Exploring LLM-Based Agents for Root Cause Analysis

Devjeet Roy*
devjeet.roy@wsu.edu
Washington State University
Pullman, Washington, USA

Xuchao Zhang
xuchaozhang@microsoft.com
Microsoft
Redmond, Washington, USA

Rashi Bhave
Microsoft
Bengaluru, Karnataka, India

Chetan Bansal
chetanb@microsoft.com
Microsoft
Redmond, Washington, USA

Pedro Las-Casas
pedrobr@microsoft.com
Microsoft
Redmond, Washington, USA

Rodrigo Fonseca
Fonseca.Rodrigo@microsoft.com
Microsoft
Redmond, Washington, USA

Saravan Rajmohan
saravan.rajmohan@microsoft.com
Microsoft
Redmond, Washington, USA

ABSTRACT

Root cause analysis (RCA), a critical part of the incident management process, is a demanding task for on-call engineers, requiring deep domain knowledge and extensive experience with a team's specific services. Automation of RCA can result in significant savings of time, and ease the burden of incident management on on-call engineers. Recently, researchers have utilized Large Language Models (LLMs) to perform RCA, and have demonstrated promising results. However, these approaches are not able to dynamically collect additional diagnostic information such as incident related logs, metrics or databases, severely restricting their ability to diagnose root causes. In this work, we explore the use of LLM based agents for RCA to address this limitation. We present a thorough empirical evaluation of a ReAct agent equipped with retrieval tools, on an out-of-distribution dataset of production incidents collected at Microsoft. Results show that ReAct performs competitively with strong retrieval and reasoning baselines, but with highly increased factual accuracy. We also conduct a case study with a team at Microsoft to allow the ReAct agent to interface with diagnostic services that are used by the team for manual RCA. Our results show how agents can overcome the limitations of prior work, and considerations for implementing such a system in practice.

CCS CONCEPTS

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

*This work was done during an internship at Microsoft.

KEYWORDS

Incident Management, Cloud Computing, Root Cause Analysis, AIOps

ACM Reference Format:
Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring LLM-Based Agents for Root Cause Analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3663529.3663841

1 INTRODUCTION

For the last several decades, large scale enterprises have been transforming their software into cloud services. With the rise of Artificial Intelligence (AI) in recent years, there has been even greater movement of computation from consumer devices to the cloud. This shift in paradigm has brought with it complex software systems that are characterized by multi-tiered architectures, microservices and distributed applications. The increased complexity of these systems makes them highly susceptible to production incidents, which can incur substantial costs and disrupt critical services. Therefore, prompt mitigation and resolution of these incidents is crucial to maintaining service availability and reliability [41]. However, cloud incident management [15, 7] is extremely labor-intensive. On-call engineers (OCEs) require extensive experience with a team's services and deep domain knowledge to be effective at incident management. Even for experienced OCEs, incident management represents a time-intensive endeavor. As software systems continue scaling in size and complexity, the demands placed on OCEs and incident management systems is only bound to increase in the future. To address these challenges, the field of AIOps (Artificial Intelligence for IT Operations) has proposed numerous techniques to ease incident management. Despite these developments, several parts of the incident management lifecycle still largely rely on human intervention.

One of the most challenging aspects of cloud incident management is root cause analysis (RCA). Before an incident can be resolved, OCEs must identify the root cause of the incident to ensure that any resolution actions comprehensively and correctly fix the incident. RCA represents one of the most labor- and skill-intensive components of the incident management lifecycle [17]. Even a veteran software engineer might need to spend several years on a team before they are able to effectively perform RCA on a team's services. Therefore, it comes as no surprise that researchers have tried to automate parts of this process. Numerous techniques have been proposed to assist OCEs with RCA, such as incident prioritization and retrieval of similar historical incidents. Recently, Ahmed et al.[1] proposed the use of fine-tuned LLMs for incident root cause analysis and mitigation. They showed that LLMs can find root causes of incidents even when working with a very limited set of information about an incident. Chen et al.[8] propose RCACopilot, which expands upon this work and adds retrieval augmentation and diagnostic collection procedures to the LLM-based root cause analysis pipeline.

While these approaches have shown promising results on the ability of LLMs to perform RCA, neither equips the LLM to dynamically query real time diagnostic information about the service(s) affected by an incident. RCACopilot [8] relies on predefined handlers that must be engineered by hand, and predicts root cause categories rather than specific root causes, while Ahmed et al.[1] rely only on the incident title and description for predicting the root cause. What's missing here is a critical step that is taken by OCEs in real world RCA scenarios - *for any incident, one of the first steps performed by OCEs is collection of novel diagnostic data that is not present in the incident report.* In prior work, LLMs do not have the ability to interact with the outside environment to be able to collect this data. In this work, we propose the use of LLM-based agents – systems that can reason, plan and interact with the external environment to collect new information – to address this limitation and help with root cause analysis.

Despite the remarkable capabilities demonstrated by LLM-based agents across diverse domains and tasks, adapting them for the purposes of RCA represents a significant challenge. Incident production data is highly confidential, and likely out of distribution for LLMs without fine-tuning, which can be costly and impractical for large models [8]. In-context examples can serve as an alternative to fine-tuning for domain adaptation, but for agent based RCA, crafting entire reasoning trajectories can be challenging. This is exacerbated by the fact that agents require sophisticated prompting and typically also require fine-tuning [40] or in-context examples [33]. Lastly, RCA poses some unique characteristics that differentiate it from standard NLP tasks. For most NLP tasks, relevant external tools such as web search engines and document retrieval are easy to use in a single step process, and do not require much prior knowledge from the LLM. For RCA, crafting a query for search or retrieval requires much more specialized domain knowledge; many sources of information such as logs, traces, and monitoring services involve querying and processing of tabular data using specialized query languages as well as knowledge of ancillary information (e.g. which database to query). Therefore, while LLM agents offer exceptional abilities that go far beyond prior approaches, it is unclear whether they can be effectively adapted to the RCA task.

In this work, we present an empirical evaluation of an LLM-based agent, ReAct for root cause analysis for cloud incident management. Our goal is to answer two important questions in this regard: 1) Can LLM agents be effective at RCA in the *absence* of fine-tuning? and 2) What are the practical considerations of using LLM agents in real world scenarios? To answer these questions, we first conduct an evaluation of the ReAct agent equipped with retrieval tools on a static dataset, mirroring the evaluation setting by Ahmed et al. [1]. In this setting, the agent does not have access to specialized, team specific, diagnostic services, thereby restricting its abilities. This establishes a lower bound for their performance, and also reflects a practical scenario where agents are incrementally adopted across an organization or company, gradually gaining access to diagnostic services over time. Next, we investigate the use of discussion comments from historical incident reports to augment our retrieval corpus. This serves two purposes; not only do discussion comments add additional context to the incident report, but they also contain records of the diagnostic steps followed by OCEs for past incidents. The latter can potentially be used in lieu of few-shot examples to guide the agent. Lastly, to explore the full potential of agents, we present a case study of a practical implementation of an LLM agent for RCA, fully equipped with team specific diagnostic resources, in collaboration with another team at MicrosoftĊoncretely, we make the following contributions:

- We present the first empirical study on the use of ReAct [40], an LLM agent, for RCA in an out of domain setting on a static dataset of real world production incidents
- We conduct a qualitative analysis of the different success and failure modes of the ReAct in RCA.
- We evaluate the use of discussion comments from historical incidents and its impact on the agent's performance.
- We present a case study of a real world implementation of an LLM-based agent for RCA with a team at a large scale enterprise
- We highlight both the potential of LLM-based agents and the challenges involved in implementing real world systems capable of fully autonomous RCA.

2 BACKGROUND AND RELATED WORK

2.1 Cloud Incident Management and Root Cause Analysis

Production incidents are unplanned events or disruptions in service that adversely affect customers. Outages in service due to production incidents can be extremely costly for enterprises. The complexity of modern software systems renders production incidents inevitable, and *incident management* a key component of the software development life cycle. The life cycle of an incident involves incident detection, triaging, diagnosis and mitigation [1]. While incidents may be reported by customers or automatically detected and triaged using monitoring services, the remaining steps are traditionally conducted by one or more on-call engineers (OCEs). The goal of incident management is to minimize the time between the occurrence of the incident, and its resolution.

2.2 Root Cause Analysis (RCA)

Root Cause Analysis constitutes one of the most time-consuming aspects of the incident management life cycle. When OCEs receive an incident, they systematically perform a series of troubleshooting

Title: SD#1234123412341234 | PRE | SEV A | Specified blob does not exist. | Cloud Services LLC
Description: Customer mentioned that after stopping stream analytics on 09/23 they are getting errors on streaming into <*database product*>[**...**] It was throwing an error "Specified blob does not exist" and "Invalid connection string format. [SessionID: <*uuid*> Found Another error message "Error while Ingesting data to <*database product*>"

Figure 1: Example Incident

steps to identify the root cause. Each troubleshooting step yields previously unknown information, helping the OCE narrow down on the set of plausible root causes. This highlights a key aspect of root cause analysis: the process of collecting additional diagnostic information related to the incident. The incident report describes the symptoms leading to the reporting of the incident, but similar symptoms can emerge from distinct root causes, which might span a diverse set of domains, such as hardware failures, network issues or software bugs. Therefore, OCEs must start the diagnosis process by collecting supplementary data from relevant logs, metrics and other monitoring and diagnostic services. For example, the incident shown in Figure 1 was resolved by checking logs collected from the affected service to identify the sequence of events that lead to the failure encountered by the customer. Another implicit requirement in this process is that OCEs know 1) what additional information needs to be collected, and 2) how to collect this information. This is why even experienced engineers need to have experience with team's services before they can effectively perform RCA. In all, successful RCA requires the following pieces of information: 1) symptoms reported in the incident report, 2) additional diagnostic information, and 3) domain expertise, i.e. what diagnostic information should be collected based on the information, how to collect it and general knowledge about the application domain

The root cause analysis pipeline demonstrates many of the challenges posed for OCEs as well as efforts to automate this procedure. OCEs must have sufficient domain knowledge and familiarity with the affected service to know 1) which supplementary data to collect, 2) how this data must be collected and 3) how to analyze all the available information (including the incident report). Depending on the scale and complexity of the underlying service, this might require OCEs to have several years of experience with the team's services to develop the requisite skill set for effective root cause analysis. Even when OCEs are sufficiently trained, the data collected can be multi-faceted, spanning from structured tabular data to unstructured logs and customer reports. This further complicates data analysis and subsequent hypothesis generation for OCEs. While OCEs can overcome these challenges by leveraging domain expertise and experience, this poses a significant challenge for prior automated approaches, that are unable to collect this supplementary data, let alone analyze it to produce a root cause.

2.3 Automated RCA

Numerous studies have proposed various techniques for automating root cause analysis, such as using machine learning models and deep learning models [32] to identify patterns in event data and determine the underlying causes of incidents. Another important area of research in RCA is the use of anomaly detection models [34], such as statistical, machine learning and deep learning models [10], have been proposed to identify anomalies in system behavior and alert operators in real-time. Studies have proposed various techniques for RCA and triage such as learning a hierarchical monitoring system [21], diagnosing and triaging performance issues [3], and correlating events with time series [16]. In addition, there have been studies exploring the use of structured knowledge mining from various artifacts, such as incident reports and root cause documentation, to mine structured knowledge in software engineering such as troubleshooting guides (TSGs) [11] and there have been efforts to improve TSG quality [28]and make them more effective for incident resolution.

Large Language Models (LLMs) have shown remarkable ability to work with a wide variety of data modalities, including unstructured natural language, tabular data and even images. Recently, Ahmed et al. [1] proposed the use of fine-tuned pretrained LLMs for RCA of cloud incidents. Since incident data is highly confidential, and unlikely to have been observed by pretrained LLMs, fine-tuning is necessary for domain adaptation of vanilla LLMs. In this work, we adopt the RCA task as framed in Ahmed et al. [1]; given an incident report, we want our model to predict a specific root cause. However, unlike the original setting, we exclude the use of fine-tuning or other training approaches for domain adaption. As pointed out by Chen et al. [8], while fine-tuning can be effective, it is also costly and time-consuming, and must be repeated every time the base model gets updated, or services evolve. To address these limitations, Chen et al. [8] introduce RCACopilot, which uses predefined handlers to automatically collect multi-modal diagnostic data relevant to the incident, and an LLM to analyze the collected data and predict a root cause category for the incident that serves to assist OCEs with RCA, without the need for finetuning. Unlike RCACopilot, the ReAct agent presented in our case study can dynamically collect related diagnostic data autonomously, without the need for predefined handlers.

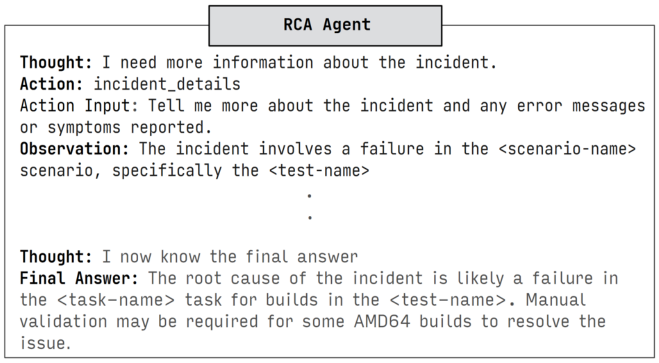
2.4 Augmented LLMs and LLM-Based Agents

A recent development in LM research has been the rise of LMs augmented with the ability to reason and use tools, or *Augmented Language Models (ALMs)* [13, 20, 27]. Augmenting LLMs extends their ability beyond what is possible in a purely language modelling regime. Primarily, these augmentations are either external components that allow the LLM to interact dynamically with its environment for a given problem setting, or prompting techniques that endow the LLM with sophisticated reasoning abilities for complex analytical tasks [37]. For example, LLMs have been augmented with external retrieval databases that can factually ground their predictions, as well as allow them to use information that was not seen in training. Retrieval can also narrow the gap between smaller models and their larger counterparts. LLMs can also be augmented with external components beyond retrieval, such as code interpreters [9] and web search engines. More recently, LLM-based agents combine the external augmentation components with reasoning and planning abilities to allow the LLM to autonomously solve for complex tasks such as sequential decision-making problems [29], knowledge-intensive question answering [35] and self debugging [6].

3 LLM-BASED AGENTS FOR RCA

An LLM agent is an ALM that has the ability to both reason and use tools. In recent years, several different formulations of LLM agents have been proposed [40, 33]. For this work, we base the RCA Agent on the ReAct framework [40]. This framework interleaves reasoning and tool usage steps, combining principles from reasoning-based approaches such as Chain of Thought [38] with tool usage models like Toolformer[12]. ReAct is a natural fit for the RCA task for many reasons: 1) Real-world RCA task has elements of both sequential decision-making (deciding which troubleshooting steps to take) and knowledge-intensive question answering (assessing available diagnostic information to produce a candidate root cause), both of which are supported by ReAct; 2) in an out of distribution setting such as the one we consider, ReActcan quickly adapt to new information since it interleaves reasoning, planning and environment feedback rather than creating a long-horizon plan upfront; and 3) it can easily be augmented with additional components such as reflection [30] and external memory mechanisms [43] which would benefit RCA for incidents requiring a longer diagnostic process.

3.1 Overview


RCA Agent

```
Thought: I need more information about the incident.
Action: incident_details
Action Input: Tell me more about the incident and any error messages
or symptoms reported.
Observation: The incident involves a failure in the <scenario-name>
scenario, specifically the <test-name>
                                .
                                .

Thought: I now know the final answer
Final Answer: The root cause of the incident is likely a failure in
the <task-name> task for builds in the <test-name>. Manual
validation may be required for some AMD64 builds to resolve the
issue.
```

Figure 2: An example of ReAct's reasoning trajectory

Figure 2 shows an example of a sample trajectory produced by a ReAct agent: the agent produces a "thought", or a reasoning step that informs the next "action" it takes. The action space is consists of a fixed set of tools available to the agent. Once the action and it's inputs are specified, the tool is executed and it's outputs are reported back to the agent as an observation. Steps 1-3 repeat for as many times as needed to perform the task at hand. We use the Langchain[5] framework to implement the ReAct agent. Note that the tools used by the agent might also rely on LLMs. To disambiguate, we refer to the LLM performing the ReAct loop as the *planner*. We limit the maximum number of iterations of this loop to 20 due to time and resource constraints.

3.2 Zero-Shot Prompting

While LLM-based agent approaches typically benefit with few-shot examples [40, 33], we use ReAct in a much more challenging setup with a zero-shot prompt. Originally, we set out to craft few-shot examples based on examples from the evaluation set. However, for the setting in RQ1 and RQ2, where we only utilize the incident title and descrption, we found it extremely challenging to come up with reasoning traces grounded in the available information that would arrive at the correct root cause.

3.3 Agent Evaluation

One of the primary benefits of agent-based RCA is their ability to collect external diagnostic information via tools. This is difficult to evaluate without the existence of a simulated environment such as WebArena [44], AlfWorld [31] or WebShop [39]. The main challenge in constructing such an environment is that it is difficult to determine what diagnostic services were used to diagnose a particular incident, since OCEs are not required to report each and every diagnostic step taken. Moreover, the type of diagnostic services used by different teams can vary greatly. Another challenge is that the environment needs to support not only the most optimal troubleshooting trajectory that the agent can take, but a reasonably large subset of other plausible trajectories, i.e. even if we know what diagnostic data is needed to resolve an incident, it does not suffice to only capture this specific data for the environment. Given such an evaluation environment does not currently exist, we evaluate the agent in a restricted setting where we do not assume access to any specialized services, similar to [1]. While this evaluation does not reflect the benefits of the agent's ability to perform autonomous diagnostic steps, it provides us with a lower bound for performance of the agent when specialized tools are unavailable, and allows us to fairly compare it to other ALMs that do not have the ability to query additional diagnostic data. In addition, to demonstrate the agent's ability to interact with diagnostic services, we also present a case study of a prototype implementation of an agent in collaboration with a team at our company. This presents a more realistic evaluation of the agent but at a much smaller scale. The goal of the case study is to examine the benefits and limitations of the agent in a practical environment, and to identify practical considerations for real world adoption based.

3.4 Tools

For the generalized setting, we restrict ourselves to general tools that apply to *all* incidents, regardless of their place of origin.

Incident Details Our investigation of the evaluation dataset revealed that sometimes incident reports contain logs, stacktraces and other diagnostic information. While this information can be noisy and present challenges with context length, it is possible to expose the raw incident description to the LLM via a question-answering tool. The agent can use this tool to answer specific questions about the incident that might get lost during summarization.

Historical Incidents This tool retrieves historical incidents based on the query made by the LLM planner. Based on experiments on our development dataset, we formulate two variants of this tool. The first variant, ReAct BR, uses the target incident title and description, along with a query produced by the agent for retrieval, and simply returns the retrieved documents as an observation without further processing. Since the query here is a passage, we exclusively use the SentenceTransformer retriever for retrieval. The second variant, REACT S+Q, first retrieves a set of historical incidents, and then uses another LLM to answer the planner's query. This two-step process allows the planner to disentangle the retrieval query from the target incident report, and also mitigates instances where the size of retrieved historical incidents might extend beyond the context length of the underlying LLM. We restrict the retrieval tool to retrieve $k = 3$ documents per query, to give the agent the opportunity to create a diverse set of queries while still maintaining an overall budget of 10 retrieved documents for parity with other baselines.

4 RESEARCH QUESTIONS

To evaluate the efficacy of LLM-based Agents in RCA, we ask the following research questions:

RQ1: How effective are LLM-based agents at finding incident root causes when given access to a generalized toolkit? In this setting, we test the efficacy of LLM-based agents at root cause analysis in an out of distribution setting when they are given access only to tools that are independent of specific teams. We equip the agent with a generalized retrieval tool over historical incidents, and a question-answering tool over the raw incident description. We consider various strong ALM baselines that, unlike the agent, are unable to use tools.

RQ2: Do discussion comments help improve LLM based approaches to root cause analysis?

Discussion comments on incident reports contain records of the diagnostic steps taken by OCEs to resolve the incident, and can guide models in performing RCA on future incidents. Here, we aim to investigate whether incorporating these discussion comments into our retrieval corpus of historical incidents impacts the performance of the agent as well as selected baselines from RQ1. To perform this evaluation, we augment incidents in our retrieval corpus with associated discussion comments *post-retrieval*, to ensure that the presence of the comments does not affect retrieval.

RQ3: How effective are LLM agents at RCA when given access to team specific diagnostic tools?

In this research question, we evaluate a real world scenario when an LLM based agent has access to a team specific knowledge base and monitoring service. To conduct this evaluation, we perform a case study with another team's on-call engineers. We package the ReAct agent with these resources into a chat interface, and conduct an in person experiment to see if this agent is able to effectively assist the on call engineer in finding the root cause of a small set of incidents.

5 METHODOLOGY

We describe the methodology used to answer RQ1 and RQ2 in this section. The methodology for RQ3 is described in Section 7.

5.1 Dataset

We collect incident data from our internal incident portal, from 01/01/2020 to 09/30/2021. Our data collection process yielded a total of 107,000 unique incidents, which we split into a train (102,000), evaluation (2000) and test (3000) sets. For this work, we randomly sample 100 incidents from the evaluation set and 500 incidents from the test set to reduce costs, in line with work in NLP [36]. We use the training set is primarily used as the retrieval corpus for our experiments. Like Ahmed et al. [1], we use the incident title and description as the primary sources of information about the incident. For RQ2, we also include discussion comments into the historical corpus. Incident descriptions and root causes do not follow a standard format, and can be quite long. This imposes limitations on the number of historical incidents that can be fit in context when using any kind of retrieval augmented generation. Hence, we use gpt-3.5-turbo to summarize descriptions and root causes. For RQ2, we also summarize discussions comments. Since discussion comments are much longer, we split them into chunks, summarize each individual chunk and recombine them, utilizing the LLM for each step. Note that the summarization process is difficult to evaluate due to a lack of reference summaries, and hence we rely on qualitative analysis and end-to-end evaluation on RCA to iterate on the summarization process.

5.2 Base LLMs

For all of our experiments, we use OpenAI GPT4-8k [22] as the primary language model. GPT-4 is the most powerful model in OpenAI's repository of models, and is one of the few models that can be used to reliably drive an agent in a zero-shot setting. The large context size (8,000 tokens) also enables us to use a larger number of retrieved incidents for our models. For summarization of incidents and discussion comments, we use gpt-3.5-turbo to lower costs.

5.3 Retrievers

We construct a retrieval corpus of historical incidents that encompasses the entire training split of our collected dataset. We consider one dense retriever and one sparse retriever.

Dense Retriever (ST) We use a pretrained Sentence-Bert [24] based encoder (all-mpnet-base-v2) from the associated Sentence-Transformers as our dense retriever and Max Marginal Relevance (MMR) [4] for search.

Sparse Retriever (BM-25) While models that perform a single retrieval step, other models such IR-CoT and the ReAct agent perform multiple retrieval steps with different queries, and can benefit from term based search [36]. We use BM-25 [25] as our sparse retriever.

5.4 Baseline Models

Here, we describe the baselines used for our evaluation in RQ1 and RQ2. We restrict ourselves to ALMs that do not require any fine-tuning.

Retrieval Baseline (RB) Retrieval Augmented Generation (RAG) is an effective strategy to providing domain adaptation for language models without additional training. For our experiments, we create a retrieval database of historical incident reports with known root causes, and use the incoming incident's title and description to retrieve top-k relevant historical incidents. These incidents are then put into the LLM's context as few-shot examples.

Chain of Thought (CoT) Chain of Thought is one of the earlier prompting methodologies developed to enhance the reasoning abilities of LLMs[37]. The idea here is to encourage the model to break the input problem into smaller parts by thinking step by step. For our experiments, we use CoT in a zero-shot setting, by appending a prefix ("Let's think step by step") to the answer prompt.

Interleaving Retrieval - Chain of Thought (IR-CoT) Trivedi et al.[35] show that interleaving vanilla CoT prompting with retrieval improves model performance on complex, multistep reasoning tasks. After every reasoning step the LLM takes, the reasoning step is used to retrieve relevant documents from the retrieval corpus. This is shown to improve performance over using single step retrieval for knowledge intensive question answering tasks.

5.5 Automatic Evaluation Metrics

For evaluating models in the general setting, we use a 3 evaluation metrics based on lexical similarity (BLEU, METEOR, Rouge) and 1 on semantic similarity (BertS). BLEU [23] is a precision based lexical similarity metric that computes the n-gram overlap between model predictions and ground truth references. We use both corpus (C-BLEU) and segment (S-BLEU) level variants. METEOR [2] considers both precision and recall, and uses more sophisticated text processing and scoring systems. rougeL [14] is commonly used to evaluate summarization and is recall based. BERTScore (BertS) [42] measures semantic similarity rather using pretrained BERT models.

5.6 Qualitative Analysis

Table 1: Manual Annotation Criteria

Outcome	Description
Correct	
Precise	Precisely matches reference root cause
Imprecise	Matches reference but misses some details
Hallucination	Matches reference but contains unrelated factual errors
Incorrect	
Hallucination	Contains factual errors in reasoning or prediction
Insufficient Evidence	Refrains from making a prediction
Other	Cause of error unknown
Reasoning Error	Reasoning contains errors
Retrieval Error	Unable to retrieve relevant historical incidents

While automatic metrics can serve as proxies for lexical and semantic similarity, they are not able to accurately measure factual accuracy or establish semantic equivalence. Common failure modes of these metrics include predictions that restate the incident report or highly generic predictions (e.g. "there was a transient network issue") [1], both of which can trivially boost lexical similarity. To better characterize the performance of the LLM agent and other baselines, two authors conduct a qualitative coding on a sample of 100 predictions for three models (300 annotations in total). The labelling is done in iteratively, and the authors engaged in extended discussions to resolve disagreements. We characterize both success and failure modes of these models based on the coding scheme shown in Table 1, adapted from Yao *et al.* [40] and specialized for the RCA task. The adaptations are a superset of the original categories and were made after performing labelling on a smaller sample 20 predictions to distinguish useful scenarios for RCA. Notably, for correct predictions, we differentiate correct predictions that unambiguously match the reference (*Precise*), match the reference semantically but exclude some specifics present in the reference (*Imprecise*), and those that match the root cause semantically but also contain unrelated factual accuracies (*Hallucinations*). The last case commonly manifests as predictions that suggest the execution of post-hoc resolutions actions (e.g. the incident was resolved by restarting the affected cluster) that did not take place. *Imprecise* predictions can be useful for OCEs, whereas factual errors can mislead OCEs. For predictions that don't match the reference root cause, we add two new categories to the ones from [40]. The first, *Insufficient Evidence*, refers to an incorrect prediction that indicates that there isn't enough evidence available to determine the root cause for the incident. The second, *Other*, refers to instances of incorrect predictions that do not have a clearly identifiable cause for error. This is an extension to the label ambiguity category from [40], and now includes other failure cases where the model predicts a plausible specific root cause (unlike *Insufficient Evidence* which is only applied to cases where no specific root cause is indicated), but does not contain obvious reasoning, retrieval, or factual errors. This is often due to the information sparsity of incident reports, especially in cases where the incident report provides details as external links that are inaccessible for the models.

6 RQ1 AND RQ2 RESULTS

Table 2: RCA performance on test set

Model	C-BLEU	S-BLEU	rougeL	METEOR	BertS
RB (k=3)	4.73	4.64	18.48	21.62	0.863
RB (k=6)	5.66	5.56	19.78	23.25	0.865
RB (k=10)	5.97	**5.74**	**20.30**	**24.11**	**0.866**
CoT	**6.31**	5.60	19.91	22.02	0.865
IR-CoT ST	3.91	3.67	16.97	18.50	0.859
IR-CoT BM25	4.61	4.02	17.56	19.94	0.860
ReAct BR	5.53	4.90	17.45	19.23	0.858
ReAct S+Q BM25	5.59	4.73	17.43	18.72	0.857
ReAct S+Q ST	5.27	4.58	17.35	18.60	0.857

6.1 RQ1: How effective are LLM based agents at finding incident root causes when given access to a generalized toolkit?

Table 2 presents the results for our quantitative evaluation based on automatic evaluation metrics. For the Retrieval Baseline model, we see that the number of historical retrieved has a positive impact on performance across the 4 lexical metrics. However, the impact on semantic metrics remains small (< 1). This is likely due to the increase in access to domain specific terms with a larger number of historical incidents. We observed similar trends in experiments on our development set, and consequently we limit $k = 10$ for the remaining models. CoT offers the highest performance on C-BLEU (6.31), followed closely by the RB (k=10)(5.97), Retrieval Baseline (k=6) (5.66), ReAct BR (5.53) and ReAct S+Q BM25 (5.59). However, these differences in performance are well within the margin of error for C-BLEU[19]. Both variants of IR-CoT lag behind, performing more at a similar level as the Retrieval Baseline (k=3) despite retrieving a larger number of historical incidents. For the remaining lexical metrics, RB (k=10) offers the highest levels of performance, closely followed by CoTand Retrieval Baseline (k=6), with the largest difference between these models on METEOR(2.09). These are followed ReAct variants, ReAct BR and ReAct BR, and Retrieval Baseline (k=3). When investigating reasoning logs, we discover that the ReAct variants retrieve a mean of 4 unique historical incidents on average, with an average of 2 lookups per incident. While on each retrieval step they retrieve 3 unique incidents, the historical_incidents tool is stateless, and does not take into consideration documents that have already been retrieved in prior steps, resulting in some duplication. This is a consequence of the limited information available in the incident title and description, making it difficult for the model to crafting separate queries that yield distinct sets of historical incidents. This is likely also why the IR-CoT variants perform poorly on lexical metrics. When we consider semantic similarity, we observe a performance envelope of < 1 across all models. Therefore, neither reasoning nor additional historical incidents drastically change the semantic content of predictions made by these models.

Qualitative Analysis Table 3 shows the results of our qualitative assessment for the RB (k=10), CoT and a variant for ReAct(ReAct S+Q BM25). CoT and RB (k=10) have an accuracy of 39%, followed by ReAct S+Q BM25at 35%. There are 28/97 examples that are solved correctly by all three models. ReAct S+Q BM25 correctly predicts 4 examples that are incorrectly predicted by the other two. When we examine these instances, we discover that in all of these instances, ReAct was able correctly filter out (in its reasoning steps) historical incidents that share some lexical similarity with the target incident report but ultimately are semantically quite different, whereas the other two models incorrectly include them in consideration for their final prediction. CoT and RB (k=10) correctly predict 8 and 9 examples respectively for which ReAct is incorrect. 2 of these instances resulted from reasoning errors by ReAct and the rest were primarily instances where it indicated a lack of evidence (*Insufficient Evidence*) for RCA. Looking more closely, 26% (10/38) of the correct predictions made by the RB (k=10) contain hallucinations, while it is $< 1\%$ for CoT and ReAct S+Q BM25. Similarly, 49% (29/59) of the RB (k=10)'s incorrect predictions are hallucinations, dropping to 18% (11/59) for CoT and 6%(4/63) for ReAct. Overall, ReAct has the highest precision among the 3, but this comes at the cost of lower overall accuracy.

Table 3: Manual Labelling of Success and Failure Cases

	Type	RB (k=10)	CoT	ReAct-BM25
Correct	Imprecise	2	7	5
	Hallucination	10	1	-
	Precise	26	30	29
	All	**38**	**38**	**34**
Incorrect	Hallucination	29	11	4
	Insufficient Evidence	11	19	39
	Other	19	27	8
	Reasoning Error	-	2	10
	Retrieval Error	-	-	2
	All	**59**	**59**	**63**

CoT makes fewer reasoning errors than ReAct. This is likely in part due to the more sophisticated prompting involved with ReAct and the zero-shot setting. We observed some instances of longer reasoning trajectories wherein ReAct would have difficulty maintaining the prompt format. 66% of ReAct's incorrect predictions indicate lack of information to make a root cause prediction (*Insufficient Information*, while this is much less frequent for CoT (32%) and RB (k=10) (18%). Lastly, a notable portion of errors for the RB (k=10) (32%) and CoT (45%) do not have a clear cause for the error (*Other*), while this happens much less for ReAct (12%). Many of these uncategorized errors are predictions that are too generic (e.g. suggesting a non-specific configuration issue), while others are plausible based on historical incidents but incorrect.

Overall, the qualitative analysis indicates that the higher correctness rates of the RB (k=10) come at the cost of factual accuracy, despite the grounding offered by retrieval. CoT offers the same correctness rate with lower rates of hallucination. This clearly demonstrates the benefits of introducing explicit reasoning into an LLM. The ReAct agent also benefits from reasoning, offering the lowest rates of hallucinations for both correct and incorrect predictions, albeit at a slightly lower overall accuracy rate.

RQ1 Takeaways: ReAct agents perform competitively with retrieval and CoT baselines on semantic similarity, while under performing on lexical metrics. Manual labelling reveals that they achieve competitive correctness rates, while providing a substantially lower rate of hallucinations.

6.2 RQ2: Do discussion comments help improve LLM based approaches to root cause analysis?

Table 4 shows the performance of the considered models after incorporating discussions into retrieved historical incidents. In general, incorporating discussions provides mixed results on model performance for lexical metrics across different models. Discussions improve performance on C-BLEU, S-BLEU and rougeL for RB (k=10) but these improvements are modest. On the other hand, it experiences a modest drop in performance for METEOR (< 1). CoT experiences performance degradation for all lexical metrics: C-BLEU (-0.13), S-BLEU (-0.39), and METEOR (-0.7) and rougeL (-1). Unlike CoT both ReAct variants are mostly positively impacted by the inclusion of discussions. ReAct BR shows improvements in performance for rougeL (+0.1) and METEOR (+0.8) but obtains lower C-BLEU (-0.1), and does not show any difference in S-BLEU. Similarly, ReAct S+Q BM25 does not display differences in C-BLEU, S-BLEU or rougeL, but gets a small improvement in METEOR (+0.24). It is important to note that these small differences in metric scores (≤ 1) are likely not be perceivable to human annotators, as has been empirically observed in NLP [19], as well as SE [26]. Lastly, semantic metrics reveal that the incorporation of discussions does not significantly impact the performance of the 4 models in Table 4. Our qualitative observations of a small set of model predictions (20) in the presence vs absence of discussions are in line with these findings. Notably, we do not observe any meaningful differences in the semantic content of the produced root cause between the two scenarios among the predictions that we analyzed.

Table 4: Test set results after incorporating discussions

Model	C-BLEU	S-BLEU	rougeL	METEOR	BertS
RB (κ=10)	**6.65** ↑	**6.01** ↑	**20.8** ↑	**23.81** ↓	**0.867**
CoT	6.18 ↓	5.21 ↓	18.8 ↓	21.32 ↓	0.861
ReAct BR	5.44 ↓	4.91	17.8 ↑	20.04 ↑	0.854
ReAct S+Q BM25	5.52	4.68	17.4	18.96 ↑	0.858

We conjecture that the small observed effect of discussions on RCA performance is due to a combination of 3 factors. Firstly, comments reporting the end result of a diagnostic step constitute a large portion of RCA relevant discussions. While these comments shed light on the troubleshooting steps that lead to incident RCA and resolution, these steps cannot be *replicated* by models in the general setting; models do not have access to the same diagnostic services and resources that were utilized by OCEs to arrive at the conclusions indicated by these discussion comments. Secondly, the sparsity of information present in incident titles and descriptions negatively impact the ability of models to connect information arising from discussions to the target incident. For example, a discussion comment might signal that the presence of a certain symptom indicates a particular root cause, but this symptom might not be reported in the target incident. This is especially true for symptoms that must be elicited using troubleshooting steps. Lastly, discussion threads on incident reports can themselves be quite data sparse, containing lots of administrative content that are not directly useful for RCA (e.g. incident acknowledgement, status updates). While we use length heuristics to remove clearly uninformative comments, comprehensively improving the quality of the comment dataset will require sophisticated techniques (such as LLM based filtering).

RQ2 Takeaways: Incorporating discussion comments into the historical corpus does not clearly improve models' performance on RCA. Depending on the metric considered, it can both improve or degrade performance on lexical metrics.

7 PRACTICAL IMPLEMENTATION OF RCA AGENT: A CASE STUDY

Our evaluation of the ReAct agent in RQ1 and RQ2 does not fully capture the capability of the agent to dynamically plan and collect additional diagnostic data from team specific diagnostic services. Here, we explore these abilities of the agent by conducting a case study with Azure Fundamental Team to shed light on these capabilities.

7.1 Approach

We work with Azure Fundamental Team over a period of 4 weeks, primarily using unstructured discussions. We start by understanding their needs and the challenges they face with regard to RCA, followed by presenting them with the potential benefits and limitations associated with integrating an LLM based agent into their workflow. Next, we identify key diagnostic services used by the team in practice, how these services are used, and iteratively develop tools that can allow the agent to interface with these services. Lastly, we conduct demonstrations of the agent with a small set of incidents in a simple chat interface with the team to collect their feedback.

7.2 Knowledge Base Articles (KBAs)

A common practice in large IT companies is to encode domain knowledge in internal knowledge base articles. In the context of incident management, these articles contain guidelines for how certain types of incidents must be diagnosed and mitigated, as well as key information about how to conduct these operations such as example database queries. At Microsoft, engineers maintain a large number of KBAs for incident management. They help in standardizing operational procedures, facilitating sharing of knowledge across various teams, and onboarding new engineers. Many types of incidents, especially ones triggered by monitoring services, are tagged with relevant KBAs either automatically, or manually during triage. For these incidents, OCEs will have access to relevant KBAs the moment they start the RCA. Incidents that do not have associated KBAs typically require OCEs to spend time searching for and locating relevant KBAs before they can start RCA. We consider both of these scenarios in our case study.

7.3 Agent Development

We replace the generalized tools in the ReAct agent from RQ1 and RQ2 with the following specialized tools that can access team specific diagnostic data:

Database Query Tool: We design and implement a tool that can be used by the agent to query databases and then analyze query results. The database framework used by the team utilizes a custom query language, which is somewhat similar to SQL. The tool design was informed by discussions with the team as well as analysis of several historical incidents experienced by the team. Based on our investigation, we settled on a design that uses two distinct components for this tool: the *Query Execution Engine* and the *Pandas DataFrame Query Engine*. The Query Execution Engine can be used by the agent to query the database. This requires not only the construction of the actual database query, but also knowledge of the cluster on which the database is deployed and the name of the database. This generic design gives the agent flexibility in making queries and also increases re-usability of this tool for other teams that are also using the same database platform. Once a query is successfully executed, the results returned by the database are transformed into a Pandas DataFrame and sent to the Pandas DataFrame Query Engine. The agent can then perform question-answering over the returned table using natural language queries. The Pandas DataFrame Query Engine itself consists of an LLM, which, based on the agent's queries, performs transformations on the DataFrame using the Python Interpreter and then generates a final answer.

KBA Q/A Tool: KBAs often contain critical information that is required to perform RCA, and are one of the most widely used resources for incident management at Microsoft. For example, one of the key pieces of information required to use the Database Query Tool is the cluster address. This information is typically only available to OCEs via KBAs. To incorporate this information into the agent, we expose a question-answering tool over a set of KBAs (14 documents) provided by the team. The tool consists of a vectorstore containing chunks of KBAs, and an LLM which, given a query from the agent, uses knowledge from the retrieved KBA chunks to answer the query. If the incident in question has an associated KBA, we do not use the vectorstore and directly use it to answer questions posed by the planner.

KBA Planning Tool: During preliminary experiments with the team, we noticed that the eager interleaving of thoughts and actions of ReAct can be detrimental to high level planning, i.e. it can sometimes start unsuccessfully carrying out troubleshooting tasks without constructing a high level plan to guide RCA. To mitigate this phenomenon, we introduce a variant of the KBA Q/A tool which is designed to be used specifically for planning. Structurally, it is identical to the Q/A tool, but introducing it explicitly into the action space of the agent encourages it to consistently construct high level plans before taking concrete diagnostic steps.

Human Interaction Tool: Our discussions with the team revealed several scenarios instances where a human-in-the-loop style workflow is necessary for RCA. For example, diagnosing certain types of incidents requires reproducing the error reported in the incident, or manually logging into a cloud device and extracting diagnostic information, which would be difficult for the agent to do. Therefore, it is desirable to have the ability for the OCE to collect such information, and provide it as an observation to the agent. Moreover, our preliminary experiments revealed that the agent struggles to make progress when key information is missing in the KBAs (such as missing cluster address for DB queries), but this information can often easily be provided by the OCE to the agent. Therefore, we add a Human Interaction Tool to allow the agent to request diagnostic information from OCEs, and also add UI enhancements to allow OCEs to interject the agent's action steps, manually verify tool executions and provide explicit feedback to the agent when desired.

7.4 RQ3 Results: How effective are LLM agents at RCA when given access to team specific diagnostic tools?

7.4.1 Challenges faced by OCEs in the team for RCA. Azure Fundamental Teamdevelops and maintains core services within the company's cloud platform, which hosts both internal and external customers that host cloud applications on their platform. While many of the incidents they receive are human reported, also maintain several systems that automatically detect and report error states. They maintain a large number of troubleshooting guides (KBAs) to mitigate the diversity of incident types and associated diagnostic steps. When a KBA exists for a certain type of incident, and the incident is relatively simple, new engineers with limited experience with the team's services are able to effectively perform RCA. However, identifying the right KBA for an incident can take time, and when incidents get more complex, a significant amount (> 1.5 years) of experience is required for RCA.

7.4.2 Real world RCA using ReAct. We started by investigating simple incidents which have a clear KBA article available, and requires a straightforward sequence of diagnostic steps with minimal branching. We use the incident shown in Figure 3 as an illustrative example of incidents of this type. This incident reports that there has been a *setting drift* in a cluster, i.e., a setting is out of sync with the central orchestrating server. This is a type of incident that is automatically reported by monitoring services, which is why the description is empty. Diagnosing this incident can lead to exactly two outcomes: 1) if there are no tenants in the affected cluster, the incident is marked as a false positive and no mitigation is required and 2) if the cluster is hosting tenants, then the OCE must identify the affected clusters (this information isn't present in the incident report) and manually instantiate a job that will rectify the setting drift to mitigate the incident. Identifying the correct outcome requires querying a database to identify affected clusters and analyzing the returned table to determine whether the incident is a false positive, or requires mitigation. Lastly, the incident report includes an associated KBA describing the necessary troubleshooting steps, example database queries as well as key pieces of information such as the database address.

Title: [SettingDrift] Enable[redacted]AppliancePathCreation is drifted
Description: <empty>

Figure 3: Sample Incident

Even though this incident is relatively straightforward for OCEs, it is not possible to identify whether it is a false alarm or not based only on the incident report. This underlines the importance of having access to diagnostic APIs for any automated RCA mechanism. In particular, it is worthwhile to note that even if an automated approach is able to correctly predict the outcome without carrying out the proper diagnostic steps, *the OCE would still have to carry them out to verify the prediction.* When we tested ReAct agent on this incident, it is able to correctly identify case 1 consistently. This involved using the KBA Planning Tool to gather the required troubleshooting steps, adapt and execute the sample query from the KBA, and correctly assess the resulting table. While this series of action is not challenging for OCEs to execute, we stress the fact that ReAct agent has no prior knowledge of the domain, the incident, or the syntax of the database query language. Yet, it is able to leverage the KBA to autonomously complete the RCA process. We observed that the agent would sometimes fail to execute the database query in its first attempt. However, since we surface appropriate error messages to the agent as observations, it was consistently able to rectify these mistakes and complete the troubleshooting process. One engineer expressed that they were "amazed by the tool's capability to automatically discern the right parameters and even rectify mistakes when the parameters are initially incorrect by querying the documents". On the other hand, the second outcome of this incident (case 2), requires an additional filtering step to remove some rows from the table returned by the database query. In our demonstrations with the team, the agent is unable to resolve this error consistently, but engineers were able to use the human-in-the-loop features of the prototype to intervene and fix the error encountered in the filtering step.

We also examined complex incidents from the team that did not have a clear set of troubleshooting steps in a single KBA, i.e. it required combining information from multiple KBAs. The diagnosis steps typically involved a series of database queries. Engineers on Azure Fundamental Teamindicated that these incidents require at least a year of experience with the team's services to effectively diagnose. Here, we observed that while the agent initially produces a plausible high level plan, it was only ever able to successfully execute one or two diagnostic steps before reaching the iteration limit (20). This is primarily due to the difficulty in producing database queries for these incidents, as information is distributed over multiple KBAs, e.g. sample queries and cluster address are not in the same KBA, requiring the agent to query the KBA Q/A tool multiple times before being able to execute a query. While the iteration limit can be extended, it will eventually fill the context. This signals the need for a scalable multi-trial framework, where experience from past trials can be used to guide future trials (e.g. [29, 43, 18].

7.5 Learnings and Practical Considerations

In this section, we distill some key considerations for the implementation of practical LLM agents for RCA based on the case study.

KBAs are critical to real world RCA. . As seen from our findings, KBAs, are critical to performing real world RCA. They contain both specialized domain knowledge and auxiliary facts about the agent's environment (e.g. database addresses, API information) that are required both for OCEs and LLM agents to effectively carry out diagnostic steps. Even experienced OCEs must either refer to KBAs in real-time or have internalized the information present in these KBAs to some degree to perform RCA. While some of this information can also be gleaned from historical incidents, incident reports typically only contain the outcome of diagnosis steps (commonly in discussion comments) rather than operational knowledge of how these steps must be performed. This is also why engineers across the company invest significant time and effort into the construction and maintenance of KBAs.

Tool usage in RCA is non-trivial. While LLMs such as GPT-4 have shown remarkable ability to use tools prevalent in NLP such as retrieval and search, querying of diagnostic services using specialized query languages requires some trial and error. For this reason, we found that it was critical to surface error messages to the agent to provide feedback to the agent in instances of tool failures. One optimization in this regard is to replace LLMs used in tools with smaller models fine-tuned for using a particular type of tool (e.g. a particular database service). These models can then be injected with team specific information (database addresses) in-context for usage within a specific team.

For complicated workflows, experiential learning and multi-trial workflows are necessary. Incidents that require a long and complex sequence of diagnostic steps for RCA typically have a large space of possible action trajectories. This poses significant challenges for the agent. For these incidents, single trial RCA, where we restrict the agent trajectory to 20 steps, is not sufficient. Extending the agent to a multi-trial setting necessitates the use of a reflection [30, 18] or long term memory component to be able to preserve progress across trials, that allows for experiential learning. These mechanisms allow for learning based using natural language as the medium, and present opportunities for building a system where learnings from a specific team's incidents can be stored in a database, and retrieved for performing RCA on future incidents for the team.

Human intervention is necessary to build trust and provide some guardrails for LLMs for critical operations. . There are many diagnostic steps which can be easily carried out by OCEs, but are not accessible as consumable services for the agent. Moreover, when agents struggle with certain parts of the diagnostic process, such as in our study, engineers can easily intervene and correct the agent's trajectory. Therefore, we recommend that agents used in practical incident management scenarios be endowed with capabilities to allow for human interaction, using a combination of explicit tools in the agent's action space, and UI features for the application exposing the agent to users. These capabilities can also be incredibly useful when combined with experiential learning; one can imagine a scenario where an engineer supervises an agent for a small set of team specific incidents, while it builds its repository of experiences, to enable quick domain adaptation for team specific knowledge, and avoid the burden of building a fine-tuning dataset for adaptation purposes.

8 THREATS TO VALIDITY

The evaluation of the agent and other baselines for RCA are conducted on an internal dataset collected at Microsoft, and might not apply to datasets constructed from other organizations. We use a smaller sample (n=500) of our test set to satisfy budget constraints, which might not reflect performance on the larger test set. However, we minimize this threat by using random sampling, and a sample size that has been employed in prior studies. Lastly, the manual annotations used to qualitatively characterize model predictions can potentially introduce bias. We mitigate this by adapting labelling criterion from prior work, and engage in multiple rounds of discussion to converge on particularly ambiguous examples.

9 CONCLUSION & FUTURE WORK

This work provides the first empirical evaluation of an LLM-based agent, ReAct for root cause analysis for cloud incident management. We have shown that in an out of domain, zero-shot setting, ReAct can perform competitively with strong baselines such as retrieval augmented generation and CoT, while offering substantially lower rates of factual inaccuracies. Surprisingly, the use of discussion comments from incident reports does not have a significant impact on the agent's performance, revealing the limitations of performing RCA on a static dataset. Lastly, through our case study, we demonstrate the potential of LLM-agents to autonomously perform RCA in a real world setting when given access to the right tools. The work presented here is a first step in the development of LLM-based agents for practical RCA. A promising direction for future work is the construction of a simulated RCA environment, which would rapidly enhance the development of agent based approaches for RCA. As we continue to explore these avenues of future work, we anticipate that ReAct and similar agents will play a pivotal role in advancing incident management practices and automating complex decision-making processes in the software engineering domain.

REFERENCES

[1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and mitigation steps for cloud incidents using large language models, (Jan. 2023). http://arxiv.org/abs/2301.03797 arXiv: 2301.03797 [cs.SE].

[2] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Association for Computational Linguistics, Ann Arbor, Michigan, (June 2005), 65–72. https://aclanthology.org/W05-0909.

[3] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2019. Decaf: diagnosing and triaging performance issues in large-scale cloud services. *CoRR*, abs/1910.05339. http://arxiv.org/abs/1910.05339 arXiv: 1910.05339.

[4] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval* (SIGIR '98). Association for Computing Machinery, Melbourne, Australia, (Aug. 1998), 335–336. https://doi.org/10.1145/290941.291025.

[5] H Chase. 2022. LangChain. *Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, HP d. O.*

[6] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304. 05128*. http://arxiv.org/abs/2304.05128.

[7] Yinfang Chen, Xudong Sun, Suman Nath, Ze Yang, and Tianyin Xu. 2023. {Push-Button} reliability testing for {Cloud-Backed} applications with rainmaker. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 1701–1716. https://www.usenix.org/conference/nsdi23/presentation/chen-yinfang.

[8] Yinfang Chen et al. 2023. Empowering practical root cause analysis by large language models for cloud incidents, (May 2023). http://arxiv.org/abs/2305.15778 arXiv: 2305.15778 [cs.SE].

[9] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. Proceedings of Machine Learning Research 202, 10764–10799. Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, (Eds.) https://proceedings.mlr.press/v202/gao23f.html.

[10] Tanja Hagemann and Katerina Katsarou. 2021. A systematic review on anomaly detection for cloud computing environments. In *Proceedings of the 2020 3rd Artificial Intelligence and Cloud Computing Conference* (AICCC '20). Association for Computing Machinery, Kyoto, Japan, 83–96. ISBN: 9781450388832. DOI: 10.1145/3442536.3442550.

[11] Jiajun Jiang et al. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2020). Association for Computing Machinery, Virtual Event, USA, 1410–1420. ISBN: 9781450370431. DOI: 10.1145/3368089.3417054.

[12] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising Sequence-to-Sequence pre-training for natural language generation, translation, and comprehension, (Oct. 2019). http://arxiv.org/abs/1910.13461 arXiv: 1910.13461 [cs.CL].

[13] Patrick Lewis et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Adv. Neural Inf. Process. Syst.*, 33, 9459–9474. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

[14] Chin-Yew Lin. 2004. ROUGE: a package for automatic evaluation of summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, (July 2004), 74–81. https://aclanthology.org/W04-1013.

[15] Chang Lou, Cong Chen, Peng Huang, Yingnong Dang, Si Qin, Xinsheng Yang, Xukun Li, Qingwei Lin, and Murali Chintalapati. 2022. {Resin}: a holistic service for dealing with memory leaks in production cloud infrastructure. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 109–125. https://www.usenix.org/conference/osdi22/presentation/lou-resin.

[16] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '14). Association for Computing Machinery, New York, New York, USA, 1583–1592. ISBN: 9781450329569. DOI: 10.1145/2623330.2623374.

[17] Minghua Ma et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings VLDB Endowment*, 13, 8, (Apr. 2020), 1176–1189. https://doi.org/10.14778/3389133.3389136.

[18] Aman Madaan et al. 2023. Self-Refine: iterative refinement with Self-Feedback, (Mar. 2023). http://arxiv.org/abs/2303.17651 arXiv: 2303.17651 [cs.CL].

[19] Nitika Mathur, Timothy Baldwin, and Trevor Cohn. 2020. Tangled up in BLEU: reevaluating the evaluation of automatic machine translation evaluation metrics, (June 2020). http://arxiv.org/abs/2006.06264 arXiv: 2006.06264 [cs.CL].

[20] Grégoire Mialon et al. 2023. Augmented language models: a survey, (Feb. 2023). http://arxiv.org/abs/2302.07842 arXiv: 2302.07842 [cs.CL].

[21] Vinod Nair, Ameya Raul, Shwetabh Khanduja, Vikas Bahirwani, Qihong Shao, Sundararajan Sellamanickam, Sathiya Keerthi, Steve Herbert, and Sudheer Dhulipalla. 2015. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '15). Association for Computing Machinery, Sydney, NSW, Australia, 2029–2038. ISBN: 9781450336642. DOI: 10.1145/2783258.2788624.

[22] OpenAI. 2023. GPT-4 technical report, (Mar. 2023). http://arxiv.org/abs/2303.08774 arXiv: 2303.08774 [cs.CL].

[23] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. https://aclanthology.org/P02-1040.pdf. Accessed: 2023-9-27. (2002). https://aclanthology.org/P02-1040.pdf.

[24] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: sentence embeddings using siamese BERT-Networks, (Aug. 2019). http://arxiv.org/abs/1908.10084 arXiv: 1908.10084 [cs.CL].

[25] Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval*, 3, 4, 333–389. http://dx.doi.org/10.1561/1500000019.

[26] Devjeet Roy, Sarah Fakhoury, and Venera Arnaoudova. 2021. Reassessing automatic evaluation metrics for code summarization tasks. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2021). Association for Computing Machinery, Athens, Greece, (Aug. 2021), 1105–1116. https://doi.org/10.1145/3468264.3468588.

[27] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools, (Feb. 2023). http://arxiv.org/abs/2302.04761 arXiv: 2302.04761 [cs.CL].

[28] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. 2022. Autotsg: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2022). Association for Computing Machinery, Singapore, Singapore, 1477–1488. ISBN: 9781450394130. DOI: 10.1145/3540250.3558958.

[29] Noah Shinn. [n. d.] Reflexion: reflexion: language agents with verbal reinforcement learning. en. (). https://github.com/noahshinn024/reflexion.

[30] Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection, (Mar. 2023). http://arxiv.org/abs/2303.11366 arXiv: 2303.11366 [cs.AI].

[31] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. ALFWorld: aligning text and embodied environments for interactive learning, (Oct. 2020). http://arxiv.org/abs/2010.03768 arXiv: 2010.03768 [cs.CL].

[32] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: a survey. *ACM Comput. Surv.*, 55, 3, Article 59, (Feb. 2022), 39 pages. DOI: 10.1145/3501297.

[33] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2022. LLM-Planner: Few-Shot grounded planning for embodied agents with large language models, (Dec. 2022). http://arxiv.org/abs/2212.04088 arXiv: 2212.04088 [cs.AI].

[34] Mbarka Soualhia and Fetahi Wuhib. 2022. Automated traces-based anomaly detection and root cause analysis in cloud platforms. In *2022 IEEE International Conference on Cloud Engineering (IC2E)*, 253–260. DOI: 10.1109/IC2E55432.2022.00034.

[35] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with Chain-of-Thought reasoning for Knowledge-Intensive Multi-Step questions, (Dec. 2022). http://arxiv.org/abs/2212.10509 arXiv: 2212.10509 [cs.CL].

[36] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with Chain-of-Thought reasoning for Knowledge-Intensive Multi-Step questions, (Dec. 2022). http://arxiv.org/abs/2212.10509 arXiv: 2212.10509 [cs.CL].

[37] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models, (Jan. 2022). http://arxiv.org/abs/2201.11903 arXiv: 2201.11903 [cs.CL].

[38] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models, (Jan. 2022), 24824–24837. S Koyejo, S Mohamed, A Agarwal, D Belgrave, K Cho, and A Oh, (Eds.) https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf arXiv: 2201.11903 [cs.CL].

[39] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: towards scalable real-world web interaction with grounded language agents, (July 2022), 20744–20757. S Koyejo, S Mohamed, A Agarwal, D Belgrave, K Cho, and A Oh, (Eds.) https://proceedings.neurips.cc/paper_files/paper/2022/file/82ad13ec01f9fe44c01cb91814fd7b8c-Paper-Conference.pdf arXiv: 2207.01206 [cs.CL].

[40] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. ReAct: synergizing reasoning and acting in language models, (Oct. 2022). http://arxiv.org/abs/2210.03629 arXiv: 2210.03629 [cs.CL].

[41] Zhengran Zeng et al. 2023. TraceArk: towards actionable performance anomaly alerting for online service systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. (May 2023), 258–269. http://dx.doi.org/10.1109/ICSE-SEIP58684.2023.00029.

[42] T Zhang, V Kishore, F Wu, K Q Weinberger, et al. 2019. Bertscore: evaluating text generation with bert. *arXiv preprint arXiv*. https://arxiv.org/abs/1904.09675.

[43] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. ExpeL: LLM agents are experiential learners, (Aug. 2023). http://arxiv.org/abs/2308.10144 arXiv: 2308.10144 [cs.LG].

[44] Shuyan Zhou et al. 2023. WebArena: a realistic web environment for building autonomous agents, (July 2023). http://arxiv.org/abs/2307.13854 arXiv: 2307.13854 [cs.AI].

Received 2024-02-08; accepted 2024-04-18