мABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture

Wei Zhang[1], Hongcheng Guo[1†], Jian Yang[1†], Yi Zhang[1], Chaoran Yan[1], Zhoujin Tian[1], Hangyuan Ji[1],
Zhoujun Li[1], Tongliang Li[2], Tieqiao Zheng[3], Chao Chen[3], Yi Liang[3], Xu Shi[3],
Liangfan Zheng[3], Bo Zhang[3]

[1]State Key Laboratory of Complex & Critical Software Environment, Beihang University
[2]Computer School, Beijing Information Science and Technology University
[3]Cloudwise Research

{zwpride,hongchengguo,jiaya,zhangyi2021,ycr2345,}@buaa.edu.cn;deritt7@gmail.com;{jhy_1,lizj}@buaa.edu.cn;
tonyliangli@bistu.edu.cn;ztieqiao2006@126.com;{clark.chen,kim.liang,tim.shi,leven.zheng,bowen.zhang}@cloudwise.com;

ABSTRACT

The escalating complexity of micro-services architecture in cloud-native technologies poses significant challenges for maintaining system stability and efficiency. To conduct root cause analysis (RCA) and resolution of alert events, we propose a pioneering framework, **m**ulti-**A**gent **B**lockchain-inspired **C**ollaboration for root cause analysis in micro-services architecture (мABC), to revolutionize the AI for IT operations (AIOps) domain, where multiple agents based on the powerful large language models (LLMs) perform blockchain-inspired voting to reach a final agreement following a standardized process for processing tasks and queries provided by *Agent Workflow*. Specifically, seven specialized agents derived from *Agent Workflow* each provide valuable insights towards root cause analysis based on their expertise and the intrinsic software knowledge of LLMs collaborating within a decentralized chain. To avoid potential instability issues in LLMs and fully leverage the transparent and egalitarian advantages inherent in a decentralized structure, мABC adopts a decision-making process inspired by blockchain governance principles while considering the contribution index and expertise index of each agent. Experimental results on the public benchmark AIOps challenge dataset and our created train-ticket dataset demonstrate superior performance in accurately identifying root causes and formulating effective solutions, compared to previous strong baselines. The ablation study further highlights the significance of each component within мABC, with *Agent Workflow*, multi-agent, and blockchain-inspired voting being crucial for achieving optimal performance. мABC offers a comprehensive automated root cause analysis and resolution in micro-services architecture and achieves a significant improvement in the AIOps domain compared to existing baselines[1].

KEYWORDS

Root Cause Analysis, Micro-services Architecture, Large Language Models, Multi-Agent, Blockchain-Inspired Voting

1 INTRODUCTION

Micro-services architecture (MSA) decomposes a large application into a series of tiny and independent service nodes, each focusing on a specific business function and interacting with others
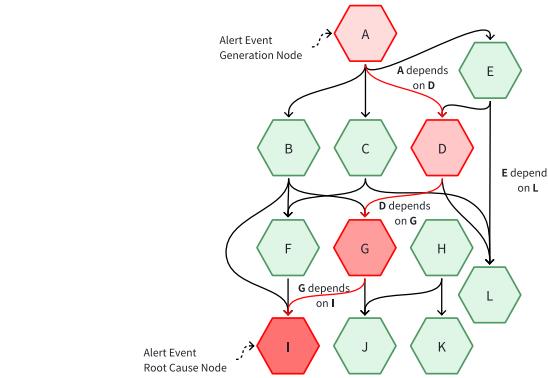

Figure 1: Example of root cause analysis in micro-services architecture (alert event arises on node *A* while alert event root cause node is *I* with fault propagation path $I{\rightarrow}G{\rightarrow}D{\rightarrow}A$.

through lightweight communication mechanisms [1–10]. MAS as the backbone of scalable and resilient systems in cloud-native technologies introduces significant challenges in managing distributed systems [11, 12]. The lifecycle of maintaining system stability in the micro-services environment is comprised of metric monitoring, alert events, root cause analysis (RCA), and resolution [13–15]. Compared with traditional architectures only containing one central service, root cause analysis and resolution in distributed deployment practices of micro-services architecture has become extremely difficult as faults continue to propagate between service nodes and alarm events become increasingly complex [16, 17].

Root cause analysis (RCA) has been a popular research interest in the AIOps domain, including both invocation-based and trace-based approaches [5, 11, 13, 18]. In Figure 1, alert event arises on **A**, while alert event root cause node is **I** with fault propagation path I→G→D→A. RCA identifies the root cause of faults, **I** here, by tracing back to the origin node and even further analyzing metrics. Unlike invocation-based methods that focus on neighboring micro-services, trace-based approaches consider all micro-services in the trace. However, existing methods such as MicroScope [8], TraceAnomaly [16], and MEPFL [17] are unable to handle cyclic dependencies well and rely heavily on data supervision with high fault coverage. The rapid advancement of large language models (LLMs) like GPT [19] and their integration with multi-agent has

significantly improved their application in AIOps for root cause analysis [20–26]. These developments have enabled models to exhibit remarkable analytical and problem-solving capabilities, essential for identifying and addressing issues in complex micro-services architecture [27–30]. Although RCACopilot [29], RCAgent [27], and D-Bot [30] have improved RCA tools with event matching, information aggregation, domain knowledge, they struggle with cross-node failure analysis in complex micro-service architectures.

For this complex scenario in micro-services architectures, we introduce MABC, a groundbreaking framework designed to revolutionize root cause analysis. By harnessing the power of Large Language Models (LLMs) within multi-agent blockchain-inspired collaboration, MABC brings a new level of intelligence and automation to AIOps. MABC can analyze a wide range of data and navigate through node dependencies to avoid the limitations of directed acyclic graphs, and their extensive software knowledge eliminates the need for high coverage of fault types, unlike supervised learning models. Specifically, *Agent Workflow* standardizes task processing flow through task difficulty and dynamic context perception. Overall pipeline encapsulates the flow from alert inception to root cause analysis within MABC. Step 1: An alert event arises due to access function blockages or monitoring system alarms in a micro-services architecture. Step 2: *Alert Receiver* ($\mathcal{A}_1$) forwards the alert event with the highest priority for the root cause analysis after a priority selection. Step 3: *Process Scheduler* ($\mathcal{A}_2$) divides unfinished root cause analyses into sub-tasks, handled by *Data Detective* ($\mathcal{A}_3$), *Dependency Explorer* ($\mathcal{A}_4$), *Probability Oracle* ($\mathcal{A}_5$), and *Fault Mapper* ($\mathcal{A}_6$) for various requests. Step 4: *Solution Engineer* ($\mathcal{A}_7$) develops resolutions for the root cause referencing previous successful cases. Seven specialized agents derived from *Agent Workflow* each provide valuable insights towards root cause analysis based on their expertise and the intrinsic software knowledge of LLMs collaborating within a decentralized *Agent Chain*. To avoid potential instability issues in LLMs and fully leverage the transparent and egalitarian advantages inherent in a decentralized structure, MABC adopts a decision-making process inspired by blockchain governance principles while considering the contribution index and expertise index of each agent on $\{\mathcal{A}_i\}_{i=1}^{7}$.

Experimental results on the public benchmark AIOps challenge dataset and our created train-ticket dataset demonstrate superior performance in accurately identifying root causes and formulating effective solutions, compared to previous strong baselines. The ablation study further highlights the significance of each component within MABC, with *Agent Workflow*, multi-agent, and blockchain-inspired voting being crucial for achieving optimal performance. Generally, the main contributions of this work are listed as follows:

- **Advancing Exploration of LLM and multi-agent in RCA**: The proposed framework, MABC driven by LLM and multi-agent collaboration, demonstrates unparalleled accuracy in RCA and resolution on alert events in a micro-services architecture.
- **Blockchain-Inspired Voting**: The integration of blockchain-inspired voting in MABC brings a decentralized and democratic way to assess agent responses, improving the accuracy and reliability of root cause analysis through collective intelligence and transparent decisions.

- **Creating Train-Ticket Benchmark**: The development of the Train-Ticket benchmark, an open-source evaluation dataset designed specifically for RCA in complex micro-services architectures, provides a comprehensive resource for the research community to develop, test, and benchmark RCA resolutions, facilitating advancements in RCA.

2 RELATED WORK

2.1 Root Cause Analysis in Micro-Services Architecture

Root cause analysis in large systems, especially within micro-services architecture, has become a critical area of focus in AIOps [1–4, 11, 31, 32]. The intricacies of RCA tasks particularly focus on log, metric, and trace, and previous studies have proposed methods for root cause analysis using any of these data sources [13–15, 33]. There are some methods to identify failure patterns [11, 12] or to explore in service dependency graphs according to metrics and traces[10, 34]. Those related to NLP for log analysis and anomaly detection, underline the importance of advanced methodologies[35–39]. These include leveraging machine learning for log analysis [40, 41] and using large language models (LLMs) for enhanced RCA performance[38, 39, 42].

2.2 LLM in Micro-Services Architecture

The accelerated advancement in language modeling [19–21, 23, 26, 28, 38, 43–49], notably through Transformer-based architectures and large language models (LLMs) like GPT-4 [19] and PaLM [50], has revolutionized various domains of natural language processing and laid the groundwork for their deployment in complex micro-services architecture [22, 49, 51–58]. The integration of LLMs with external tools and APIs further amplifies their functionality, enabling more effective engagement and manipulation of environments, particularly in cloud Root Cause Analysis (RCA) [59–61], where they enhance precision in log analysis and anomaly detection. The capabilities exhibited by LLMs have sparked interest in their application as the core intelligence within autonomous multi-agent [62]. Researchers have leveraged these models to develop agents capable of understanding and interacting with their environment in a human-like manner [22, 56]. The exploration of autonomous agents in tasks ranging from toy examples to real-world cloud RCA underscores the versatility and potential of LLM-based systems in dynamic environments [63, 64]. This evolution underscores a paradigm shift towards leveraging LLMs in micro-services architecture for enhanced autonomy, intelligence, and efficiency [29, 31, 65]. LLM has brought unprecedented vitality and progress to complex micro-service architecture. A series of tasks have emerged, including database diagnosis, event processing, root cause analysis [27, 30, 65, 66], etc. End-to-end intelligent operation and maintenance is booming.

3 METHODOLOGY

3.1 Overview

In this section, we provide a comprehensive overview of MABC, specifically engineered to pinpoint the root causes of alert events in a complex micro-services architecture. Illustrated in Figure 2, MABC
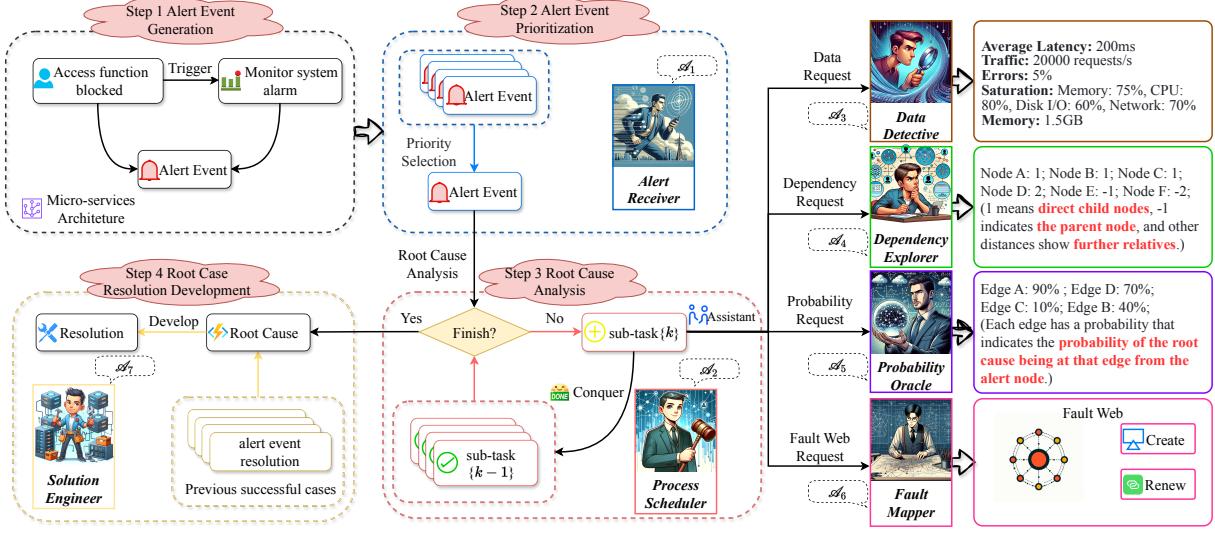
Figure 2: Overview of MABC. Overall pipeline encapsulates the flow from alert inception to root cause analysis within MABC. Step 1: An alert event arises due to access function blockages or monitoring system alarms in a micro-services architecture. Step 2: *Alert Receiver* ($\mathscr{A}_1$) forwards the alert event with the highest priority for the root cause analysis after a priority selection. Step 3: *Process Scheduler* ($\mathscr{A}_2$) divides unfinished root cause analyses into sub-tasks, handled by *Data Detective* ($\mathscr{A}_3$), *Dependency Explorer* ($\mathscr{A}_4$), *Probability Oracle* ($\mathscr{A}_5$), and *Fault Mapper* ($\mathscr{A}_6$) for various requests. Step 4: *Solution Engineer* ($\mathscr{A}_7$) develops resolutions for the root cause referencing previous successful cases.
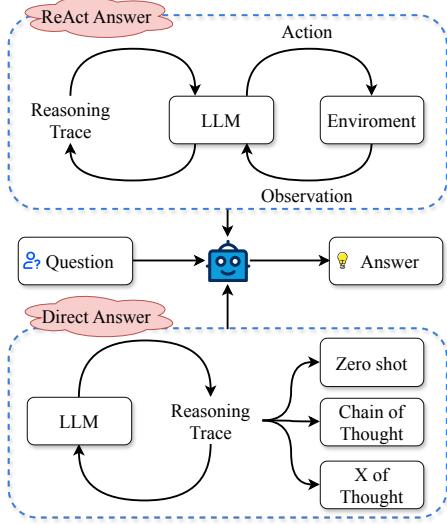

Figure 3: Two distinct workflows of agent. ReAct answer involves an iterative cycle of thought, action, and observation until a satisfactory answer is reached, while responses are directly formulated based on the prompt provided following the direct answer.

introduces seven agents: *Alert Receiver*, *Process Scheduler*, *Data Detective*, *Dependency Explorer*, *Probability Oracle*, *Fault Mapper*, and *Solution Engineer*. These agents collaborate transparently and equally, invoking each other to address alert events in the MABC pipeline. In micro-services architecture, alert events can arise from user-side blocked function access and monitoring system alarms,

such as increased login response times and network latency in the login node.

3.2 Agent Workflow

Agent Workflow, foundational to the framework, enables all agents to complete their tasks effectively, adhering to a prescribed methodology. The differences among agents within this framework are limited to the types of questions they address and the tools they utilize. In Figure 3, *Agent Workflow* assesses the complexity of each question to determine the most appropriate response method. For questions that require real-time data or additional information, *Agent Workflow* activates the ReAct answer workflow, which involves an iterative cycle of thought, action, and observation until a satisfactory answer is reached. Conversely, when no external tools are necessary, *Agent Workflow* defaults to the direct answer workflow, where responses are directly formulated based on the prompt provided.

3.3 Multi-Agent

In this section, we provide a detailed and thorough introduction of agents in MABC. *Alert Receiver* quickly prioritizes incoming alerts. *Process Scheduler* smoothly divides tasks and asks for assistance from other agents. *Data Detective*, *Dependency Explorer Probability Oracle* and *Fault Mapper* gather data, analyze dependencies, assess fault probabilities, visualize fault networks. Finally, *Solution Engineer* analyzes root causes and develops resolutions.

3.3.1 Alert Receiver. In Figure 2, *Alert Receiver* plays a crucial role in the pipeline. The responsibility of *Alert Receiver* is to sort the received alert events based on the time, urgency, and scope of impact. After determining the priority of the alert events, *Alert*

Receiver dispatches the most urgent and widely impacting alert events to *Process scheduler* further processing following the pipeline.

3.3.2 Process Scheduler. In Figure 2, *Process Scheduler* orchestrates various sub-tasks to efficiently resolve alert events. When an alert arrives from *Alert Receiver*, *Process Scheduler* kick-starts a comprehensive process that includes data gathering, fault web construction, dependency analysis, and probability scoring, engaging with specialized agents for each task. These agents, *Data Detective* for data collection, *Dependency Explorer* for querying node dependencies, *Probability Oracle* for assessing node probabilities, and *Fault Mapper* for constructing fault impact, provide critical insights that are then funneled to *Solution Engineer*. *Solution Engineer* conducts root cause analysis and devises solutions. Throughout this process, *Process Scheduler* checks whether the root cause has been identified after each sub-task; if not, it iterates, generating new sub-tasks and soliciting further assistance from the agents until the root cause analysis is finalized. Ultimately, *Process Scheduler* delivers the root cause, an updated fault web, and a resolution strategy, thereby concluding the alert event handling process and setting the stage for the next alert event.

3.3.3 Data Detective. In Figure 2, *Data Detective* plays a pivotal role in the pipeline, tasked with collecting data from designated nodes within a specific time window as dictated by *Process Scheduler*. It gathers comprehensive information, including but not limited to average latency, traffic volume, error rates, resource saturation levels, and concurrent user counts. This wealth of data enables MABC to accurately identify the root cause of issues, leading to the development of effective repair strategies or mitigation actions. To facilitate thorough analysis and maximize the informational value within constrained context windows, *Data Detective* employs a *Data Collection Tool* and a *Data Analysis Tool*. These tools are engineered to exclude non-essential data and apply fuzzy matching based on critical parameters such as nodes and time. *Data Analysis Tool* further processes this data into easily comprehensible charts and reports, thereby democratizing data understanding and decision-making across MABC. This design strategy ensures that LLMs are shielded from the complexities of diverse data sets, simplifying the task for *Data Detective* and eliminating inefficiencies in data exploration and analysis.

3.3.4 Dependency Explorer. *Dependency Explorer* is an expert within the pipeline for analyzing the dependencies among internal nodes of the micro-services architecture. In Figure 2, upon receiving an alert event that requires further analysis from *Process Scheduler*, *Dependency Request* is sent to *Dependency Explorer*, containing information about the specific node and the time of the alert. Utilizing a query tool, in conjunction with global topology and calls within the time window, *Dependency Explorer* precisely identifies the direct and indirect dependent nodes for the specific node and returns to *Process Scheduler*, which is vital for identifying fault path, marking impacted nodes, and setting the stage for further root analysis and resolution.

3.3.5 Probability Oracle. *Probability Oracle* evaluates the probability of fault across different nodes within a micro-services architecture. Receiving requests from *Process Scheduler* and analyzing the specified list of nodes to determine the failure probability of each node, which involves checking whether each node is accessible. For inaccessible nodes, a high failure probability is assigned by default. For accessible nodes, the failure probability is evaluated based on performance metrics such as response time, error rate, or resource utilization using a computational model. By analyzing the correlation of data between nodes, for example, finding that the response time of a node is positively correlated with the error rate of another node (Pearson correlation coefficient close to 1), it indicates that the performance degradation of the node may be related to the failure of another node. Based on the results of the correlation analysis, the failure probabilities are adjusted accordingly by increasing the probability of highly correlated nodes and decreasing the probability of other nodes. Finally, *Probability Oracle* sends these failure probabilities back to *Process Scheduler*, providing crucial data for updating the failure web, root cause analysis, and resolution development.

3.3.6 Fault Mapper. *Fault Mapper* plays a role in visualizing and updating data within the pipeline. *Fault Mapper* receives fault probability information from *Probability Oracle* and updates *Fault Web* accordingly. In Figure 2, when *Fault Web* needs to be updated, *Process Scheduler* issues a *Fault Web Request*, which includes nodes and their corresponding fault probabilities. *Fault Mapper* creates or renews *Fault Web* based on this information to visually represent the fault probabilities between different nodes. *Fault Web* not only displays the alert source node but also depicts other related nodes and the fault probabilities of their connecting edges. *Fault Mapper* ensures that *Process Scheduler* can make decisions based on the most up-to-date information, thereby guiding *Solution Engineer* to develop appropriate resolutions.

3.3.7 Solution Engineer. In Figure 2, *Solution Engineer* is responsible for the final root cause analysis and development of solutions in the pipeline. *Solution Engineer* receives *Root Cause Analysis and Solution Requests* from *Process Scheduler* and then decides on the course of action based on the available node data. If node data is available, *Solution Engineer* conducts a metric-level analysis by using historical data and performance metrics. If there is no accessible node data, *Solution Engineer* performs a node-level analysis by examining the topology of micro-services architecture to identify the impact of the fault. *Solution Engineer* references previous successful cases, like in Table 1, to guide the development of the current solution and returns this information for further action and the conclusion of the process ensuring that the proposed resolution is practical and effective.

3.4 Blockchain-Inspired Voting

3.4.1 Blockchain Communication. Although LLM has rapidly developed, it is still sensitive to generating content that may be fluent but lacks authenticity. To enhance the correctness of each answer, we have designed *blockchain-inspired voting* as a reflection for any answer to any question from any agent. The agents in the MABC are transparent and equal to each other, despite their different responsibilities, and compose a decentralized structure *Agent Chain*. Blockchain, a decentralized structure, follows governance guidelines divided into on-chain and off-chain governance. On-chain

Table 1: Examples of Alert Events

Alert Events	Description
Tablespace High Utilization	Indicates extensive data occupation in tablespace, potentially degrading database performance.
Database Connectivity Fault	Signifies possible connection issues due to excessive connections, impacting response and transactions.
CPU Resource Insufficiency	High session average CPU time suggest significant CPU occupation, risking performance and stability.
Memory Overflow	Shows memory usage exceeding safe limits, risking performance degradation or crashes.
Disk IO Performance Fault	Abnormal increase in physical read rates, indicating potential disk IO issues.

governance is more democratic and transparent, allowing participants to trust each other and leaving decision-making power in the hands of decentralized entities. *Agent Chain* in MABC follows the decisions of on-chain governance to better leverage the advantages of decentralization.

3.4.2 Voting Weights. Voting weight is determined by $w_c * w_e$, where *contribution index* (w_c) reflects a level of activity and effectiveness in proposal contributions and voting for the agent, while *expertise index* (w_e) is determined by the professional contributions of the agent in a specific field and the quality of their actions.

To better encapsulate the dynamics of the contribution and expertise indices with an emphasis on fairness and continuous engagement, *contribution index* w_c is presented as follows:

$$w_c = \min\left(w_c \cdot (1 - \delta) + \Delta w_c,\ w_{c_{\max}}\right) \quad (1)$$
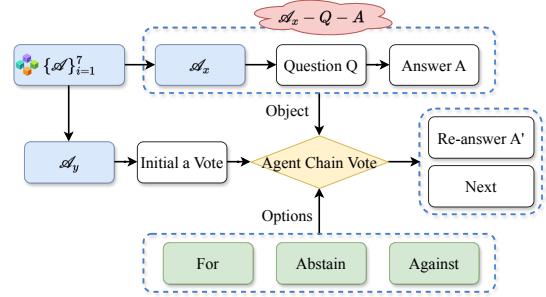
where w_c denotes *contribution index*, initially set to 1.0. δ represents a random decay rate, ranging from 0 to 0.03, applied after each voting event to encourage ongoing contribution and prevent the concentration of power. Δw_c is the increase 0.1 in *contribution index* resulting from active participation in voting (excluding abstentions) and proposal submission. $w_{c_{\max}}$ is the upper limit for both *contribution index* and *expertise index*, set to 1.5 to ensure fairness.

To ensure fairness within the system and encourage informed decision-making, *expertise index* w_e is governed by the following formula:

$$w_e = \min\left(w_e + \Delta w_e,\ w_{e_{\max}}\right) \quad (2)$$

where w_e represents *expertise index*, which does not undergo automatic decay, reflecting the accumulated expertise of an Agent over time. Δw_e is the change in *expertise index*, which increases by 0.01 if the voting choice of agent aligns with the final outcome and decreases by 0.01 otherwise. This mechanism rewards accurate predictions and penalizes inaccuracies. $w_{e_{\max}}$ is the upper limit for *expertise index*, set to 1.5 to maintain fairness by preventing disproportionate influence from highly experienced agents.

3.4.3 Voting Process. On *Agent Chain*, every agent is entitled to participate in voting. The voting process works in Figure 4: When $\mathscr{A}_x \in \{\mathscr{A}_i\}_{i=1}^7$ gets an answer A for question Q, all agents on the chain will examine A and face the choice of whether to initiate a vote on $X - Q - A$. If no Agent initiates a vote, the answer is accepted. If $\mathscr{A}_y \in \{\mathscr{A}_i\}_{i=1}^7$ requires a vote, all agents on the agent chain will vote on $\mathscr{A}_y - \mathscr{A}_x - Q - A$, with the voting options being **For**, **Abstain**, and **Against**. If the vote passes, $\mathscr{A}_x$ will re-answer Question Q to generate a new Answer A'.

Figure 4: Vote process on *Agent Chain*

3.4.4 Voting Outcome Determination. The support rate (s) and the participation rate (p) are calculated as follows:

$$s = \frac{\sum_{i=1}^n \mathbf{1}(w_i)}{\sum_{i=1}^n w_i} \quad (3)$$

$$p = \frac{\sum_{i=1}^n \mathbf{1}'(w_i)}{\sum_{i=1}^n w_i} \quad (4)$$

where, n is the total number of agents to vote on *Agent Chain*, vote_i is the vote of the i-th agent, and w_i is the weight of the vote of i-th agent. A proposal is considered passed if $s \geq \alpha\ \&\ p \geq \beta$, where α and β are predefined threshold percentages. The indicator function $\mathbf{1}(\cdot)$ outputs w_i if i-th agent votes **For** and 0 if i-th agent votes **Abstain**, and **Against**. The indicator function $\mathbf{1}'(\cdot)$ outputs w_i if i-th agent votes **For** and **Against**, 0 if i-th agent votes **Abstain**.

4 EXPERIMENTS

4.1 Datasets

The experiments are performed on two datasets, which are described in detail.

4.1.1 Train-Ticket Dataset. Our dataset is built on Train-Ticket [67, 68], an open-source micro-service benchmark system developed by the CodeWisdom team at Fudan University following industry micro-service practices. As shown in Figure 5, Train-Ticket is a train booking system deployed on Kubernetes and integrated with monitoring and analysis tools. It comprises 41 micro-services that provide typical high-concurrency train ticket booking functions, such as ticket query, ticket reservation, payment, ticket change, and user notification.

Table 2: Train-Ticket Process Descriptions

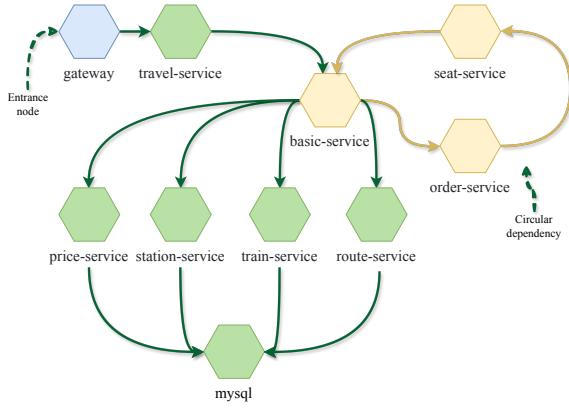| Process	Description
Admin Operations	Admin login, site, and route addition, train information addition, user and contact addition, multiple queries (routes, trains, sites, etc.), updates, deletions, repeat queries.
Normal Flow	User registration and login, ticket availability search, ticket booking, order status refresh, order payment, and ticket check-in.
Re-book Flow	Registration and login, availability search, booking, latest order status refresh, re-booking, new order payment (if applicable), check-in.
Re-Book Fail Flow	Registration and login, availability search and booking, order status refresh, successful first re-booking, order payment, failed second re-booking attempt.
Search Fail to Add	User registration and login, failed ticket search (due to missing stations), admin adds missing info, ticket research and booking, latest order status refresh.
Consign Preserve	User registration and login, ticket search and booking, order status refresh, order payment, luggage consignment addition, and check-in.
Preserve Successfully	User registration and login, ticket availability search and booking, order status refresh, payment, and check-in.

Figure 5: An example of Train-Ticket architecture (query remaining tickets). *basic service* relies on *seat service, order service,* and others. A circular dependency of *basic-service→order-service→seat service→basic-service* brings a new challenge for root cause analysis.

Table 3: Types of Faults Injected in the Experiment

Fault Category	Fault Case Examples
Network	High packet loss, high retransmission rates, DNS failures, bandwidth saturation, high TCP connection setup delays
Storage	High I/O latency
CPU	High CPU usage by the code itself, CPU contention from other processes
Memory	High frequency of FULL GC, memory contention from other processes
Code	Exceptions thrown by the code leading to error codes, HTTP requests, returning error codes

To simulate the operation of the actual system, we have designed 7 processes and 100 virtual users. Table 2 illustrates each process implementing a specific sequence of operations by calling different combinations of functions. These functions range from user registration, login, querying, booking, changing tickets, and adding extra services such as baggage delivery. This covers a variety of common operation scenarios in train ticket management systems. Each user will randomly select a process to execute to access the system.

Using ChaosBlade[69], we introduce faults into the Train-Ticket System following Table 3. ChaosBlade is an experimental injection tool open-sourced by Alibaba. It follows the principles of chaos engineering and experimental models. We thus constructed a dataset of 233,111 call chains consisting of 800,656 call spans embedded in 112 time periods and 53 nodes, including 900 alert events directly caused by the alert nodes and 294 alert events induced by external nodes.

Table 4: Summary of Alert Events in 2020 International AIOps Challenge Dataset

Alert Node	Alert Events Type	Count
os_021	CPU	8434
docker_006	Database Connectivity	130
docker_006	Database Local Method	7174
os_021	Operate System	6352
docker_008	Database Connectivity	233
docker_008	Database Local Method	7552
docker_005	Database Connectivity	211
docker_005	Database Local Method	9684
os_022	CPU	15099
docker_001	Network	2
os_022	Operate System	109
docker_004	Network	124

4.1.2 2020 International AIOps Challenge Dataset. Similar to *Train-Ticket Dataset* mentioned above, *2020 AIOps International Challenge Dataset* aims to discover alert events and their root causes in micro-service applications, such as cloud platform services, which include containers, service meshes, micro-service, and variable infrastructures. The dataset includes 14 days of system logs totaling 145,907,050 entries.

In Table 4, the types of alert events mainly include container CPU utilization, container memory utilization, database connection limit, database close, host network delay, and container network loss. All types of alert events are distributed across various nodes of the system.

4.2 Evaluation Metrics

To objectively evaluate the effectiveness of MABC in identifying and handling anomalies in complex micro-service systems, we adopt two complementary metrics: *Root Cause Result Accuracy (RA)* and *Root Cause Path Accuracy (PA).*

Table 5: Main Results On Train-Ticket Dataset and AIOps challenge Dataset

Model	Base	Train-Ticket Dataset			AIOps Challenge Dataset			Average
		RA	PA	Average	RA	PA	Average	
Decision Tree [70]	-	31.8	29.7	30.8	23.3	21.7	22.5	26.6
ReAct [26]	GPT-3.5-Turbo	26.8	21.8	24.3	20.1	17.7	18.9	21.6
ReAct [26]	GPT-4-Turbo	32.0	27.9	30.0	26.5	23.4	25.0	27.5
MABC	GPT-3.5-Turbo	58.1	52.8	55.5	51.1	46.7	48.9	52.2
MABC	GPT-4-Turbo	**72.4**	**66.2**	**69.3**	**63.5**	**57.3**	**60.4**	**64.9**

4.2.1 Root Cause Result Accuracy (RA). Following the previous work [30, 33], we use result accuracy (RA) to quantify the precision of MABC in finding the root cause.

$$RA = \frac{A_c - \sigma \cdot A_i}{A_t} \quad (5)$$

where A_c denotes the number of correct causes, A_t denotes the total number of causes, A_i denotes the number of wrongly detected causes, and σ is a hyper-parameter with 0.1 as the default value because we recognize *redundant causes is less harmful than missing causes.* Therefore, we limit the identification to a maximum of 4 root causes for an anomaly.

4.2.2 Root Cause Path Accuracy (PA). To assess the accuracy of the inference path between alert nodes and root cause nodes identified by MABC and how accurately these inferred paths align with the actual root cause paths in the system, we use the root cause path accuracy (PA) metric, which aims to measure the effectiveness of MABC in tracing the correct path from the symptoms (alerts) back to the root causes. The formula for PA similar to RA focuses on path accuracy as:

$$PA = \frac{P_c - \tau \cdot P_i}{P_t} \quad (6)$$

where P_c denotes the number of correctly identified paths leading to the root cause, P_t is the total number of actual root cause paths present, P_i denotes the number of incorrectly inferred paths, which do not align with the actual root cause paths, and τ is a hyper-parameter designed to penalize the inaccuracies in path inference, with a default value of 0.2, reflecting the understanding that inaccurately inferred paths are less detrimental than completely missing the correct paths, but there is a stronger emphasis on precision due to the potential complexity and relevance of paths.

4.3 Implementation and Configuration

We implement MABC on Ubuntu 22.04, equipped with an Intel (R) Xeon (R) Gold 6348 CPU @ 2.60GHz, four NVIDIA A100 GPUs (80 GB), and 528 GB of memory. The software setup includes NVIDIA-SMI version 535.104.05 and CUDA version 12.3. In all experiments, support threshold α and participation threshold β are set to 0.5.

4.4 Main Results

In Table 5, we carefully conduct experiments to evaluate MABC against other baseline models on two distinct datasets: train-ticket dataset and AIOps challenge dataset. The experiments are designed to assess the efficacy of the models in root cause analysis and

decision-making tasks within the AIOps domain based on RA and PA, along with their average performance. The results from the table indicate that MABC, particularly when utilizing GPT-4-Turbo, significantly outperforms the other models on both datasets, achieving the highest scores in all evaluated metrics. This superior performance underscores the effectiveness of MABC in handling complex AIOps tasks. The GPT-3.5-Turbo variant of MABC also demonstrates strong results, although it does not reach the same heights as its GPT-4-Turbo counterpart. ReAct [26], despite being grounded in advanced GPT architectures, does not match the performance of MABC, suggesting that the methodology of latter is better suited for the demands of AIOps. The inclusion of the Decision Tree [70] serves as a benchmark, illustrating the advancements and advantages of neural network-based solutions over traditional methods in the field. Overall, the table summarizes the comparative success of MABC in advancing automated root cause analysis in AIOps, with its state-of-the-art GPT-4-Turbo variant leading the way.

Table 6: Decision Efficiency Evaluation

Model	Base	Train-Ticket Dataset		AIOps Challenge Dataset	
		PR	APL	PR	APL
Decision Tree [70]	-	62.4	12.1	53.8	13.4
ReAct [26]	GPT-3.5-Turbo	41.7	15.9	38.0	16.2
ReAct [26]	GPT-4-Turbo	47.1	13.9	44.2	14.3
MABC	GPT-3.5-Turbo	58.1	52.8	51.1	46.7
MABC	GPT-4-Turbo	**73.0**	**10.4**	**68.8**	**11.7**

Table 7: Human Evaluation

Model	Base	R-Useful (Train)	R-Useful (AIOps)
Decision Tree [70]	-	-	-
ReAct [26]	GPT-3.5-Turbo	2.1	2.1
ReAct [26]	GPT-4-Turbo	2.4	2.3
MABC	GPT-3.5-Turbo	3.1	3.2
MABC	GPT-4-Turbo	**4.2**	**3.6**

5 ANALYSIS

5.1 Decision Efficiency.

Following RCAgent [27], we use pass rate (PR) and average path length (APL) to evaluate the thinking trajectory steps of MABC in accomplishing the task, considering the validness of action trajectories and stability of the autonomous agent. PR calculated by

Table 8: Component Impact Evaluation

Model	Train-Ticket Dataset					AIOps Challenge Dataset				
	RA	PA	PR	APL	R-Useful	RA	PA	PR	APL	R-Useful
MABC	**72.4**	**66.2**	**84.7**	**8.6**	**4.2**	**63.5**	**57.3**	**79.3**	**9.1**	**3.6**
MABC w/o *Agent Workflow*	64.2	56.7	77.7	9.8	3.5	54.6	50.3	71.3	10.7	3.3
MABC w/o *Multi-Agent*	38.4	33.0	52.9	13.7	2.8	32.4	28.8	50.1	13.7	2.7
MABC w/o Voting	62.8	57.9	75.7	10.9	3.3	58.1	54.7	68.0	10.2	3.4

$\frac{N_p}{N_t}$, where N_p denotes the number of trajectories completed within θ steps, θ typically set to 15, and N_t denotes the total number of trajectories. Besides, APL is denoted by $\frac{\sum_{k=1}^{N_p} L_k}{N_p}$, where L_k denotes the path length of the k-th successful trajectory.

In Table 6, the results confirm that MABC exhibits superior decision efficiency, as evidenced by its high pass rate (PR) and low average path length (APL) across both datasets, which demonstrates that MABC not only successfully completes tasks with a higher probability but also does so with fewer steps, indicating a more efficient and stable decision-making process. The GPT-4-Turbo variant, in particular, showcases the most effective decision trajectories, suggesting that MABC is highly capable of generating efficient actions in the context of root cause analysis in a micro-services architecture.

MABC delivers outstanding decision efficiency, with GPT-4-Turbo leading the charge in fostering both effective and stable decision-making processes in the AIOps domain.

5.2 Human evaluation.

The evaluation was conducted based on a random selection of 200 completed cases, which included analyses of root causes, pathways, and resolutions. We invited 10 AIOps experts to rate each case on a scale from 1 to 5, where 1 signifies **very useless** and 5 denotes **very useful**. Subsequently, we calculated the average rating for each case to derive *Resolution Evaluation Metrics (R-Useful)* score.

In a more detailed examination of the human evaluation results presented in Table 7, it is evident that the responses from AIOps experts reveal a strong preference for the resolutions generated by the MABC model, especially when augmented with GPT-4-Turbo. The superior R-Useful scores for MABC with GPT-4-Turbo across both datasets not only demonstrate its exceptional capability in crafting highly useful solutions but also indicate that the approach of model resonates well with the expert understanding and expectations in the complex AIOps domain. The moderate R-Useful scores assigned to the ReAct suggest that while they are capable of generating resolutions, there is a notable gap in terms of meeting the nuanced needs of AIOps experts. The decision tree model could not be evaluated for R-Useful due to its inability to generate resolutions.

The human evaluation clearly confirms that MABC, when paired with GPT-4-Turbo, outshines the competition. It is highly effective at developing solutions that align with expert insights and shows great promise in improving decision-making in the AIOps domain, potentially enhancing overall performance and productivity.

5.3 Component Impact.

In this section, we verify the impact of three main components in MABC, i.e., MABC without *Agent Workflow* (based on ReAct rather than *Agent Workflow*), MABC without *Multi-Agent* (*Agent Workflow*), and MABC without Blockchain-Inspired Voting.

In Table 8, it is evident that MABC in its complete form demonstrates superior performance across all metrics on both datasets, highlighting the importance of integrating all components for optimal functionality. The removal of *Agent Workflow* leads to a noticeable decline in performance metrics, indicating the role of *Agent Workflow* in enhancing the effectiveness of model. Similarly, limiting the framework to only a Single Agent significantly reduces the capability of model to effectively address AIOps tasks, as seen by the lowest scores in all evaluated metrics. Lastly, excluding the Blockchain-Inspired Voting component also results in decreased performance, albeit to a lesser extent than removing *Agent Workflow* or resorting to a Single Agent setup, underscoring the value of the voting mechanism in refining and validating the resolutions generated by the framework.

Component impact evaluation clearly demonstrates the significance of each individual component within MABC. The integration of *Agent Workflow* contributes substantially to the ability of model to generate effective solutions by providing a structured approach to problem-solving. *Multi-Agent* architecture plays a crucial role in capturing diverse perspectives and enhancing the robustness of the decision-making process. Meanwhile, the Blockchain-Inspired Voting mechanism is essential for achieving consensus and ensuring the reliability of the final decisions. Together, these components work in synergy to elevate the performance of MABC, making it a powerful tool in the AIOps domain. The results underscore the importance of a holistic design where each component complements and reinforces the others, leading to a more effective and reliable root cause analysis in a micro-services architecture.

6 CONCLUSION

In this paper, we introduce MABC, a novel framework aimed at improving alert incident resolution in complex micro-services through a blend of multi-agent, LLMs, and a unique blockchain voting. Then, we develop train-ticket benchmark, an open-source evaluation dataset designed specifically for RCA in complex micro-services architectures. Experimental results on the public benchmark AIOps challenge dataset and our created train-ticket dataset, MABC demonstrated effectiveness in identifying root causes and offering solutions, with its *Agent Workflow* and voting mechanism playing crucial roles. MABC streamlines root cause analysis, boosting system reliability and operational efficiency. Our future efforts will focus on enhancing its components, incorporating more data sources, and improving agent collaboration. Our goal is to make MABC an essential tool for addressing various IT operations challenges.

REFERENCES

[1] A. Alquraan, H. Takruri, M. Alfatafta, and S. Al-Kiswany, "An analysis of network-partitioning failures in cloud systems," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*, 2018.

[2] Y. Gao, W. Dou, F. Qin, C. Gao, D. Wang, J. Wei, R. Huang, L. Zhou, and Y. Wu, "An empirical study on crash recovery bugs in large-scale distributed systems," in *Proceedings of the 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering (ESEC/FSE'18)*, 2018.

[3] Y. Zhang, J. Yang, Z. Jin, U. Sethi, K. Rodrigues, S. Lu, and D. Yuan, "Understanding and detecting software upgrade failures in distributed systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, 2021.

[4] H. Liu, S. Lu, M. Musuvathi, and S. Nath, "What bugs cause production cloud incidents?" in *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS'19)*, 2019.

[5] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[6] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93–104, 2013.

[7] K. Wang, C. Fung, C. Ding, P. Pei, S. Huang, Z. Luan, and D. Qian, "A methodology for root-cause analysis in component based systems," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. IEEE, 2015, pp. 243–248.

[8] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*. Springer, 2018, pp. 3–20.

[9] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 492–502.

[10] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, 2020.

[11] M. Ma, Z. Yin, S. Zhang, S. Wang, C. Zheng, X. Jiang, H. Hu, C. Luo, Y. Li, N. Qiu *et al.*, "Diagnosing root causes of intermittent slow queries in cloud databases," *Proceedings of the VLDB Endowment (VLDB'20)*, 2020.

[12] Y. Zhang, Z. Guan, H. Qian, L. Xu, H. Liu, Q. Wen, L. Sun, J. Jiang, L. Fan, and M. Ke, "Cloudrca: a root cause analysis framework for cloud computing platforms," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

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

[14] D. Yuan, Y. Luo, X. Zhuang, G. R. Rodrigues, X. Zhao, Y. Zhang, P. Jain, and M. Stumm, "Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems." in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*, 2014.

[15] T. Leesatapornwongsa, C. A. Stuardo, R. O. Suminto, H. Ke, J. F. Lukman, and H. S. Gunawi, "Scalability bugs: When 100-node testing is not enough," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS'17)*, 2017.

[16] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, and D. Pei, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 48–58.

[17] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 683–694.

[18] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, and D. Pei, "Practical root cause localization for microservice systems via trace analysis," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.

[19] OpenAI, "Gpt-4 technical report," 2023.

[20] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," 2022.

[21] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *arXiv preprint arXiv:2205.11916*, 2022.

[22] J. Wei *et al.*, "Chain of thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[23] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[24] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems*, 2023.

[25] 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," in *Advances in Neural Information Processing Systems*, 2020, pp. 9459–9474.

[26] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023.

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

[28] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving ai tasks with chatgpt and its friends in huggingface," *arXiv preprint arXiv:2303.17580*, 2023.

[29] Y. Chen *et al.*, "Empowering cloud rca with augmented large language models," *arXiv preprint arXiv:2311.00000*, 2023.

[30] 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," 2023.

[31] H. Chen, W. Dou, Y. Jiang, and F. Qin, "Understanding exception-related bugs in large-scale cloud systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19)*, 2019.

[32] C. Lou, P. Huang, and S. Smith, "Understanding, detecting and localizing partial failures in large system software." in *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*, 2020.

[33] 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.

[34] M. Li, M. Ma, X. Nie, K. Yin, L. Cao, X. Wen, Z. Yuan, D. Wu, G. Li, W. Liu *et al.*, "Mining fluctuation propagation graph among time series with active learning," in *Database and Expert Systems Applications: 33rd International Conference*, 2022.

[35] H. Guo, X. Lin, J. Yang, Y. Zhuang, J. Bai, T. Zheng, B. Zhang, and Z. Li, "Translog: A unified transformer-based framework for log anomaly detection," *arXiv preprint arXiv:2201.00016*, 2021.

[36] H. Guo, J. Yang, J. Liu, J. Bai, B. Wang, Z. Li, T. Zheng, B. Zhang, J. Peng, and Q. Tian, "Logformer: A pre-train and tuning pipeline for log anomaly detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 1, 2024, pp. 135–143.

[37] H. Guo, Y. Guo, R. Chen, J. Yang, J. Liu, Z. Li, T. Zheng, W. Hou, L. Zheng, and B. Zhang, "Loglg: Weakly supervised log anomaly detection via log-event graph construction," 2023.

[38] 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," *arXiv preprint arXiv:2309.09298*, 2023.

[39] W. Zhang, H. Guo, A. Le, J. Yang, J. Liu, Z. Li, T. Zheng, S. Xu, R. Zang, L. Zheng, and B. Zhang, "Lemur: Log parsing with entropy sampling and chain-of-thought merging," 2024.

[40] S. Locke, H. Li, T.-H. P. Chen, W. Shang, and W. Liu, "Logassist: Assisting log analysis through log summarization," *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3227–3241, 2021.

[41] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[42] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Llmparser: A llm-based log parsing framework," *arXiv preprint arXiv:2310.01796*, 2023.

[43] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *NIPS*, 2017.

[44] J. Yang, S. Ma, D. Zhang, J. Wan, Z. Li, and M. Zhou, "Smart-start decoding for neural machine translation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 3982–3988.

[45] J. Yang, S. Ma, D. Zhang, Z. Li, and M. Zhou, "Improving neural machine translation with soft template prediction," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 5979–5989.

[46] J. Yang, S. Ma, D. Zhang, S. Wu, Z. Li, and M. Zhou, "Alternating language modeling for cross-lingual pre-training," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 9386–9393.

[47] J. Yang, S. Ma, L. Dong, S. Huang, H. Huang, Y. Yin, D. Zhang, L. Yang, F. Wei, and Z. Li, "Ganlm: Encoder-decoder pre-training with an auxiliary discriminator," *arXiv preprint arXiv:2212.10218*, 2022.

[48] L. Chai, J. Yang, T. Sun, H. Guo, J. Liu, B. Wang, X. Liang, J. Bai, T. Li, Q. Peng *et al.*, "xcot: Cross-lingual instruction tuning for cross-lingual chain-of-thought reasoning," *arXiv preprint arXiv:2401.07037*, 2024.

[49] A. Aghajanyan, L. Yu, A. Conneau, W. Hsu, K. Hambardzumyan, S. Zhang, S. Roller, N. Goyal, O. Levy, and L. Zettlemoyer, "Scaling laws for generative mixed-modal language models," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and

J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 265–279.

[50] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen, E. Chu, J. H. Clark, L. E. Shafey, Y. Huang, K. Meier-Hellstern, G. Mishra, E. Moreira, M. Omernick, K. Robinson, S. Ruder, Y. Tay, K. Xiao, Y. Xu, Y. Zhang, G. H. Ábrego, J. Ahn, J. Austin, P. Barham, J. A. Botha, J. Bradbury, S. Brahma, K. Brooks, M. Catasta, Y. Cheng, C. Cherry, C. A. Choquette-Choo, A. Chowdhery, C. Crepy, S. Dave, M. Dehghani, S. Dev, J. Devlin, M. Díaz, N. Du, E. Dyer, V. Feinberg, F. Feng, V. Fienber, M. Freitag, X. Garcia, S. Gehrmann, L. Gonzalez, and et al., "Palm 2 technical report," *CoRR*, vol. abs/2305.10403, 2023.

[51] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[52] M. Lewis *et al.*, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *ACL*, 2020.

[53] T. B. Brown *et al.*, "Language models are few-shot learners," *NeurIPS*, 2020.

[54] A. Chowdhery *et al.*, "Palm: Scaling language modeling with pathways," *ArXiv*, 2022.

[55] J. Kaplan *et al.*, "Scaling laws for neural language models," *ArXiv*, 2020.

[56] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022.

[57] J. H. Park *et al.*, "Generative models as multi-agent systems," *Journal of Artificial Intelligence Research*, 2023.

[58] L. Sumers *et al.*, "Cognitive architectures for autonomous agents: A survey," *arXiv preprint arXiv:2302.00000*, 2023.

[59] L. Qin *et al.*, "Toolllm: Enhancing large language models with external tools for advanced problem solving," *arXiv preprint arXiv:2305.00000*, 2023.

[60] M. Li *et al.*, "Api-aware language modeling for micro-service architectures," *arXiv preprint arXiv:2308.00000*, 2023.

[61] X. Jin *et al.*, "Assessing the impact of large language models in cloud service root cause analysis," *arXiv preprint arXiv:2309.00000*, 2023.

[62] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable code actions elicit better llm agents," 2024.

[63] A. Wang *et al.*, "Interactive learning with autonomous agents and large language models," *arXiv preprint arXiv:2303.00000*, 2023.

[64] Y. Zhou *et al.*, "Llm-based autonomous agents for dynamic environments," *arXiv preprint arXiv:2304.00000*, 2023.

[65] S. Ahmed *et al.*, "Recommending fixes for cloud service failures with llm-enhanced tools," *arXiv preprint arXiv:2310.00000*, 2023.

[66] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu, "Automatic root cause analysis via large language models for cloud incidents," 2023.

[67] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2018.

[68] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, and X. Liu, "Enjoy your observability: an industrial survey of microservice tracing and analysis," *Empirical Software Engineering*, vol. 27, pp. 1–28, 2022.

[69] alibaba. (2021) https://github.com/chaosblade-io/chaosblade. [Online]. Available: https://github.com/chaosblade-io/chaosblade

[70] I. Abdallah, V. Dertimanis, H. Mylonas, K. Tatsis, E. Chatzi, N. Dervili, K. Worden, and E. Maguire, "Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data," in *Safety and Reliability–Safe Societies in a Changing World*. CRC Press, 2018, pp. 3053–3061.