflicting**: Summarizes non-targeted concepts. **Overly Generic**: Offers vague summary without specifics. **Deficiency**: Omits key dimensions or offers misleading synthesis.
Terminology Explanation	Provides a clear and accurate definition of the targeted term with relevant details, as supported by sources.	**Hallucination**: Fabricates definitions unsupported by sources. **Intent-conflicting**: Explains unrelated concepts. **Overly Generic**: Gives vague or overly brief definitions. **Deficiency**: Gives incorrect or incomplete definitions.
Fact Verification	Explicitly confirms or refutes the queried fact with explanations and supporting evidence from the sources.	**Hallucination**: States facts not grounded in sources. **Intent-conflicting**: Fails to directly answer the question. **Overly Generic**: Provides vague explanations for the verification. **Deficiency**: Gives incorrect or incomplete explanations.
Operational Guidance	Delivers thorough, step-by-step instructions with actionable details, ensuring completeness and practical applicability.	**Hallucination**: Suggests unverified or incorrect procedures. **Intent-conflicting**: Fails to provide steps or targets wrong task. **Overly Generic**: Lacks concrete actions or operational details. **Deficiency**: Misses critical steps or details.

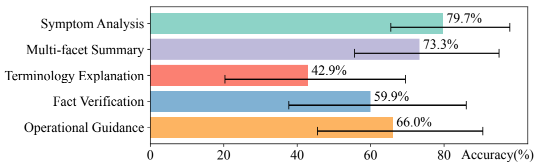

Fig. 3. Response accuracy of different intent types.

system achieves high accuracy for symptom analysis queries (79.7%), the most common intent, demonstrating its effectiveness in supporting primary operational needs. However, accuracy drops notably for queries seeking factual details, such as terminology explanations (42.9%). This gap suggests that while the chatbot effectively addresses the most common queries, it is less reliable for direct factual retrieval, which is mainly due to incomplete knowledge and queries (see RQ3 (§ III-D)). Therefore, addressing these limitations is essential to enhance the accuracy of RAG-based chatbots in OpsQA.

Answer requirements for queries of different intents are distinct, resulting in intent-specific failure symptoms. In OpsQA, each query intent reflects a unique information need from cloud operators, which in turn determines the criteria for a satisfactory response. For example, `symptom analysis` queries require answers that can accurately diagnose issues and provide actionable guidance, while `terminology explanation` queries expect precise and context-aware definitions. Due to these varying requirements, the symptoms of failed responses are also intent-specific: an unsatisfactory `symptom analysis` answer may lack diagnostic insight or actionable detail, whereas a failed `terminology explanation` may present an imprecise or off-the-topic definition. Table IV summarizes these intent-specific answer requirements and corresponding failure symptoms.

Failed responses consistently exhibit four primary types of symptoms, though their manifestations vary by intent. Across all query intents, incorrect answers from existing RAG-based chatbots tend to fall into four major symptom categories: *hallucination* (fabricating content not present in sources), *intent-conflicting* (misunderstanding or ignoring the query intent), *overly generic* (lacking specificity or actionable detail), and *deficiency* (providing incomplete or misleading information). While these types are observed consistently, their specific manifestations are shaped by the correspond-

TABLE V
ROOT CAUSES OF OPSQA FAILURES IN RAG-BASED CHATBOT

Root Cause	Definition	Example
Incomplete Query	The query lacks sufficient context or specificity, often containing only a noun/term without signal words (*e.g.*, what, why, how), making the intent ambiguous.	"ESN?"; "Snapshot Residue"; "SSL_ERROR_SYSCALL"
Out-of-Scope Query	The query is unrelated to IT operations or outside the domain covered by the system.	"What's the financial status of customerA?"
Invalid Query	The query contains nonsensical text, corrupted input, or severe misspellings, rendering it unanswerable.	"sdfa"; "Snaphot Resdieu"
Knowledge Missing	The query is relevant and well-formed, but the required information is absent from the document corpus, resulting in no retrievable evidence.	Asking about a newly released feature not yet documented in the knowledge base.
Inaccurate Retriever	The retriever fails to return all necessary documents due to model or data preparation issues (*e.g.*, improper chunking, poor embeddings, poor ranking methods).	Relevant documents exist in corpus but are not included in the top-k retrieved chunks.
Inaccurate Generation	The model generates incorrect or unsupported output despite receiving relevant context, due to decoding, prompt, or alignment issues (*e.g.*, misunderstanding terms or instructions, omitting details).	Misinterpreting "indicator" as a general term, or providing a generic summary when a step-by-step guide was requested.

ing answer requirements. For example, an overly generic response in `symptom analysis` might simply restate the problem without offering concrete diagnostic steps, whereas in `terminology explanation` it could provide a vague definition lacking technical depth. This consistent symptom taxonomy supports systematic and fine-grained evaluation of answers across diverse operational scenarios.

Summary of RQ2: RAG-based chatbots show varying effectiveness across OpsQA query intents. They excel at addressing primary symptom analysis queries (79.7%), but perform poorer on queries requiring factual or terminology details. Distinct answer requirements for each intent lead to specific failure symptoms, underscoring the need for intent-aware improvements to enhance overall answer quality.

D. RQ3: Why does existing RAG fail to answer the questions?

1) Study Setting: RQ2 reveals the inconsistent performance of RAG-based chatbots across different query intents. Next, we aim to systematically uncover the root causes of answer failures. To this end, we analyze the 683 incorrect cases identified in RQ2 and follow the study method described in § III-A to identify root causes and collect labels. While multiple causes may co-occur, for clarity, we report only the most dominant cause for each case in our analysis. Table V outlines the taxonomy with definitions and examples. While prior works explored fine-grained analysis of model-side failures in RAG pipelines (*e.g.*, insufficient training or prompt deficiency) [30], [31], [45], our focus is on practical failures in the cloud operation domain. Thus we group model-side causes into two broad categories (*i.e.*, `inaccurate retriever` and `inaccurate generation`) rather than further subdividing them. Fig. 4 and Fig. 5 present the overall and intent-specific distributions of root causes, respectively. We discuss key findings in the following analysis.

2) Analysis: **Over half of failures (51%) originate from user query issues, with incomplete query emerging as the leading cause (32%). This highlights the need for clarification to obtain more accurate answers.** Incomplete queries often lack sufficient context or explicit intent, challenging both retrieval and generation stages, as the system must infer user intent from minimal input. For example, a

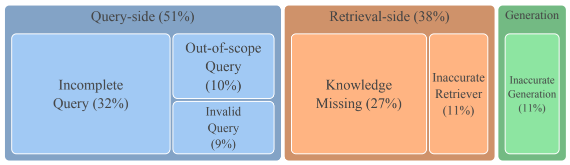

Fig. 4. The overall failure distribution.

query like "Snapshot Residue" is ambiguous, as it is unclear what information the user seeks, a definition or a solution, which can confuse the retriever and the generator to produce off-target responses. This cause is especially pronounced for terminology explanation queries, where 58.8% of failures are due to incompleteness. The prevalence of such queries reveals a gap between real-world user behavior and the well-formed queries found in many benchmark RAG datasets [44], [46], [47]. Addressing this requires both system-side improvements, such as intent clarification or query reformulation, and user education to encourage more explicit queries.

Out-of-scope (10%) and invalid queries (9%) are crucial causes that should be properly detected and communicated to users. Although imperfect responses to these irrelevant queries do not adversely affect cloud operations, processing them still results in unnecessary computational consumption, which is suboptimal for system efficiency. Early detection and user notification can help avoid this inefficiency. Notably, these root causes are most frequent in `terminology explanation` queries, suggesting that intent-specific detection methods, such as intent classification or query validation, could effectively identify and address these issues.

Knowledge missing (27%) is the second most frequent root cause, often leading to hallucinated responses that lack grounded knowledge and may adversely impact cloud operations. When required information is absent from the system's knowledge base, accurate retrieval may fail and the LLMs may generate plausible but incorrect answers [11], [48], which can mislead operators and potentially compromise operational reliability. This root cause is especially problematic for `fact verification`, `symptom analysis`, and `operational guidance` queries, which require concrete details that are often not fully documented due to the difficulty of enumerating all possible failure scenarios. To address this, the QA system should assess knowledge sufficiency and notify users when information is incomplete or unavailable, thus improving transparency and reducing erroneous actions.

Inaccurate generation (11%) accounts a smaller proportion of root causes, but still require attention and improvement. This indicates that, given correct queries and retrieved context, a 32B LLM can resolve most cloud operation questions. Nevertheless, further enhancing generation accuracy remains important, as this phase directly affects response quality and user satisfaction. Improvements may involve adopting more capable LLMs with broader operational knowledge and refining prompt instructions to better capture user intent.

Summary of RQ3: We identify six root causes in RAG-based chatbot failures for OpsQA, with incomplete queries (32%) and missing knowledge (27%) being the most common. Efforts should prioritize improving query clarity and detecting knowledge gaps in retrieval to address the failures.

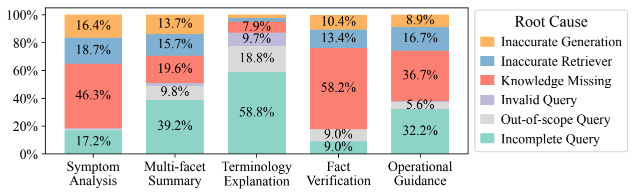

Fig. 5. The failure distribution of varying intents.

IV. iKnow: An Intent-aware OpsQA RAG System

In this section, we introduce our optimization experience on RAG-based chatbot to mitigate the above limitations. From our study, we identify the prevalence of query incompleteness and knowledge absence. Motivated by the strong correlation of these with query intents, we propose iKnow, an intent-aware chatbot to mitigate these failures to handle the diverse information needs of operational queries.

Fig. 6(b) shows the overall architecture of iKnow. Given a query, iKnow first identifies the intent and extracts metadata (*e.g.*, document version). This intent directs query rewriting by enhancing specific requirements, while the metadata informs the selection of a proper vector database considering the document's evolution in OpsQA. Once relevant chunks are retrieved by two-stage retrieval, iKnow then assesses the presence of knowledge and determine if to provide a degraded response. Once sufficient knowledge is confirmed, an enhanced prompt is utilized to produce a valid response. In the following, we introduce iKnow in detail.

A. Vector Databases

Before online QA, we first build vector databases (VecDBs) offline with operation documents. Due to rapid software evolution, multiple document versions are available for operator reference. Hence, we construct multiple VecDBs offline and select a proper VecDB for querying a specific version. For each VecDB, documents are divided into semantically meaningful chunks, balancing between including complete sentences and adhering to maximum token counts. The document title of chunks is prefixed to enrich contextual information [30]. The chunks are encoded into vectors using a BGE-m3 [34] embedding model for indexing. The vectors are then stored in a FAISS [33] VecDB to enable fast retrieval.

B. Intent Detection and Metadata Extraction

To enable intent-guided query rewriting for different requirements, we first classify the intent category of each OpsQA query. As shown in RQ1 (§III-B), OpsQA queries with different intents often show distinct linguistic patterns. For example, terminology explanation queries are typically concise and contain only a technical term, while operational guidance queries often start with "how to" or "what is the process for." These semantic cues allow semantic-based intent classification.

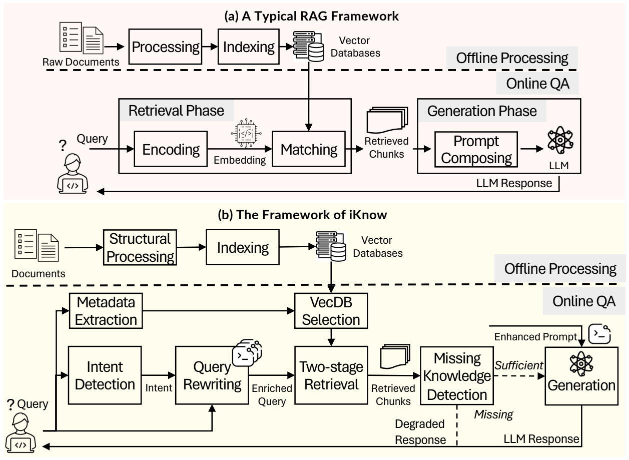

Fig. 6. The framework of iKnow compared to the existent system.

In iKnow, we use Prototypical Networks (PN) [16], [17] for intent detection, due to its efficiency and strong performance in low-resource settings. PN first constructs an intent's embedding to serve as its prototype and then finds the closest intent prototype to classify a given query embedding. Specifically, we select BGE-m3 [34] to vectorize queries, due to its excellence in sentence embedding modeling. The mean of query vectors that share the same intent in the training dataset is calculated as the prototype. During classification, the query vector is firstly computed and then assessed for cosine similarity [49] with all five intent prototypes. The prototype with the maximum similarity is assigned as the target intent category.

In this work, we extract two types of metadata, *i.e.*, application and version, to select VecDB with the appropriate documents. For efficiency, we build a term database and match the string to extract query metadata. In case no matched metadata are found, we provide the latest version and default application to guide VecDB selection.

C. Intent-guided Query Rewriting

As query issues account for a majority of failures, we propose rewriting queries [43], [44], [50] to improve query clarity. High-quality rewritten queries should be well structured and clear [47] to facilitate retrieval and QA. However, due to the variety of information needs in operational queries, simply adopting a uniform rewriting style might alter the original intent. To address this, we employ the identified intent to guide query rewriting.

Specifically, we utilize an LLM to rewrite queries by providing detailed rewritten prompts. These prompts contain rewriting instructions and three examples specific to an intent category. For example, for terminology explanation queries, the prompt guides the LLM to rephrase the terminology query into a "what" question concerning its definition and attributes, *e.g.*, "What are the definition and use scenarios of an ESN?". The result is a rewritten query with enhanced detail and focus, which is beneficial for subsequent retrieval and generation [47]. Due to space limitations, the full rewritten prompt is available in our anonymous repository [51].

D. Two-stage Evidence Retrieval

Retrieval quality is crucial for answer accuracy in RAG frameworks, as the LLM's reasoning is limited by the relevance of retrieved evidence [52], [53]. However, as shown in RQ3 (§ III-D), many OpsQA queries still face partial or imprecise retrieval, where relevant documents are not ranked highly enough to be included in the LLM's context. To address this, we adopt a two-stage retrieval method. First, a vector-based similarity search retrieves a broad set of candidate chunks (*e.g.*, top-15), just as in the typical RAG framework, which maximizes recall but may introduce irrelevant results. After that, a cross-encoder reranker [35] re-scores these candidates based on their pair-wise relevance to the query, selecting the top-k (*e.g.*, top-5) for LLM input. This approach balances the efficiency of dense retrieval with the precision of reranking, enabling fast and accurate retrieval.

Specifically in the online phase, a query is first encoded with the same embedding model and then searched in the vector database with cosine similarity to obtain candidate sets. Chunks with similarity scores below a threshold (*i.e.*, 0.8) are excluded. After retrieval, candidates are ordered based on reranking scores computed by a pre-trained bce-reranker-base_v1 [35]. This involves feeding the concatenated query and each candidate into the reranker to estimate the relevance score. In this work, top-10 results are from initial search and top-3 reranked candidates are used for generation.

E. Missing Knowledge Detection

As shown in RQ3 (§III-D), the absence of knowledge in the retrieved chunks can lead chatbots to generate hallucinatory responses, which has severe impacts to mislead operators and potentially harm software systems. Therefore, we propose to detect knowledge unavailability to improve response trustworthiness and transparency. Inspired by [48], we identify knowledge unavailability with LLMs. Specifically, the LLM evaluates whether the chunks adequately address the revised query and, if not, generates a degraded response that both signals the knowledge insufficiency and suggests an alternative query answerable by the current data. The detection prompt is available in our anonymous repository [51].

F. Intent-guided Generation

Finally, we develop a prompt tailored for RAG-based chatbot in OpsQA to facilitate accurate responses. In particular, we concatenate the original query, rewritten query, and retrieved chunks with a prompt that instructs LLMs to answer the question based on the context provided. In addition, we incorporate instructions to improve output transparency by specifying the rewritten query in the responses. The complete prompt is available in our anonymous repository [51]. Finally, the LLM response with document references including document metadata and links is delivered to operators.

G. Implementation Details

We deploy iKnow on an Ubuntu 20.04 server with 64 CPUs, 128 GB RAM, and an NVIDIA Tesla T4 16 GB

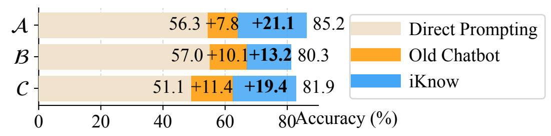

Fig. 7. The LLM judge accuracy of iKnow and the old chatbot.

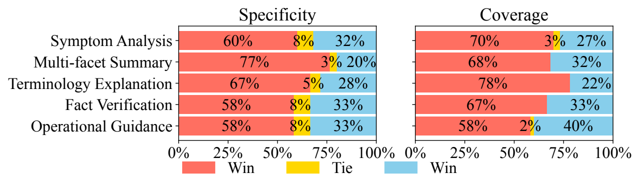

Fig. 8. The pairwise accuracy in different intents.

GPU. To balance performance and computational feasibility, we use the open-source Qwen-2.5-32B-instruct [36] model without fine-tuning for query rewriting, missing knowledge detection, and response generation. The LLM is deployed as an internal service on a remote server and accessed via an API. For retrieval, we use BGE-m3 [34] as the embedding model and bce-reranker-base_v1 [35] as the reranker. The retrieval similarity threshold is set to 0.8 and the number of retrieved chunks (k) is set to 3. All prompts used in iKnow are available in our anonymous repository [51] for reproducibility.

V. Experimental Results

We investigate the following research questions (RQs) to study the effectiveness and efficiency of iKnow in OpsQA.

- **RQ4**: How effective is iKnow in OpsQA?
- **RQ5**: How effective are different components?
- **RQ6**: How efficient is iKnow in OpsQA?

A. *RQ4: Effectiveness of iKnow*

Settings. We evaluate iKnow with an LLM-as-a-judge approach [54], [55], which instructs LLMs to assess answers based on predefined correctness criteria. This method accommodates the diverse valid expressions of LLM-generated answers and is widely adopted in RAG evaluations [55], [56]. We use three datasets introduced in § III with 2,000 queries for evaluation. For each query, a judge LLM (*i.e.*, Qwen-2.5-32B-instruct with a temperature of 0 to avoid randomness) evaluates answer correctness using the binary criteria defined in Table IV. We report LLM-judged accuracy as the main metric and validate judge's reliability via Pearson correlation [57] with human annotations on the old chatbot's answers ($r = 0.713$, indicating substantial agreement). Following [56], [58], we also conduct a pairwise win-tie-loss analysis, where the same judge LLM compares iKnow and the old system along two dimensions: *coverage* (degree to which all relevant aspects are addressed) and *specificity* (amount of precise, concrete information provided). We report the proportions of wins, ties, and losses for iKnow in each dimension.

Analysis. Figure 7 shows the LLM-judged accuracy across the three datasets. iKnow achieves high accuracy on all datasets, with 85.2%/ 80.3%/ 81.9% for $\mathcal{A}$/$\mathcal{B}$/$\mathcal{C}$, respectively. As open-source LLMs are not trained with the proprietary documents, directly querying LLMs achieves low accuracy, with 56.3%/ 57.0%/ 51.1% for $\mathcal{A}$/$\mathcal{B}$/$\mathcal{C}$. Equipped with our intent-guided framework, iKnow shows substantial improvements compared to the old system with naive RAG, with accuracy gains of 21.1%/ 13.2%/ 19.4% on the respective datasets. The average accuracy has been further improved from 65.8% to 81.3%. These results indicate that iKnow is effective in generating correct responses for cloud operators.

Fig. 8 shows the mean win-rate of iKnow's answers in varying intents on three datasets. We find that iKnow consistently outperforms the old system in specificity and coverage across all query intents. Notably, for terminology explanation queries, iKnow shows the highest improvement in coverage (win rate: 78%), whereas for multi-facet summary queries, the greatest gain is in specificity (win rate: 77%). The improvements likely stem from enhanced rewritten queries, which clarifies query intent and expands required details, enabling the LLM to generate more targeted and precise answers.

Error Analysis To further assess the limitations of iKnow, we conduct an analysis of the 374 errors identified by the LLM judge. We find five errors types: (1) *Intent detection errors* (23.0%), where the system misclassifies the user's intent, leading to inappropriate query rewriting and ultimately incorrect responses (*e.g.*, interpreting a symptom analysis query as a fact verification query); (2) *Retrieval errors* (12.6%), where relevant documents are not ranked highly enough, often due to insufficient semantic overlap between the query and documents; (3) *Missing knowledge* (37.2%), where the required information is absent from the current knowledge base (*e.g.*, queries needing real-time monitoring data or external resources); (4) *LLM judge mislabelling* (16.6%), where LLM judges label a correct answer as incorrect, reflecting limitations in automatic evaluation; and (5) *Other errors* (9.6%). Notably, most errors stem from missing knowledge, indicating the limitations in the current knowledge base in covering all operator needs. Intent detection and retrieval errors also remain significant, suggesting further improvements in these components could enhance accuracy. Moreover, LLM judge mislabelling highlights the need for more reliable evaluation, such as selective human review. These insights inform future work on expanding knowledge coverage, refining intent detection, and improving evaluation protocols.

B. *RQ5: Effectiveness of different components*

We conduct three ablation studies to evaluate the effectiveness of three core components in iKnow, *i.e.*, intent detection, query rewriting, and missing knowledge detection.

Intent Detection To evaluate the precision of intent detection, we perform a ten-fold cross-validation [59] with all 2,000 examples. Given the cost and effort of building an intent dataset of such a size in practice, we also explore how detection accuracy varies with different amounts of training data. To achieve this, we incrementally increase the training set size by 5% steps and report accuracy on held-out test set. Fig. 9 shows the results. We find that with the full dataset, the prototypical network achieves a mean accuracy of 85.3% in classifying intents, indicating its effectiveness in guiding the rewriting thereafter. Furthermore, accuracy improves steadily as more training data is added, reaching a plateau of 84.7% with approximately 40% of data (*i.e.*, 720 labels). This suggests that effective intent detection can be achieved with a relatively small amount of labeled data, demonstrating the efficiency of prototypical networks in low-resource settings.

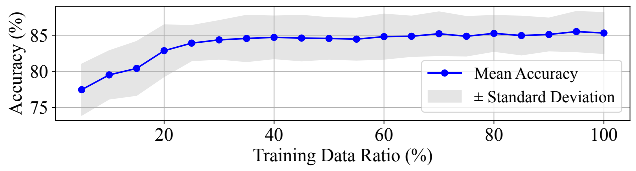

Fig. 9. Accuracy of intend detection.

Query Rewriting A rewritten query that complies user's intent while being well-formatted is beneficial to retrieving more relevant documents and instructing LLMs to respond. Therefore, we evaluate the effectiveness of query rewriting. Specifically, we use the judge LLM to assess rewritten queries for all the 2,000 samples. For each query, the judge provides a binary label for whether the rewritten query better expresses and enriches the user's original intent without altering its meaning. Our results show that 99% of queries are enhanced in intents while keeping coherence and completeness, indicating the effectiveness of rewriting process. Table VI shows some enhanced rewritten query examples.

TABLE VI
EXAMPLES OF QUERY REWRITING CASES FOR CLOUD OPERATION QA

Intent	Original Query	Rewritten Query
Symptom Analysis	BlockA reserved block < threshld alarm?	Why and how to solve the BlockA reserved block count falling below a threshold that triggers an alarm?
Multi-facet Summary	High-level performance troubleshooting guide?	How to conduct a comprehensive performance troubleshooting? What key indicators and steps should be included? What are the common diagnostic tools?
Terminology Explanation	mdc process?	What is a MDC process? Explain its concept, functions, and role in detail.
Fact Verification	Is it possible to configure available partitions to be partially visible?	Can available partitions be configured to be visible only to certain users? How can such a configuration be performed?
Operational Guidance	A-Manager upgrade?	How to perform A-Manager upgrade operation? Please provide detailed steps.

Missing Knowledge Detection To evaluate the accuracy of detecting missing knowledge in the retrieved chunks, we have developed a binary classification dataset using pre-existing labels. Specifically, the negative examples (*i.e.*, where knowledge is missing) contain 60 cases that exhibit knowledge missing failures in § III-D. On the other hand, positive examples (*i.e.*, where knowledge is sufficient) are randomly chosen from correct cases in equal numbers, based on the assumption that adequate context is necessary for accurate responses. Experimental results show that our LLM-based detection module can effectively determine whether necessary knowledge is present in the context, achieving an accuracy of 94.3% precision, 83.3% recall, and 88.5% F1 score in classification. The precise detection serves as a foundation, where iKnow can selectively comply to user queries, which can be introduced in the responses to improve transparency and user trust. The precise detection forms the basis for more

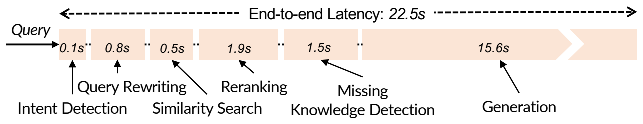

Fig. 10. End-to-end and component-wise latency of iKnow.

transparent and reliable responses by offering guidelines of selectively complying to queries [48], [60].

C. RQ6: Efficiency of iKnow

Efficiency is critical for industrial QA systems, as high latency can negatively impact user satisfaction [61], [62] In this RQ, we evaluate the efficiency of iKnow by end-to-end and component-wise latency on all the 2,000 samples. Specifically, we record the time spent on different components and report the mean latency. As shown in Fig. 10, the mean end-to-end latency of iKnow is 22.5 seconds, which is significantly lower than the time of manual knowledge retrieving by cloud operators. A detailed breakdown shows that answer generation accounts for the largest portion of the total latency (69%), primarily due to the extensive computation of LLM decoding over extended context. Additionally, the extra time introduced by intent detection, query rewriting, two-stage retrieval and missing knowledge detection accounts for just 19.1% of total latency. which improves answer accuracy and transparency with modest overhead.

Summary of RQ4–RQ6: Our intent-aware enhancements, *i.e.*, intent detection, query rewriting, and missing knowledge detection, significantly improve the accuracy and comprehensiveness of answers while necesitates a modest increase of latency in RAG-based OpsQA, enabling efficient and practical support for cloud operators.

VI. LESSONS LEARNED

We have deployed iKnow in CloudA for six months, providing automatic QA services for thousands of users from diverse operation teams, including customer service, on-call engineers, cloud testing engineers, DevOps engineering, *etc.* In the following, we introduce the lessons learned from our deployment experience and how our results can implicate chatbot users, OpsQA chatbot provider, and researcher to better develop and leverage OpsQA chatbots.

A. Implications for Chatbot Users

Formulating a complete question with explicit information need and specific details is of great importance. Our findings from RQ1-2 reveal that incomplete queries account for approximately one third of failed answers, particularly in terminology explanation scenarios (§ III-D). Interacting with chatbots is different from the way we use a search engine. Therefore, instead of directly entering terminology or pasting a piece of error code, users should clarify their need and specific details in the question to improve the efficiency of knowledge acquisition.

Be aware of knowledge boundary of chatbots and ask relevant questions. Although we inform the users of the included knowledge bases, we find iKnow still receives operation-irrelevant questions that seek knowledge from other domains such as coding questions (RQ3). Though chatbots can respond to these questions, the correctness and trustworthiness is questionable, as LLMs can produce hallucinated answers in the absence of proper knowledge. Therefore, users should ask relevant questions to best assist in their work.

Exercise caution and verify critical responses. While iKnow generally provides accurate answers grounded on operational knowledge, occasional errors or hallucinations still exist (RQ4). Therefore, users are recommended to verify chatbot responses by checking the linked source documents for reference, especially for those high-impact decisions. This practice helps mitigate risks associated with overreliance on automated answers.

B. Implications for OpsQA Chatbot Provider

Be aware of deployment challenges and approach to monitoring data. Crafting a successful RAG-based chatbot for operation, while promising, is not easy. The process demands meticulous engineering of RAG pipelines, such as design and parameter choices in each stage of the pipeline [30], [44]. Our success experience suggest that OpsQA chatbot providers can pinpoint deficiency and fix problems of their systems through guided failure analysis. They should carefully approach to the data produced in the pipeline, analyze them to understand the running status of chatbot service and make improvements in order to build a reliable OpsQA chatbot.

OpsQA chatbot providers can benefit from integrating intent detection and query rewriting. As shown in § V-B, iKnow can effectively predict query intents and produce enhanced rewritten query. OpsQA chatbot providers can leverage iKnow to help users formulate a better question with intent detection (*e.g.*, terminology explanation) and query rewriting. These approaches can improve the response quality and finally improve user experience.

Including links to reference documents in the answer enhances transparency. We implemented this strategy in iKnow and noticed that it was appreciated by chatbot users. With a clear connection to authoritative sources, users can directly verify the origin and context of the information provided. This approach can help minimize the risk of misinformation and reinforce user trust in the generated responses, ultimately contributing to a more reliable chatbot.

C. Implications for Researchers

Synthesizing domain-specific QA datasets for customized training. Fine-tuning a personalized LLM on domain-specific corpora can significantly improve QA performance [63], however the challenges lie in how to build SFT data reflecting the style and diversity of target domains [64]. Our intent taxonomy reflects the real distribution of queries and operator preferences in cloud operation, which can serve as a blueprint for synthesizing cloud operation QA datasets. Future works could explore training a specialized LLM on operation data to enhance answer accuracy.

Incorporating real-time monitoring data for symptom analysis. A subset of users has turned to iKnow for diagnosing system failures, a task that often requires insight into the current system status. Traditional static document corpora lack the real-time, dynamic data captured in logs and traces. Researchers are encouraged to explore methods that integrate diagnostic data into RAG systems to better address symptom analysis problems with more accurate diagnostics.

Aggregating evidence for multi-perspective questions. As shown in § III-B, users often submit complex questions that demand explanations from several angles. In these cases, the relevant evidence may be dispersed across multiple document chunks, rendering direct answers insufficient. Advanced techniques, such as GraphRAG [65], [66] that aggregate evidence from diverse sources, offer promising solutions for these scenarios. Future work could focus on synthesizing coherent responses from fragmented data.

VII. Threats to Validity

1) Internal Threats: The manual labeling process in our study(§ III and § V) may introduce subjective bias. To address this, we apply open coding with clear guidelines, consensus discussions, and measured inter-annotator agreement using Krippendorff's alpha. Additionally, to ensure consistency in LLM outputs, we set the temperature to 0, eliminating randomness for identical input contexts. Although queries are randomly sampled from three systems, selection bias may remain, as our data covers only a two-month period and may not capture all operational scenarios. Furthermore, despite our efforts to define annotation guidelines clearly, some intent and root causes may still be ambiguous or overlapping, potentially affecting annotation consistency. Finally, using an LLM as a judge may introduce bias, as its assessments might not always align with human judgment, especially for nuanced cases.

2) External Threats: Our study is conducted exclusively in CloudA; however, we believe that the queries in OpsQA are common, and our findings can be generalized to other cloud platforms mainly for two reasons. Firstly, the studied documents include common operational sources, including production manuals, incident tickets, FAQs, *etc*, which are analogical to those of other platforms [20], [21]. Secondly, we construct datasets from three teams with varying document types and tasks, and observed that while distributions differed, the main intent and root cause categories were consistent. Nevertheless, further validation in other companies and on public platforms (*e.g.*, StackOverflow) is needed to confirm broader applicability. Furthermore, we examine proprietary, internal operational queries in companies. Public conversations, like those on StackOverflow and forums, may involve variations in question formulation and terminology, potentially impacting the external validity of our research.

VIII. Related Works

Prior studies have explored question answering in various software engineering (SE) domains, including programming [67], software development [6], [68], security [69], testing [70], and many more [71]–[73]. These efforts motivate the development of chatbots [24]–[26] to support SE tasks by providing automated, natural language answers [24], [25]. However, most focus on developer-centric scenarios, with limited attention to operational contexts such as cloud management. Despite promise, developing chatbots often relies on large-scale labeled data and costly model training, limiting their adoption and scalability [8], [74]. Recent advances in large language models (LLMs) and retrieval-augmented generation (RAG) have enabled more capable chatbots that can leverage domain-specific documents [30], [64], [69], [75]. While some studies have reported on building chatbots for developer support [64], [76], there is limited research focused specifically on the unique needs of cloud operators.

Cloud operation involves managing and maintaining complex infrastructures, requiring operators to follow standard procedures documented in manuals and troubleshooting guides [20], [21], [77]–[79]. Accessing and interpreting this information efficiently remains a challenge. Many methods are proposed to assist cloud operators by analyzing monitoring data [80]–[84], diagnosing incidents [85]–[87], writing scripts [5], [21], providing mitigation actions [88], [89] and conducting postmortem analysis [90]. Although these methods provide valuable task-specific insights, they often overlook the broader spectrum of information needs in OpsQA, where operators seek diverse resources such as diagnosis and mitigation steps. Recent studies introduce benchmarks and fine-tuned LLMs for OpsQA [8]–[10], [74], [91], but they focus model improvement on artificial, exam-style questions and overlook practical challenges like ambiguous queries and integration of dynamic, proprietary knowledge. A most-similar and closely related work is [92], which employs RAG with a query rewriting module to tackle similar real-world challenges of query-vagueness in CI/CD domain QA. Our work enhances the query rewriting with an intent-guided design and incorporates missing knowledge detection, which are directly motivated by our comprehensive empirical findings.

Some works share lessons of deploying RAG-based chatbots. For example, some works [30], [31], [45] identify common failure points and propose solutions addressing issues such as content freshness and system architecture. We encountered similar challenges during development. However, these studies rarely examine the impact of query intents on chatbot failures, especially in OpsQA, and lack quantitative analyses to inform practitioners about the prevalence of reliability issues. Beyond chatbots, RAG reliability has also been studied in domains such as software defect detection [93] and code generation [94]. In contrast, we focus on chatbot reliability for cloud operations, with an emphasis on understanding query intent and its relationship to system failures.

IX. Conclusion

In this paper, we share our experience in developing a RAG-based chatbot for OpsQA at CloudA. Towards this goal, we first performed empirical studies to identify five intents of OpsQA queries and assessed the performance of the current RAG framework using 2,000 operational queries from three deployed chatbots. Our analysis revealed six root causes and their respective distributions, highlighting two critical issues: query incompleteness and knowledge deficiency. To mitigate these failures, we introduce iKnow, an intent-guided RAG-based chatbot designed to meet a variety of informational needs. Experiment results show the effectiveness and efficiency of our approach. Additionally, we discuss lessons learned for chatbot users, providers, and researchers involved in OpsQA. With our successful implementation and gained experience, we believe that a reliable RAG-based chatbot can significantly enhance the efficiency of cloud operators.

Acknowledgment

We extend our sincere gratitude to the anonymous reviewers for their constructive feedback. We also thank Renyi Zhong, Yulun Wu, Jinyang Liu, Jiazhen Gu, Ziyu Liu, Yijing Liu, Peng Han, Qiao Chen, Yi Yang, Cong Feng for their valuable discussions and annotations. The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. SRFS2425-4S03 of the Senior Research Fellow Scheme and No. CUHK 14209124 of the General Research Fund).

X. Data Availability

The source code and prompt of iKnow's modules can be found in https://github.com/Jun-jie-Huang/iKnow. However, due to privacy concerns, we do not plan to release the entire chatbot system and data.

References

[1] "What is microsoft azure?" 2025, [Online; accessed 29 May 2025]. [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure

[2] "Cloud computing with aws," 2025, [Online; accessed 29 May 2025]. [Online]. Available: https://aws.amazon.com/what-is-aws/

[3] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," *Acm Computing Surveys (CSUR)*, vol. 44, no. 1, pp. 1–50, 2012.

[4] A. Białecki, R. Muir, G. Ingersoll, and L. Imagination, "Apache lucene 4," in *SIGIR 2012 workshop on open source information retrieval*. sn, 2012, p. 17.

[5] Y. Jiang, C. Zhang, S. He, Z. Yang, M. Ma, S. Qin, Y. Kang, Y. Dang, S. Rajmohan, Q. Lin *et al.*, "Xpert: Empowering incident management with query recommendations via large language models," in *ICSE*, 2024, pp. 1–13.

[6] Y. Tian, F. Thung, A. Sharma, and D. Lo, "Apibot: question answering bot for api documentation," in *ASE*. IEEE, 2017, pp. 153–158.

[7] A. Lill, A. N. Meyer, and T. Fritz, "On the helpfulness of answering developer questions on discord with similar conversations and posts from the past," in *ICSE*, 2024, pp. 1–13.

[8] H. Guo, J. Yang, J. Liu, L. Yang, L. Chai, J. Bai, J. Peng, X. Hu, C. Chen, D. Zhang *et al.*, "Owl: A large language model for it operations," in *ICLR*, 2024.

[9] Y. Liu, C. Pei, L. Xu, B. Chen, M. Sun, Z. Zhang, Y. Sun, S. Zhang, K. Wang, H. Zhang *et al.*, "Opseval: A comprehensive task-oriented aiops benchmark for large language models," *arXiv preprint arXiv:2310.07637*, 2023.

[10] Y. Miao, Y. Bai, L. Chen, D. Li, H. Sun, X. Wang, Z. Luo, Y. Ren, D. Sun, X. Xu *et al.*, "An empirical study of netops capability of pre-trained large language models," *arXiv preprint arXiv:2309.05557*, 2023.

[11] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin *et al.*, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, pp. 1–55, 2025.

[12] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *NeurIPS*, pp. 9459–9474, 2020.

[13] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, "A survey on rag meeting llms: Towards retrieval-augmented large language models," in *KDD*, 2024, pp. 6491–6501.

[14] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?(nier track)," in *ICSE*, 2011, pp. 804–807.

[15] S. Beyer and M. Pinzger, "A manual categorization of android app development issues on stack overflow," in *ICSME*, 2014, pp. 531–535.

[16] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," *NeurIPS*, 2017.

[17] T. Ma, Q. Wu, Z. Yu, T. Zhao, and C.-Y. Lin, "On the effectiveness of sentence encoding for intent detection meta-learning," in *Proceedings of NAACL: HLT*, 2022, pp. 3806–3818.

[18] A. Spadaccini and K. Guliani, "Being an on-call engineer: A google sre perspective," *;login:*, vol. 40, pp. 43–47, 2015. [Online]. Available: https://www.usenix.org/publications/login/oct15/spadaccini

[19] P. Hamadanian, B. Arzani, S. Fouladi, S. K. R. Kakarla, R. Fonseca, D. Billor, A. Cheema, E. Nkposong, and R. Chandra, "A holistic view of ai-driven network incident management," in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, 2023, pp. 180–188.

[20] L. Zhang, T. Jia, M. Jia, Y. Wu, A. Liu, Y. Yang, Z. Wu, X. Hu, P. S. Yu, and Y. Li, "A survey of aiops for failure management in the era of large language models," *arXiv preprint arXiv:2406.11213*, 2024.

[21] M. Shetty, C. Bansal, S. P. Upadhyayula, A. Radhakrishna, and A. Gupta, "Autotsg: learning and synthesis for incident troubleshooting," in *ES-EC/FSE*, 2022, pp. 1477–1488.

[22] S. Ghosh, M. Shetty, C. Bansal, and S. Nath, "How to fight production incidents? an empirical study on a large-scale cloud service," in *SoCC*, 2022, pp. 126–141.

[23] X. Zhou, G. Li, Z. Sun, Z. Liu, W. Chen, J. Wu, J. Liu, R. Feng, and G. Zeng, "D-bot: Database diagnosis system using large language models," *Proceedings of the VLDB Endowment*, pp. 2514–2527, 2024.

[24] S. Lambiase, G. Catolino, F. Palomba, and F. Ferrucci, "Motivations, challenges, best practices, and benefits for bots and conversational agents in software engineering: A multivocal literature review," *ACM Computing Surveys*, vol. 57, no. 4, pp. 1–37, 2024.

[25] C. Lebeuf, M.-A. Storey, and A. Zagalsky, "Software bots," *IEEE Software*, vol. 35, no. 1, pp. 18–23, 2017.

[26] Q. Motger, X. Franch, and J. Marco, "Software-based dialogue systems: survey, taxonomy, and challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–42, 2022.

[27] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *TOSEM*, pp. 1–79, 2024.

[28] "Azure ai-powered assistant," 2025, [Online; accessed 29 May 2025]. [Online]. Available: https://azure.microsoft.com/en-us

[29] "Qnabot on aws," 2025, [Online; accessed 29 May 2025]. [Online]. Available: https://aws.amazon.com/cn/solutions/implementations/qnabot-on-aws/

[30] R. Akkiraju, A. Xu, D. Bora, T. Yu, L. An, V. Seth, A. Shukla, P. Gundecha, H. Mehta, A. Jha *et al.*, "Facts about building retrieval augmented generation-based chatbots," *arXiv preprint arXiv:2407.07858*, 2024.

[31] S. Barnett, S. Kurniawan, S. Thudumu, Z. Brannelly, and M. Abdelrazek, "Seven failure points when engineering a retrieval augmented generation system," in *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*, 2024, pp. 194–199.

[32] H. Chase, "LangChain," Oct. 2022. [Online]. Available: https://github.com/langchain-ai/langchain

[33] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," *arXiv preprint arXiv:2401.08281*, 2024.

[34] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," in *Findings of ACL*, 2024, pp. 2318–2335.

[35] I. NetEase Youdao, "Bcembedding: Bilingual and crosslingual embedding for rag," https://github.com/netease-youdao/BCEmbedding, 2023.

[36] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei *et al.*, "Qwen2. 5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.

[37] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: a critical review and guidelines," in *ICSE*, 2016, pp. 120–131.

[38] Z. Jiang, J. Huang, G. Yu, Z. Chen, Y. Li, R. Zhong, C. Feng, Y. Yang, Z. Yang, and M. Lyu, "L4: Diagnosing large-scale llm training failures via automated log analysis," in *Proceedings of the FSE*, 2025, pp. 51–63.

[39] J. M. Corbin, *Grounded theory in practice*. Sage, 1997.

[40] J. A. Holton, "The coding process and its challenges," *The Sage handbook of grounded theory*, vol. 3, pp. 265–289, 2007.

[41] K. Krippendorff, "Computing krippendorff's alpha-reliability," 2011.

[42] "Amazon Mechanical Turk," Oct. 2022. [Online]. Available: https://www.mturk.com

[43] S. Beyer, C. Macho, M. Pinzger, and M. Di Penta, "Automatically classifying posts into question categories on stack overflow," in *ICPC*, 2018, pp. 211–221.

[44] P. Fathollahzadeh, M. E. Mezouar, H. Li, Y. Zou, and A. E. Hassan, "Towards refining developer questions using llm-based named entity recognition for developer chatroom conversations," *arXiv preprint arXiv:2503.00673*, 2025.

[45] W. Glantz, "12 rag pain points and proposed solutions." 2025, [Online; accessed 29 May 2025]. [Online]. Available: https://towardsdatascience.com/12-rag-pain-points-and-proposed-solutions-43709939a28c/

[46] C. Choi, J. Kwon, J. Ha, H. Choi, C. Kim, Y. Lee, J.-y. Sohn, and A. Lopez-Lira, "Finder: Financial dataset for question answering and evaluating retrieval-augmented generation," *arXiv preprint arXiv:2504.15800*, 2025.

[47] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan, "Query rewriting in retrieval-augmented large language models," in *EMNLP*, 2023, pp. 5303–5315.

[48] X. Peng, P. K. Choubey, C. Xiong, and C.-S. Wu, "Unanswerability evaluation for retreival augmented generation," *arXiv preprint arXiv:2412.12300*, 2024.

[49] M. Haering, C. Stanik, and W. Maalej, "Automatically matching bug reports with related app reviews," in *ICSE*. IEEE, 2021, pp. 970–981.

[50] Z. Eberhart and C. McMillan, "Generating clarifying questions for query refinement in source code search," in *SANER*, 2022, pp. 140–151.

[51] "Opensource repository of iknow," 2025, [Online; accessed 30 Sep 2025]. [Online]. Available: https://github.com/Jun-jie-Huang/iKnow

[52] F. Cuconasu, G. Trappolini, F. Siciliano, S. Filice, C. Campagnano, Y. Maarek, N. Tonellotto, and F. Silvestri, "The power of noise: Redefining retrieval for rag systems," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024, pp. 719–729.

[53] M. Adeyemi, A. Oladipo, R. Pradeep, and J. Lin, "Zero-shot cross-lingual reranking with large language models for low-resource languages," in *Proceedings of ACL*, 2024, pp. 650–656.

[54] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *NeurIPS*, pp. 46 595–46 623, 2023.

[55] J. Gu, X. Jiang, Z. Shi, H. Tan, X. Zhai, C. Xu, W. Li, Y. Shen, S. Ma, H. Liu *et al.*, "A survey on llm-as-a-judge," *arXiv preprint arXiv:2411.15594*, 2024.

[56] S. Wang, X. Yu, M. Wang, W. Chen, Y. Zhu, and Z. Dou, "Richrag: Crafting rich responses for multi-faceted queries in retrieval-augmented generation," in *COLING*, 2025, pp. 11 317–11 333.

[57] P. Sedgwick, "Pearson's correlation coefficient," *Bmj*, vol. 345, 2012.

[58] S. Es, J. James, L. E. Anke, and S. Schockaert, "Ragas: Automated evaluation of retrieval augmented generation," in *EACL: System Demonstrations*, 2024, pp. 150–158.

[59] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, 1995, pp. 1137–1145.

[60] F. Brahman, S. Kumar, V. Balachandran, P. Dasigi, V. Pyatkin, A. Ravichander, S. Wiegreffe, N. Dziri, K. Chandu, J. Hessel *et al.*, "The art of saying no: Contextual noncompliance in language models," *NeurIPS*, pp. 49 706–49 748, 2024.

[61] C. Jin, Z. Zhang, X. Jiang, F. Liu, X. Liu, X. Liu, and X. Jin, "Ragcache: Efficient knowledge caching for retrieval-augmented generation," *arXiv preprint arXiv:2404.12457*, 2024.

[62] Z. Jiang, Y. Huang, G. Yu, J. Huang, J. Gu, and M. R. Lyu, "Hierarchical prediction-based management for lmaas systems," *arXiv preprint arXiv:2504.03702*, 2025.

[63] G. Dong, H. Yuan, K. Lu, C. Li, M. Xue, D. Liu, W. Wang, Z. Yuan, C. Zhou, and J. Zhou, "How abilities in large language models are affected by supervised fine-tuning data composition," in *ACL*, 2024, pp. 177–198.

[64] X. Liang, J. Ren, J. Qi, C. Peng, and B. Jiang, “Dialogagent: An auto-engagement agent for code question answering data production,” in *ICSE-SEIP*, 2025.

[65] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, “From local to global: A graph rag approach to query-focused summarization,” *arXiv preprint arXiv:2404.16130*, 2024.

[66] Y. Zhou, Y. Su, Y. Sun, S. Wang, T. Wang, R. He, Y. Zhang, S. Liang, X. Liu, Y. Ma *et al.*, “In-depth analysis of graph-based rag in a unified framework,” *arXiv preprint arXiv:2503.04338*, 2025.

[67] C. Liu and X. Wan, “Codeqa: A question answering dataset for source code comprehension,” in *Findings of EMNLP*, 2021, pp. 2618–2632.

[68] J. Huang, D. Tang, L. Shou, M. Gong, K. Xu, D. Jiang, M. Zhou, and N. Duan, “Cosqa: 20,000+ web queries for code search and question answering,” in *ACL-IJANLP*, 2021, pp. 5690–5700.

[69] A. Sajadi, B. Le, A. Nguyen, K. Damevski, and P. Chatterjee, “Do llms consider security? an empirical study on responses to programming questions,” *arXiv preprint arXiv:2502.14202*, 2025.

[70] D. Okanović, S. Beck, L. Merz, C. Zorn, L. Merino, A. van Hoorn, and F. Beck, “Can a chatbot support software engineers with load testing? approach and experiences,” in *ICPE*, 2020, pp. 120–129.

[71] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, “Ai-based question answering assistance for analyzing natural-language requirements,” in *ICSE*. IEEE, 2023, pp. 1277–1289.

[72] A. Bansal, Z. Eberhart, L. Wu, and C. McMillan, “A neural question answering system for basic questions about subroutines,” in *SANER*. IEEE, 2021, pp. 60–71.

[73] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager *et al.*, “Building watson: An overview of the deepqa project,” *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[74] F. Yang, P. Zhao, Z. Wang, L. Wang, B. Qiao, J. Zhang, M. Garg, Q. Lin, S. Rajmohan, and D. Zhang, “Empower large language model to perform better on industrial domain-specific question answering,” in *EMNLP: Industry Track*, 2023, pp. 294–312.

[75] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, “The programmer’s assistant: Conversational interaction with a large language model for software development,” in *IUI*, 2023, pp. 491–514.

[76] D. Abrahamyan and F. H. Fard, “Stackrag agent: Improving developer answers with retrieval-augmented generation,” in *ICSME*. IEEE, 2024, pp. 893–897.

[77] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu *et al.*, “Towards intelligent incident management: why we need it and how we make it,” in *ESEC/FSE*, 2020, pp. 1487–1497.

[78] P. Notaro, J. Cardoso, and M. Gerndt, “A survey of aiops methods for failure management,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 12, no. 6, pp. 1–45, 2021.

[79] M. Shetty, C. Bansal, S. Kumar, N. Rao, N. Nagappan, and T. Zimmermann, “Neural knowledge extraction from cloud service incidents,” in *ICSE-SEIP*. IEEE, 2021, pp. 218–227.

[80] J. Gu, C. Luo, S. Qin, B. Qiao, Q. Lin, H. Zhang, Z. Li, Y. Dang, S. Cai, W. Wu *et al.*, “Efficient incident identification from multi-dimensional issue reports via meta-heuristic search,” in *ESEC/FSE*, 2020, pp. 292–303.

[81] J. Liu, S. He, Z. Chen, L. Li, Y. Kang, X. Zhang, P. He, H. Zhang, Q. Lin, Z. Xu *et al.*, “Incident-aware duplicate ticket aggregation for cloud systems,” in *ICSE*. IEEE, 2023, pp. 2299–2311.

[82] J. Kuang, J. Liu, J. Huang, R. Zhong, J. Gu, L. Yu, R. Tan, Z. Yang, and M. R. Lyu, “Knowledge-aware alert aggregation in large-scale cloud systems: a hybrid approach,” in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, 2024, pp. 369–380.

[83] J. Liu, J. Huang, Y. Huo, Z. Jiang, J. Gu, Z. Chen, C. Feng, M. Yan, and M. R. Lyu, “Scalable and adaptive log-based anomaly detection with expert in the loop,” *CoRR*, 2023.

[84] J. Huang, Z. Jiang, J. Liu, Y. Huo, J. Gu, Z. Chen, C. Feng, H. Dong, Z. Yang, and M. R. Lyu, “Demystifying and extracting fault-indicating information from logs for failure diagnosis,” in *ISSRE*. IEEE, 2024, pp. 511–522.

[85] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan, “Recommending root-cause and mitigation steps for cloud incidents using large language models,” in *ICSE*, 2023, pp. 1737–1749.

[86] 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 of FSE*, 2024, pp. 266–277.

[87] D. Zhang, X. Zhang, C. Bansal, P. Las-Casas, R. Fonseca, and S. Rajmohan, “Lm-pace: Confidence estimation by large language models for effective root causing of cloud incidents,” in *Companion of FSE*, 2024, pp. 388–398.

[88] P. T. Isaza, M. Nidd, N. Zheutlin, J.-w. Ahn, C. A. Bhatt, Y. Deng, R. Mahindru, M. Franz, H. Florian, and S. Roukos, “Retrieval augmented generation-based incident resolution recommendation system for it support,” *arXiv preprint arXiv:2409.13707*, 2024.

[89] K. An, F. Yang, L. Li, Z. Ren, H. Huang, L. Wang, P. Zhao, Y. Kang, H. Ding, Q. Lin *et al.*, “Nissist: An incident mitigation copilot based on troubleshooting guides,” *arXiv preprint arXiv:2402.17531*, 2024.

[90] J. Huang, J. Liu, Z. Chen, Z. Jiang, Y. Li, J. Gu, C. Feng, Z. Yang, Y. Yang, and M. R. Lyu, “Faultprofit: Hierarchical fault profiling of incident tickets in large-scale cloud systems,” in *ICSE-SEIP*, 2024, pp. 392–404.

[91] T. Zhang, Z. Jiang, S. Bai, T. Zhang, L. Lin, Y. Liu, and J. Ren, “Rag4itops: A supervised fine-tunable and comprehensive rag framework for it operations and maintenance,” in *EMNLP: Industry Track*, 2024, pp. 738–754.

[92] D. Chaudhary, S. L. Vadlamani, D. Thomas, S. Nejati, and M. Sabetzadeh, “Developing a llama-based chatbot for ci/cd question answering: A case study at ericsson,” in *ICSME*, 2024, pp. 707–718.

[93] Y. Shao, Y. Huang, J. Shen, L. Ma, T. Su, and C. Wan, “Are llms correctly integrated into software systems?” in *ICSE*, 2025, pp. 741–741.

[94] S. Zhao, Y. Huang, J. Song, Z. Wang, C. Wan, and L. Ma, “Towards understanding retrieval accuracy and prompt quality in rag systems,” *arXiv preprint arXiv:2411.19463*, 2024.