Accurate and Interpretable Log-Based Fault Diagnosis Using Large Language Models

Yongqian Sun, *Member, IEEE*, Shiyu Ma, Tong Xiao, Yongxin Zhao, Xuhui Cai, Wei Dong, Yue Shen, Yao Zhao, Shenglin Zhang, *Member, IEEE*, Jing Han, and Dan Pei, *Senior Member, IEEE*

***Abstract*—Log-based fault diagnosis is essential for ensuring system reliability and resilience. However, current methods only provide fault diagnosis results without explanations, which undermines their credibility. Large language models (LLMs) have extensive pre-trained knowledge and show potential in log analysis, yet they cannot be directly applied to log-based fault diagnosis due to limited specialized capabilities and domain-specific insights. Furthermore, LLMs have limitations in context length and are too diverse to select a suitable one. To address these issues, this paper presents *LogInsight*, a framework that enables accurate and interpretable log-based fault diagnosis using LLMs. We fine-tune a medium-sized, open-source LLM to incorporate domain expertise and leverage its interpretive capability. Additionally, we design a Fault-Oriented Log Summary (FOLS) module to extract essential information from log sequences, mitigating LLMs' context length limitation. Extensive evaluations on two public datasets and a real-world production dataset demonstrate *LogInsight*'s superiority over state-of-the-art methods in both performance and interpretability.**

***Index Terms*—Log-based fault diagnosis, log analysis, large language models, interpretability.**

Received 23 January 2025; revised 28 June 2025; accepted 4 August 2025. Date of publication 15 August 2025; date of current version 9 October 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62272249 and Grant 62302244 and in part by the Fundamental Research Funds for the Central Universities under Grant XXX-63253249. *(Corresponding author: Shenglin Zhang.)*

Yongqian Sun is with the College of Software, Nankai University, Tianjin 300350, China, and also with the Tianjin Key Laboratory of Software Experience and Human Computer Interaction (TKL-SEHCI), Tianjin 300074, China (e-mail: sunyongqian@nankai.edu.cn).

Shiyu Ma and Yongxin Zhao are with the College of Software, Nankai University, Tianjin 300350, China (e-mail: mashiyu@mail.nankai.edu.cn; zyx_nkcs@mail.nankai.edu.cn).

Tong Xiao and Dan Pei are with the Department of Computer Science, Tsinghua University, Beijing 100084, China (e-mail: xiaotong@tsinghua.edu.cn; peidan@tsinghua.edu.cn).

Xuhui Cai, Wei Dong, and Yue Shen are with China Mobile Communications Group Company Ltd., Beijing 100032, China (e-mail: caixuhui@chinamobile.com; dongweiwl@chinamobile.com; shenyue@chinamobile.com).

Yao Zhao is with China Mobile Communications Group Jiangsu Company Ltd., Jiangsu 223600, China (e-mail: zhaoyao@js.chinamobile.com).

Shenglin Zhang is with the College of Software, Nankai University, Tianjin 300350, China, also with the Haihe Laboratory of Information Technology Application Innovation (HL-IT), Tianjin 300459, China, and also with the Key Laboratory of Data and Intelligent System Security, Ministry of Education, 300350, China (e-mail: zhangsl@nankai.edu.cn).

Jing Han is with ZTE Corporation, Shanghai 201203, China (e-mail: han.jing28@zte.com.cn).

Digital Object Identifier 10.1109/TSC.2025.3599494

I. Introduction

In MODERN large-scale and complex online service systems, faults can disrupt interdependent services, significantly degrade system performance, and impact millions of users [1]. If these faults are not promptly mitigated, they can result in severe consequences and substantial financial losses [2]. When a fault occurs, the system generates extensive logs containing vital information about the underlying issue [3]. Operations and Maintenance (O&M) engineers analyze these logs to diagnose the fault, a process that involves fault triage, which is indispensable for prioritizing remediation efforts and assigning the fault to the appropriate team for resolution [4]. Effective fault diagnosis not only accelerates recovery times but also minimizes the risk of cascading failures [5]. Consequently, log-based fault diagnosis plays a critical role in ensuring the reliability and stability of online services [6].

Manual log analysis is inherently time-consuming and labor-intensive, posing significant challenges for O&M engineers tasked with managing the massive and complex log data generated by large-scale systems [6]. To address these challenges, automated log-based fault diagnosis leveraging machine learning and deep learning algorithms has emerged as a promising solution [7]. Despite significant advancements in this field [8], [9], [10], existing approaches often suffer from a critical limitation: poor interpretability. Most approaches focus on producing diagnostic results without providing clear explanations for these outcomes, which can hinder O&M engineers' ability to trust and act on the recommendations effectively [11].

Interpretable fault diagnosis goes beyond merely triaging faults; it provides the rationale behind the triage results, enabling O&M engineers to quickly and accurately assign faults to the appropriate teams while fostering a better understanding of the results and facilitating targeted mitigation measures. As illustrated in Fig. 1, the transition from traditional fault diagnosis to interpretable methods fosters a more transparent and informed approach, bolstering confidence in decision-making and enhancing the overall reliability of fault management systems.

Recently, large language models (LLMs) have demonstrated promising capabilities across various natural language processing (NLP) tasks [12], [13]. Trained on vast datasets encompassing code [14] and log data [15], LLMs have the potential to provide a more comprehensive and context-aware understanding of logs. Several studies have explored the application of LLMs in log analysis [11], [16], [17]. However, leveraging

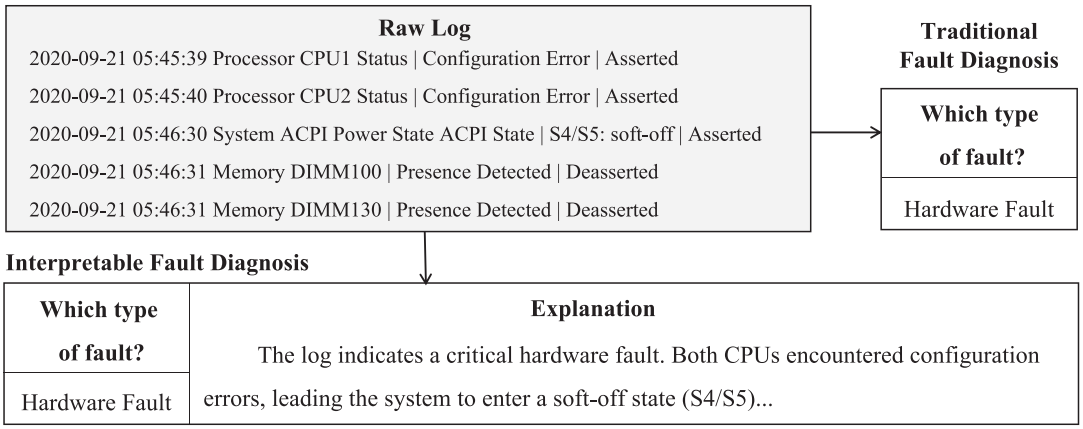


Fig. 1. Comparison between traditional fault diagnosis and interpretable fault diagnosis.

LLMs for interpretable fault diagnosis faces the following challenges:

1) *LLMs' lack of domain expertise:* LLMs are not specifically trained for fault diagnosis and often lack the domain-specific skills and contextual understanding required to perform fault diagnosis effectively. Consequently, directly applying LLMs to log-based fault diagnosis may result in poor performance.
2) *LLMs' limitations in context length:* Fault diagnosis typically requires analyzing extensive logs, yet LLMs have limited context length, making it impractical to input all logs at once. Although recent researche has focused on extending the context length of LLMs [18], [19], challenges persist due to the exponential increase in computational demands and accuracy degradation [20].

To address the above challenges, we propose *LogInsight*, a framework for accurate and interpretable log-based fault diagnosis using LLMs. Specifically, we fine-tune a medium-sized, open-source LLM to incorporate domain-specific knowledge and design a Fault-Oriented Log Summary (FOLS) module to extract critical information from each log sequence, thereby mitigating LLMs' inherent context-length limitations. We conduct comprehensive evaluations on two public datasets and one production dataset. The results demonstrate that *LogInsight* achieves the highest average F1 score in fault diagnosis, outperforming state-of-the-art baseline methods by 36.9%, 12.8%, and 7.3%, respectively. Additionally, *LogInsight* enhances interpretability by providing justifications for its diagnostic outcomes, supporting more informed decision-making for O&M engineers. In summary, the main contributions of this paper are as follows:

- We propose a practical framework for log-based fault diagnosis using LLMs that provides explanations for fault triage results.
- We design a Fault-Oriented Log Summary (FOLS) module to address LLMs' context-length limitations, enabling the processing of large log volumes essential for fault diagnosis.
- We conduct extensive evaluations on three real-world log datasets to validate the effectiveness of *LogInsight*, particularly its ability to provide interpretable results while achieving superior performance compared to existing state-of-the-art methods.

II. Background

A. Log-Based Fault Diagnosis

When a fault occurs in a system, O&M engineers are responsible for performing triage, diagnosing the root cause, and promptly implementing mitigation measures. Fault diagnosis, which encompasses fault triage, generally involves analyzing log data generated during the fault period—referred to as fault-related log data—serving as the primary source of diagnostic information [3]. However, the sheer volume of this data can be overwhelming, making manual review both labor-intensive and time-consuming and often delaying responses to critical faults [6].

To reduce manual workload, O&M engineers often create heuristic rules based on their experience to aid in log retrieval and fault identification. While useful, these heuristics have notable limitations. First, manually creating and updating rules is labor-intensive, especially as system requirements and configurations evolve. Second, heuristic-based diagnosis is prone to errors, leading to misdiagnoses and ineffective mitigation strategies [8]. As a result, automated log-based fault diagnosis using machine learning and deep learning techniques has gained significant attention [3], [7], [8], [21].

B. Large Language Models

In recent years, large language models (LLMs) have transformed the field of natural language processing (NLP), drawing significant interest from researchers and practitioners. OpenAI's introduction of GPT [22] demonstrated the potential of models pre-trained on large datasets and fine-tuned for specific tasks, paving the way for models like BERT [23], T5 [24], and GPT-3 [25]. More recently, advancements such as ChatGPT have broadened the application of LLMs, providing highly conversational interfaces with remarkable dialogue capabilities.

This trend has continued with OpenAI's GPT-4 [26], Meta AI's LLaMA [27], and Google's PaLM [28].

LLMs are predominantly built on the Transformer architecture [29] and are trained on extensive datasets using self-supervised learning objectives. They generate text autoregressively—predicting one token at a time—until a sequence is complete [30]. At the core of the Transformer is the self-attention mechanism, which allows models to capture dependencies across sequences; however, its computational cost is high, with memory and processing requirements growing quadratically with input length. This constraint makes handling long text inputs challenging, as information relevance can diminish over extended contexts, leading to issues such as attention dispersion.

Despite these challenges, LLMs' extensive pre-training on datasets that include code and log data makes them promising candidates for log-based fault diagnosis. Effectively applying LLMs to these tasks has become a research priority, with fine-tuning task-specific datasets emerging as the dominant approach.

C. Pre-Training & Fine-Tuning

The introduction of BERT [23] marked a pivotal shift in NLP, establishing the pre-training and fine-tuning paradigm. This paradigm involves first pre-training language models on large, diverse datasets to capture general linguistic and semantic patterns, then fine-tuning them on domain-specific data to meet the specific needs of downstream tasks. While general-purpose LLMs exhibit robust language understanding, they may underperform in specialized domains, where fine-tuning is essential for achieving optimal performance.

Fine-tuning entails additional training of a pre-trained model on domain-specific data, aligning the model more closely with the nuances of a specific application. There are two primary approaches to fine-tuning large models:

- *Full Parameter Fine-Tuning:* This approach involves updating all parameters of the pre-trained model with task-specific data, allowing comprehensive adaptation to the new domain. While full parameter fine-tuning demands substantial computational resources, it fully leverages the model's pre-trained features, enabling deep task-specific customization.
- *Parameter-Efficient Fine-Tuning (PEFT):* PEFT selectively updates a small subset of the model's parameters or adds a small number of parameters, thereby reducing computational costs and training time. This is particularly useful when resources are constrained, and includes methods such as Prefix Tuning [31], Prompt Tuning [32], Adapter Tuning [33], [34], [35], and LoRA [36], [37]. By concentrating on updating specific components, PEFT provides a resource-efficient method for model adaptation without sacrificing effectiveness.

In summary, fine-tuning provides a critical mechanism for adapting powerful language models to specialized domains, significantly enhancing their applicability and efficacy in practical, domain-specific applications.

III. Related Work

A. Log-Based Fault Diagnosis

With advances in machine learning and deep learning, numerous approaches have been developed for automating log-based fault diagnosis. These approaches can be categorized into three main types:

Machine Learning-Based Methods: This type of method typically converts log events into vector representations for clustering and classification. For example, LogCluster [7] employs TF-IDF to represent log sequences as weighted vectors and utilizes hierarchical clustering, while Cloud19 [3] leverages word2vec embeddings to capture semantic features and classify faults. However, both approaches are challenged by the presence of noise and redundant log data, which can impact diagnostic accuracy.

Deep Learning-Based Methods: More recent deep learning-based methods have enhanced fault diagnosis capabilities, particularly for complex log patterns. MoniLog [38], for instance, is a distributed, real-time anomaly detection framework for large-scale systems that monitors structured log streams to detect sequential and quantitative anomalies. It identifies faults within log sequences and categorizes anomalies based on severity. Another example, SwissLog [39], combines semantic and temporal embeddings to diagnose faults that involve changes in log sequence order or timing intervals. Nevertheless, both methods may produce false positives or fail to detect critical issues when log data is ambiguous or complex.

Data Mining-Based Methods: Data mining-based methods enhance diagnostic detail by extracting richer information from logs. LogBASA [40] incorporates a pre-trained BERT model to capture semantic relationships within log event sequences, using system behavior analysis to improve diagnostic efficiency in response to anomalies. LogKG [8] takes a different approach by constructing a knowledge graph of entities and relationships extracted from logs. This graph, combined with a fault-oriented log representation module, leverages OPTICS clustering [41] to analyze fault patterns and perform online fault diagnosis. While these methods perform well with specific log formats, their applicability across varied log structures remains limited.

In summary, these methods have some unique advantages, but they all share a common shortcoming: lacking the ability to provide interpretable explanations for fault diagnosis results.

B. LLM-Based Log Analysis

The prevalence of LLMs has attracted considerable interest in applying them in log analysis tasks, particularly in log parsing and log anomaly detection.

LLM-Based Log Parsing: Log parsing, the process of transforming raw logs into structured data, serves as the foundation of the typical log analysis workflow. Recent research has explored using LLMs to improve log parsing. For instance, DivLog [42] leverages the in-context learning (ICL) capability of LLMs, using diverse log samples as guiding examples to achieve accurate parsing across varied log formats. LILAC [43] enhances parsing precision and efficiency by combining LLMs with an

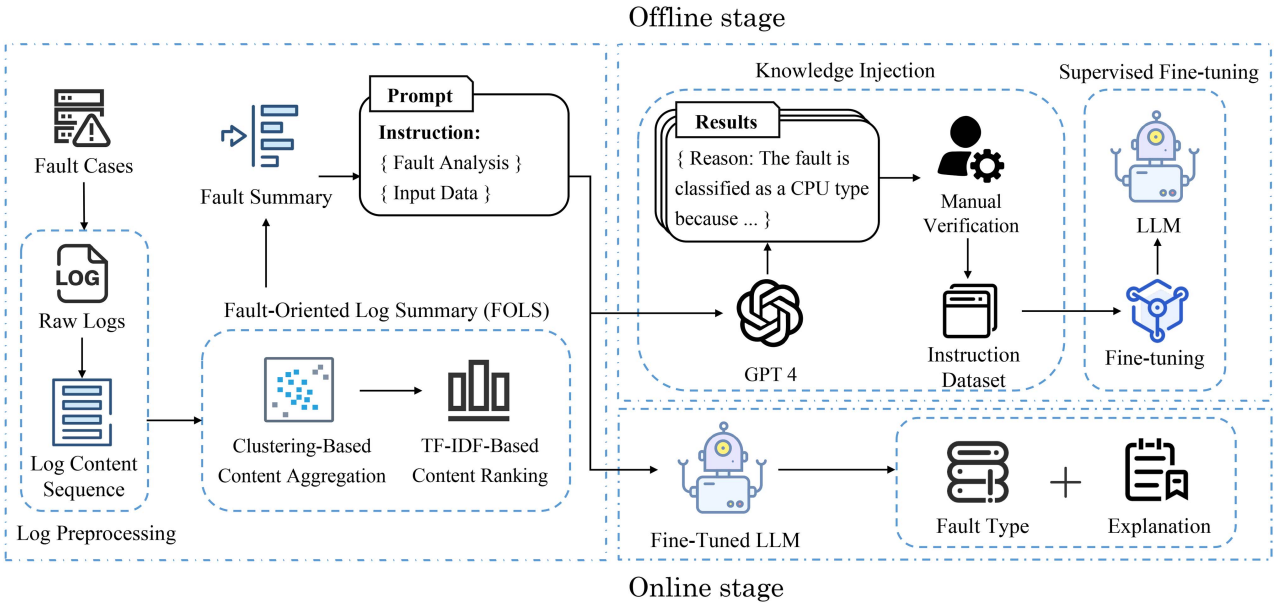


Fig. 2. The overall framework of *LogInsight*.

Adaptive Parsing Cache, which stores refined templates for rapid retrieval and incorporates hierarchical candidate sampling to further optimize ICL performance.

LLM-Based Log Anomaly Detection: Detecting anomalies within logs is essential for fault identification and early warning. Recently, LLMs have emerged as valuable tools for enhancing anomaly detection. For example, LogPrompt [11] employs a prompting strategy tailored for log analysis, enabling zero-shot parsing and anomaly detection without extensive model fine-tuning. LogGPT [16], based on the ChatGPT framework, uses the interpretive capabilities of LLMs for anomaly detection, although it does not explicitly classify fault types. SeaLog [44] utilizes a Trie-based Detection Agent for real-time anomaly detection, incorporating expert insights, including those informed by LLMs, to enhance detection accuracy. RAGLog [45] combines a Retrieval-Augmented Generation (RAG) LLM with a vector database, enabling effective handling of log volume, variety, and velocity in anomaly detection scenarios.

Additionally, there is research exploring the use of LLMs for log generation. UniLog [46] serves as an automatic logging framework that exploits LLMs' ICL abilities to autonomously generate logging statements, offering a versatile and minimally tuned solution that caters to diverse organizational and developer logging needs.

While these studies highlight the capabilities of LLMs in log analysis tasks such as log parsing and log anomaly detection, the application of LLMs specifically for log-based fault diagnosis remains underexplored. This gap highlights the need for approaches that address the distinct requirements of interpretable fault diagnosis in log analysis.

IV. Approach

In this section, we introduce the details of *LogInsight*. Fig. 2 shows the overall framework of *LogInsight*, which consists of four main components: **Log Preprocessing**, which organizes raw log entries into structured log sequences; **Fault-Oriented Log Summary (FOLS)**, which extracts key information from log sequences; **Knowledge Injection**, which generates an explanation for each fault summary using GPT-4, followed by manual verification of the explanation; and **Supervised Fine-tuning**, which fine-tunes an LLM using an instruction dataset. Next, we explain each component in detail.

A. Log Preprocessing

This component preprocesses log data for subsequent analysis. Suppose a fault occurs at time t, we first collect raw logs generated within the time frame $[t - w : t]$ for this fault, where w is determined according to the requirement of the application scenario. We then transform these raw logs into a log content sequence to facilitate analysis. Unlike traditional log analysis methods, *LogInsight* does not require log parsing, as it utilizes LLMs to understand the content of the raw logs.

Fig. 3 illustrates the log preprocessing process, where regular expressions are used to structure the raw logs and extract the *Content* field from each log entry, forming a sequence of log contents. This resulting sequence removes redundant and irrelevant fields from the original logs, providing more detailed information than the log template sequences typically generated through log parsing.

B. Fault-Oriented Log Summary

To address *Challenge 2: LLMs' limitations in context length*, we design the Fault-Oriented Log Summary (FOLS) module. Through an extensive analysis of fault cases, we observed that logs generated during failure periods often contain a large amount of redundant and noisy information while the truly valuable diagnostic clues are sparse and rarely repeated. Moreover, different types of faults are characterized by distinct log patterns, making it challenging to identify key information using simple heuristics. These insights motivated the development of

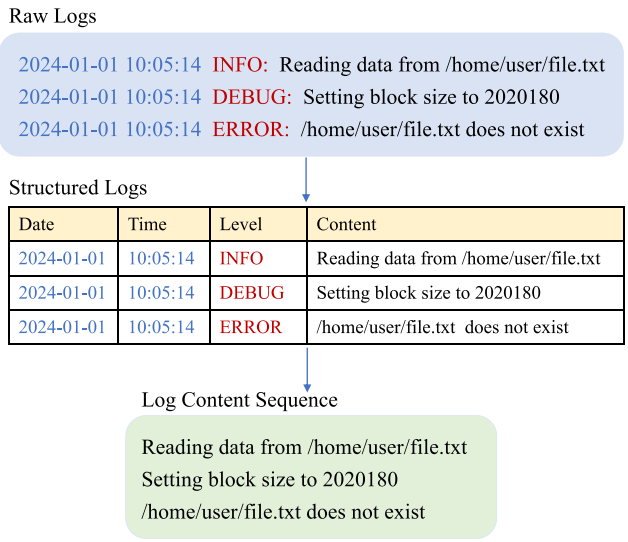


Fig. 3. The workflow of log preprocessing.

the FOLS module, which is specifically tailored to identify and highlight meaningful and unique content within large, repetitive datasets.

Given the substantial size of raw logs required for fault diagnosis and the context-length constraints of LLMs, FOLS aims to extract essential information while significantly reducing input length. As illustrated in Fig. 4, this approach makes the time and memory overhead of feeding logs into LLMs manageable. Specifically, FOLS employs two operations: *Clustering-Based Content Aggregation* to group similar log entries and reduce redundancy, and *TF-IDF-Based Content Ranking* to prioritize log messages that are most informative for fault diagnosis.

1) Clustering-Based Content Aggregation: The log data for a fault case can exhibit complex correlations. To ensure the diversity of summarized sequences and capture faults from different perspectives, we group related logs together by clustering. This allows subsequent analysis to focus on a small subset of logs from each cluster rather than all logs. This operation consists of three steps:

i) Distance Measurement: Given N log contents to be clustered, we create an $N \times N$ distance matrix to record the distance between each pair of log contents. We choose the Jaccard distance, which can measure syntactic-level similarity between two log contents [47]. The Jaccard distance is defined based on the intersection and union of the token sets from two log contents, effectively capturing their overlap and diversity.

Since each log content can be represented as a set of tokens, the Jaccard distance between two log contents x and y is defined as follows:

$$d(x,\, y) = 1 - \frac{|\mathbb{X} \cap \mathbb{Y}|}{|\mathbb{X} \cup \mathbb{Y}|}, \tag{1}$$

where $\mathbb{X}$ and $\mathbb{Y}$ are the token sets of log content x and y, respectively. It can be seen that the Jaccard distance is between 0 to 1.

ii) Clustering: After obtaining the distance matrix, we apply the DBSCAN [48] algorithm to cluster all log contents. DBSCAN is a density-based spatial clustering algorithm that partitions regions with sufficient density into clusters, capable of discovering clusters of arbitrary shapes in a spatial database with noise. Compared to other popular clustering algorithms, such as k-means [49], DBSCAN does not require the number of clusters to be specified in advance and can effectively cluster dense datasets of varying shapes. DBSCAN has two key parameters: ϵ, which describes the maximum neighborhood distance for a point, and $MinPts$, which indicates the minimum number of points required for a cluster. We set these values as recommended in [48].

iii) Representative Log Content Selection: After clustering, the log contents within each cluster are similar, so analyzing a single log content from each cluster can reveal the overall pattern. Without loss of generality, we select the centroid of each cluster as its representative log content and discard the rest. For a cluster with n log contents, the centroid is the log content that has the minimum average distance to all others within the cluster, formulated as follows:

$$centroid = \arg \min_{x_i \in cluster} \frac{1}{n} \sum_{j=1}^{n} d\left(x_i, x_j\right), \tag{2}$$

where $d(x_i, x_j)$ is the Jaccard distance between log contents x_i and x_j.

2) TF-IDF-Based Content Ranking: Often, some logs unrelated to the fault may appear multiple times. To mitigate this issue, we propose a TF-IDF-based ranking algorithm to evaluate the importance of each log content and filter out the less significant ones. TF-IDF [50] is a widely used weighting technique in information retrieval and data mining [7], comprising two parts: Term Frequency (TF) and Inverse Document Frequency (IDF). TF measures how frequently a term appears, while IDF reduces the weight of common terms, assigning them a smaller weight inversely proportional to their frequency.

Specifically, for a given log content, we first tokenize it into a token list $\mathbb{T} = [t_1, t_2, \ldots, t_n]$. The TF of an arbitrary token t is calculated as follows:

$$TF(t) = \frac{n_t}{n}, \tag{3}$$

where n_t is the number of appearances of token t in $\mathbb{T}$ and n is the total number of tokens in $\mathbb{T}$. The IDF is calculated as:

$$IDF(t) = \log \frac{N}{n_t + 1}, \tag{4}$$

where N is the total number of faults and n_t is the number of faults whose logs contain token t. After obtaining the TF and IDF for each token, the score of a log content is calculated as:

$$score = \sum\nolimits_i TF(t_i) \times IDF(t_i). \tag{5}$$

After ranking all log contents in descending order according to their respective scores, those falling below a predefined threshold are excluded. The remaining high-scoring log entries are retained as the fault summary and are then reordered to follow their original chronological order.

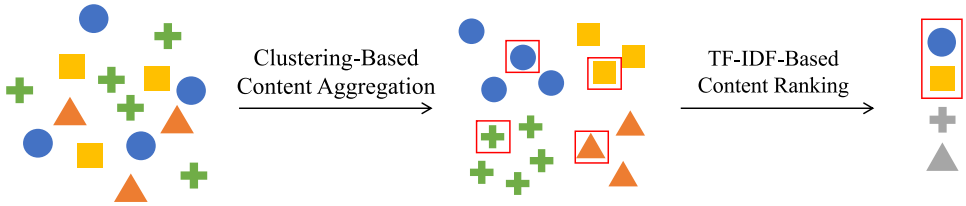


Fig. 4. The workflow of the FOLS module, where different shapes represent log contents of different types.

Instruction: Your task is to determine what type of fault a given set of log information belongs to. Here are the possible fault types in our data scenario: {placeholder}. Please determine its fault type based on the log sequence I input and provide your explanation.
Input: Log sequence: [placeholder]
Output: Answer:

Fig. 5. Example structure of a data item in *LFDInstruction*.

C. Knowledge Injection

To address *Challenge 1: LLMs' lack of domain expertise*, we introduce a knowledge injection step. General LLMs often lack the specialized capabilities required for effective log-based fault diagnosis. To address this gap, we inject domain knowledge into LLMs through fine-tuning. The quality of the training data is a crucial factor in determining the performance of LLMs [51]. To this end, we curated a high-quality dataset called *LFDInstruction*, specifically designed for log-based fault diagnosis.

We first enlisted O&M experts to collect log data for a wide range of faults and then employed the Fault-Oriented Log Summary (FOLS) module to generate concise summaries for each fault. These summaries serve as inputs, while the correct fault diagnoses and detailed analyses act as outputs. These input-output pairs, along with carefully crafted instructions, comprise the *LFDInstruction* dataset. Fig. 5 illustrates the structure of a data item in this dataset.

To streamline the dataset creation process and reduce manual labor, we initially utilized GPT-4 [26] to generate preliminary outputs. This generation process was carefully designed to ensure that the content was both contextually relevant and diverse. However, recognizing the limitations of automated content generation, we implemented a validation approach: expert reviewers meticulously evaluate each output sequence. This manual review process involves a comprehensive assessment of the fault analyses in relation to the corresponding logs, allowing experts to correct any inaccuracies or inconsistencies in the diagnoses. This combined strategy of automated generation followed by expert validation not only reduces labor costs but also significantly enhances the overall quality of the data.

D. Supervised Fine-Tuning

We employed LoRA [36] to conduct supervised fine-tuning. As previously mentioned, for log-based fault diagnosis, we developed the *LFDInstruction* dataset as our training corpus. This dataset comprises a series of triplets $\{x_i, y_i, I_i\}$, where x_i represents the input sequence, y_i denotes the output sequence, and I_i is the corresponding instruction.

The primary objective of this process is to optimize an LLM (denoted as M), enabling it to generate the output sequence by modeling the relationship $y_i = M(x_i, I_i)$. To facilitate this, we concatenate the instruction I_i and the input sequence x_i before feeding them into model M. The fine-tuning process aims to minimize the error between the model's output and the actual diagnoses by minimizing a loss function $\mathcal{L}$, defined as:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(M_\theta(x_i, I_i), y_i\right) \tag{6}$$

where θ represents the model parameters, N is the number of training samples, and $M_\theta(x_i, I_i)$ denotes the model's predicted output given input x_i and instruction I_i. Through iterative training, we aim to find the optimal parameters θ^* that minimize the average loss, thereby enhancing the model's accuracy in generating the correct diagnosis.

V. Evaluation

In this section, we evaluate our approach, *LogInsight*, by addressing the following research questions:

- **RQ1**: How effective is *LogInsight* in log-based fault diagnosis compared to state-of-the-art methods?
- **RQ2**: Can *LogInsight* meet the efficiency requirement for online deployment?
- **RQ3**: How does the FOLS module affect the performance of *LogInsight*?
- **RQ4**: Can *LogInsight* provide useful and comprehensible explanations for the results?
- **RQ5**: Is *LogInsight* compatible with different LLMs?

A. Datasets

We conduct experiments using two public log datasets and a production log dataset from China Mobile Communications Group Co., Ltd. (CMCC), a leading global Internet Service Provider (ISP). Table I summarizes key details of these three datasets, including the number of log templates identified by the Drain [52] algorithm.

Dataset 1:[1] This dataset consists of logs from servers in real-world business environments, including server serial

[1] Available at https://tianchi.aliyun.com/competition/entrance/531947/information

TABLE I
DETAILS OF THE DATASETS FOR EVALUATION

Dataset	Log Entries	Log Templates	Fault Types	Fault Cases
Dataset 1	282,537	157	3	2,671
Dataset 2	1,461,006	727	6	93
Dataset 3	178,773	99	9	2,267

Prompt: Your task is to determine what type of fault a given set of log information belongs to. Here are the possible fault types in our data scenario: {placeholder}. Please determine its fault type based on the log sequence I input and provide your explanation.

Fig. 6. The prompt used for log-based fault diagnosis.

numbers (SN), server models (SM), fault times, and associated message information for each fault. There are a total of 2,671 fault cases, which are categorized into three types: Processor CPU Caterr, Memory Constraint Error, and Hardware Error. To retrieve fault-related logs, we organized logs chronologically by timestamp and extracted logs in the 12 hours preceding each fault occurrence.

Dataset 2: [2] Sourced from OpenStack, a widely adopted open-source cloud computing platform [53], this dataset includes 1,461,006 log entries spanning 93 fault cases collected over 24 days. The logs cover six fault types: AMQP Server Unreachable, MySQL Lost Connection, Computing Node Down, Flavor Disk Too Small, Linuxbridge-agent Anomalies, and Nova-conductor Lost Connection.

Dataset 3: This dataset contains network device logs from CMCC's production environment, which supports 4 G/5 G core networks for a global user base. The dataset includes alarm logs collected from 322 switches over one year, totaling 178,773 log entries across nine fault types: Power Supply Fault, Fan Fault, Optics Module Fault, Port Flapping Fault, CRC Error, STP Fault, BFD Down, LACP Flapping, and OSPF Neighbor Flapping.

For each dataset, we randomly select 50 fault cases to construct the *LFDInstruction* dataset for fine-tuning, with the remaining cases for testing. We manually verify that the *LFDInstruction* dataset includes all fault types to ensure comprehensive coverage during fine-tuning.

B. Baseline Methods

We select three state-of-the-art log-based fault diagnosis methods as baselines: an unsupervised method, LogCluster [7], and two supervised methods, Cloud19 [3] and LogKG [8]. To ensure fairness, we set the hyperparameters according to the descriptions provided in the respective papers. Specifically, LogCluster utilizes a hierarchical clustering algorithm with a distance threshold of 0.5. For LogKG, we set the FOLR model threshold to 0.5, and configure the OPTICS clustering algorithm with a minimum of 10 samples per cluster.

To further demonstrate the importance and advantage of fine-tuning, we compared *LogInsight* with GPT-4 using the same prompt, as illustrated in Fig. 6

C. Evaluation Metrics

Fault diagnosis can be framed as a multi-class classification problem. To evaluate the effectiveness of *LogInsight*, we employ the following three metrics: Micro-Averaged F1-score (Micro F1), Macro-Averaged F1-score (Macro F1), and Weighted-Averaged F1-score (Weighted F1) [54].

Macro F1 calculates the F1 score for each class and then averages these scores across all classes. This metric assigns equal weight to each class, regardless of class size, providing insight into performance across all classes. The formula is as follows:

$$\text{Macro F1} = \frac{1}{N}\sum_{i=1}^{N}\text{F1}_i \tag{7}$$

where N is the total number of classes.

Micro F1 aggregates True Positives (TP), False Positives (FP), and False Negatives (FN) across all classes to calculate overall precision and recall. This metric is particularly suitable for handling imbalanced class distributions, as it weights each instance equally across classes. The formulas are as follows:

$$\text{Precision}_{\text{micro}} = \frac{\sum TP}{\sum TP + \sum FP} \tag{8}$$

$$\text{Recall}_{\text{micro}} = \frac{\sum TP}{\sum TP + \sum FN} \tag{9}$$

$$\text{F1}_{\text{micro}} = \frac{2 \cdot \text{Precision}_{\text{micro}} \cdot \text{Recall}_{\text{micro}}}{\text{Precision}_{\text{micro}} + \text{Recall}_{\text{micro}}} \tag{10}$$

Weighted F1, meanwhile, computes a weighted average of the F1 scores, factoring in the number of samples in each class. This approach gives greater influence to larger classes, ensuring that the metric reflects the overall distribution. The formula for Weighted F1 is:

$$\text{F1}_{\text{weighted}} = \sum_{i=1}^{N} w_i \cdot \text{F1}_i \tag{11}$$

where w_i is the proportion of samples in class i.

D. Implementation Details

All experiments are conducted on an Ubuntu 18.04 LTS server with two Intel(R) Xeon(R) Gold 6430 CPUs, each offering 64 cores and 128 threads, and two NVIDIA A800-80 GB GPUs. In selecting the base model, we evaluated several options. Given the constraints of memory and time, we chose four open-source LLMs, each with approximately 7 billion parameters, as candidates: Mistral-7B, Qwen1.5-7B-Chat, LLaMA2-7B, and Gemma-7B. As shown in Table II, Mistral-7B demonstrated superior performance compared to the other models, leading us to select Mistral-7B as our base model.

For supervised fine-tuning, we configure the learning rate to 10^{-4}, the weight decay to 0.1, the batch size to 16, and

[2] Avaialable at https://github.com/SycIsDD/LogKG

TABLE II
EVALUATION OF BASE MODEL

LLM	Dataset 1			Dataset 2			Dataset 3		
	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1
Mistral-7B	**0.451**	**0.218**	**0.508**	**0.302**	**0.212**	**0.292**	**0.489**	**0.483**	**0.591**
Qwen1.5-7B	0.431	0.175	0.351	0.209	0.057	0.088	0.218	0.173	0.232
LLaMA2-7B	0.187	0.063	0.189	0.139	0.068	0.098	0.044	0.039	0.055
Gemma-7B	0.151	0.050	0.166	0.069	0.053	0.082	0.056	0.046	0.056

TABLE III
PERFORMANCE COMPARISON BETWEEN *LOGINSIGHT* AND THE BASELINES

Method	Dataset1			Dataset2			Dataset3		
	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1
LogInsight	**0.884**	**0.883**	**0.883**	**0.998**	**0.998**	**0.997**	**0.997**	**0.997**	**0.997**
GPT-4	0.498	0.446	0.490	0.814	0.696	0.751	0.896	0.812	0.924
LogCluster	0.502	0.474	0.474	0.906	0.800	0.869	0.768	0.659	0.756
Cloud19	0.521	0.514	0.514	0.837	0.780	0.830	0.739	0.638	0.720
LogKG	0.446	0.377	0.377	0.739	0.664	0.677	0.676	0.478	0.557

the maximum token limit to 4096. We utilize LoRA [36] for parameter-efficient fine-tuning, setting the rank to 8 and the alpha parameter to 32, with a dropout rate of 0.05 to prevent overfitting.

E. Results and Analysis

1) RQ1: How Effective is LogInsight in Log-Based Fault Diagnosis Compared to State-of-the-Art Methods?

Table III presents the overall performance comparison on the three datasets, showing that *LogInsight* consistently outperforms all baseline methods across all evaluation metrics.

Specifically, *LogInsight* achieves Weighted F1 scores of 0.883, 0.997, and 0.997 on the three datasets, representing improvements of 36.9%, 12.8%, and 7.3% over the best-performing baseline method, respectively. The performance gains are especially pronounced on Dataset 1, likely due to the dataset's complex fault patterns, which challenge traditional methods. In contrast, the simpler log patterns in Datasets 2 and 3 allow all methods to triage faults accurately, albeit *LogInsight* still demonstrates superior performance.

Additionally, we observe that baseline methods consistently produce lower Macro F1 scores compared to their Micro and Weighted F1 scores, suggesting they struggle to handle imbalanced fault types effectively. *LogInsight*, however, maintains high stability across all metrics, demonstrating adaptability to varying data distributions. This advantage stems from *LogInsight*'s utilization of an LLM, which effectively captures the semantic content of log data, thereby facilitating more accurate fault triage. In contrast, traditional methods cannot fully exploit the rich semantic information inherent in log messages.

Furthermore, when compared to GPT-4, *LogInsight* demonstrates superior performance across all datasets. This improvement is attributed to the model's fine-tuning with domain-specific data, which enhances its diagnostic performance and enables it to outperform general-purpose LLMs in specialized fault diagnosis tasks.

2) RQ2: Can LogInsight Meet the Efficiency Requirement for Online Deployment?

To evaluate the efficiency of *LogInsight*, we measure the time required to diagnose each fault case and calculate the average diagnosis time on each dataset. The results are shown in Table IV, from which we can see that *LogInsight* takes 2.7 s, 8.5 s, and 2.8 s on average to diagnose a fault in the three datasets, respectively.

LogInsight is built upon LLMs, which inherently involve substantial computational overhead during both training and inference due to their massive parameter sizes and complex architectures. As a result, the time cost for model inference is generally higher than that of traditional baseline methods. Despite this, our results demonstrate that *LogInsight* can still complete fault diagnosis within an average of 8.5 seconds per case, which is well within the acceptable range for real-time online deployment scenarios. This efficiency is sufficient to significantly reduce the manual diagnosis workload while meeting the latency requirements of practical applications.

3) RQ3: How Does the FOLS module Affect the Performance of LogInsight?

We further investigate the effect of the Fault-Oriented Log Summary (FOLS) module on *LogInsight*'s performance. To this end, we remove the FOLS module and feed the log content sequences directly into the fine-tuned LLM, constrained by a maximum token limit of 8192 to include as much log data as possible.

Additionally, to evaluate the effectiveness of clustering in improving log summarization, we create several variants by replacing the DBSCAN algorithm in the Clustering-Based Content Aggregation module with different clustering and log parsing techniques. These include K-means [49], Hierarchical Agglomerative Clustering (HAC) [55], Drain [52], and two LLM-based parsing methods, DivLog [42] and LILAC [43].

Table V reveals a marked decrease in *LogInsight*'s diagnostic performance when the FOLS module is removed. This

TABLE IV
EFFICIENCY COMPARISON BETWEEN *LOGINSIGHT* AND THE BASELINES

Method	Dataset 1		Dataset 2		Dataset 3	
	Offline	Online	Offline	Online	Offline	Online
LogInsight	1802.28s	2.707s	4052.58s	8.541s	1977.11s	2.819s
GPT-4	-	1.637s	-	2.516s	-	1.124s
LogCluster	29.20s	<0.01s	9.41s	<0.01s	8.42s	<0.01s
Cloud19	52.79s	<0.01s	228.52s	<0.01s	15.721s	<0.01s
LogKG	140.46s	0.56s	1662.78s	8.12s	137.64s	0.71s

TABLE V
ABLATION STUDY OF THE FOLS MODULE

Method	Dataset 1			Dataset 2			Dataset 3		
	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1
LogInsight	**0.884**	**0.883**	**0.883**	**0.998**	**0.998**	**0.997**	**0.997**	**0.997**	**0.997**
LogInsight w/o FOLS	0.773	0.626	0.767	0.470	0.326	0.382	0.806	0.505	0.838
LogInsight with K-means	0.821	0.814	0.819	0.837	0.833	0.833	0.918	0.879	0.934
LogInsight with HAC	0.814	0.789	0.806	0.874	0.873	0.873	0.917	0.878	0.933
LogInsight with Drain	0.811	0.813	0.813	0.814	0.551	0.784	0.908	0.718	0.918
LogInsight with DivLog	0.769	0.770	0.770	0.733	0.578	0.636	0.877	0.763	0.884
LogInsight with LILAC	0.810	0.817	0.816	0.818	0.603	0.783	0.907	0.794	0.917

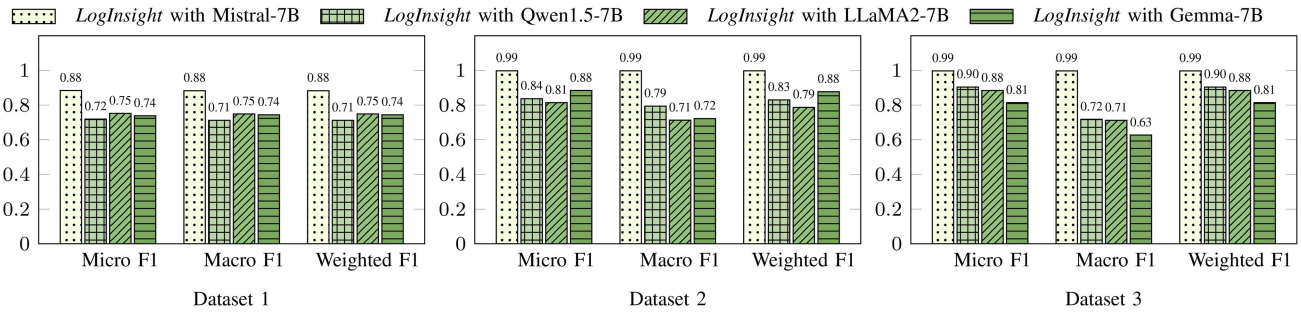


Fig. 7. The performance of *LogInsight* with different LLMs as base models.

decline can be attributed to two primary factors: (i) the raw log sequences contain redundancy and noise, leading to reduced information entropy; and (ii) the LLM's input token limits necessitate truncating log data, which risks omitting critical information.

Furthermore, the results show that clustering methods generally outperform log parsing methods in terms of accuracy. This is primarily due to the inherent parsing errors in these methods, which negatively impact downstream processing. The use of placeholders (e.g., "∗") in parsed content can mislead the LLM during fault analysis, thereby compromising diagnostic performance.

In general, the findings underscore the importance of the FOLS module in improving the robustness and effectiveness of *LogInsight*, particularly in managing the complexities and nuances of log data.

4) RQ4: Can LogInsight Provide Useful and Comprehensible Explanations for the Results?

To evaluate the interpretability of *LogInsight*'s outputs, we enlisted O&M experts from CMCC to assesses the quality of explanations generated for 200 randomly selected fault cases from Dataset 3. Fig. 8 shows the diagnostic results for two representative fault cases.

Following the evaluation methodology employed by LogPrompt [11], we assess *LogInsight*'s explanations along two primary dimensions:

- *Usefulness:* Reviewers were asked to evaluate the extent to which the explanations were detailed, specific, relevant, logically sound, and practical for actual fault diagnosis.
- *Readability:* Reviewers were asked to assess how easily a reader could understand the provided text. A text was considered readable if it was grammatically correct, meaningful, and professional.

For each evaluation dimension, we use a Likert scale ranging from one to five, with Table VI providing the detailed criteria for each level. The scoring system is as follows: 1 - Not Inter-

Fault Summary	Interpretable Fault Diagnosis
Port down reason Interface XXX1 is down, due to Los LACP protection status changed Port down reason Interface XXX2 is down, due to Los LACP set the smartgroup protocol down	**Fault Type:** Port Flapping Fault **Explanation:** Based on the provided log sequence, the issues primarily involve port flapping faults caused by LOS alarms. The logs show multiple ports going down due to these alarms. Additionally, there are state changes and down statuses related to LACP and the Smartgroup protocol. Therefore, the primary fault type is identified as the Port Flapping Fault.
Power abnormal Shelfnum =... groupid =... powerid =2 Power malfunction alarm Power input voltage error Power output voltage error	**Fault Type:** Power Supply Fault **Explanation:** The logs explicitly mention "power abnormal" and "Power malfunction alarm," indicating an issue with the power system. Additionally, "Power input voltage error" and "Power output voltage error" point to the input and output voltages being out of the normal range. This information indicates that the fault is a Power Supply Fault.

Fig. 8. Two fault cases from Dataset 3.

TABLE VI
HUMAN SCORING CRITERIA FOR EVALUATING THE INTERPRETABILITY OF *LOGINSIGHT*

Scores	Usefulness	Readability
1	No justification for the fault, only a simple prediction label.	The text contains numerous unintelligible elements or grammatical errors.
2	The justification is irrelevant or logically inconsistent with the facts.	Most of the text is readable, but it may contain grammar errors or unclear phrases.
3	The justification supports the prediction well, but lacks clarity or sufficient detail.	The text has few grammar errors, though some terms may need refinement.
4	The justification is specific, accurate, and relevant, helping engineers to reduce false alarms and guide further analysis.	The text is clear, grammatically correct, with only a few technical terms that may require refinement.
5	The justification is detailed, relevant, and clear, significantly aiding engineers in ruling out false alarms and identifying the root cause.	The text is clear, detailed, grammatically flawless, and professional for software engineering.

pretable, 2 - Low Interpretability, 3 - Moderate Interpretability, 4 - High Interpretability, and 5 - Very High Interpretability.

Each reviewer independently evaluated all 200 model-generated explanations, assigning scores for both usefulness and readability by referencing the original log context from the corresponding failure incident and adhering to predefined criteria.

Two key metrics are reported: (1) *Mean:* the mean score assigned by reviewers across all samples; (2) *High Interpretability Percentage (HIP):* the proportion of samples receiving a score of four or higher.

The evaluation results are listed in Table VII. In terms of usefulness, the average score is 3.80, with a HIP of 75.7% . This indicates that a majority of the explanations provided by *LogInsight* are effective in justifying the diagnosis and aiding engineers in their analysis. Regarding readability, the average score is 4.04, with a HIP of 86.7%, demonstrating that the explanations are grammatically correct and highly comprehensible. These results suggest that the explanations generated by *LogInsight* hold significant potential for assisting O&M engineers in validating fault diagnosis results and guiding subsequent analyses.

TABLE VII
INTERPRETABILITY OF *LOGINSIGHT* RATED BY EXPERTS

Raters	Usefulness		Readability	
	Mean	HIP	Mean	HIP
R1	3.62	62.5%	4.02	87.5%
R2	3.90	85.5%	4.03	85.0%
R3	3.87	79.0%	4.08	87.5%
Avg.	3.80	75.7%	4.04	86.7%

5) RQ5: Is LogInsight Compatible with Different LLMs?

To ensure the long-term effectiveness of *LogInsight*, it is essential to evaluate its compatibility with various LLM architectures and versions. To this end, we test several open-source models, including Qwen1.5-7B-Chat, LLaMA2-7B, and

Gemma-7B, as replacements for the original base model used in *LogInsight*.

As illustrated in Fig. 7, *LogInsight* achieves the highest performance when utilizing its original base model. However, it demonstrates comparable results with the alternative LLMs tested, reflecting its adaptability across diverse architectures. This compatibility highlights *LogInsight*'s flexibility and robustness, ensuring it can maintain satisfactory performance regardless of the underlying LLM used.

VI. Case Study

To ensure the case study is representative, we randomly select two fault cases from the dataset and use them, as shown in Fig. 8, to illustrate the workflow of *LogInsight*.

Case 1. Port Flapping Fault: For this case, 796 log entries were retrieved from the 10-minute period leading up to the fault event. The log entries were then preprocessed into a log content sequence, filtering out irrelevant data using regular expressions, after which the FOLS module extracted critical fault-related information. Using Clustering-Based Content Aggregation, the entries were reduced to 29 distinct logs. TF-IDF-Based Content Ranking further refined this set, eliminating 11 entries less important and resulting in a distilled summary of 18 key log entries, with notable entries including "Port Down" and "LACP".

This summarized information was then combined with crafted instruction and input into the fine-tuned LLM. As shown in the upper part of Fig. 8, *LogInsight* accurately diagnosed the fault, identifying the root cause as port-down events triggered by Loss of Signal (LOS) alarms and state changes in LACP and Smartgroup.

Case 2. Power Supply Fault: In this case, 365 log entries related to the power supply issue were gathered. Following preprocessing and summarization by the FOLS module, a fault summary containing 23 significant log entries was created.

Based on this summary, *LogInsight* accurately identified the fault type, as depicted in the lower part of Fig. 8. The output described power anomalies and voltage irregularities, offering O&M engineers detailed and actionable information about the fault.

Overall, these case studies demonstrate the fault summarization capabilities of the FOLS module and highlight *LogInsight*'s effectiveness in providing interpretable fault diagnosis results, which can significantly enhance operational decision-making.

VII. Discussion

A. Lessons Learned

LLM Context Limitations: To assess the impact of LLM context length constraints, we analyzed the log lengths in our datasets. As shown in Table VIII, we report the average and maximum number of log entries, as well as the average log length per fault case. While the average log length is moderate for Dataset 1 and Dataset 3, the maximum log length in many cases far exceeds the LLM's context window of 4096 tokens. In fact, for the majority of fault cases, the raw log data cannot fit within the model's input limit. This is a common challenge for LLM-based systems, as exceeding the token limit can lead to incomplete or truncated analysis and loss of critical diagnostic information. To address this, effective log summarization is essential. Our FOLS module is specifically designed to extract and condense the most informative log content, ensuring robust and efficient fault diagnosis even when log sequences are exceptionally long.

TABLE VIII
Log Length Statistics for Each Fault Case

Dataset	Avg. Log Length	Avg. Log Entries	Max. Log Entries
Dataset 1	7.84	127.88	4996
Dataset 2	5.67	16882.10	58056
Dataset 3	12.41	98.10	37600

Advantages of Fine-Tuning: To justify our choice of a fine-tune based approach, we compared prompt-based, RAG-based, and fine-tuned *LogInsight*. For the prompt-based method, we used the same prompts on the base LLM without fine-tuning. For RAG, we retrieved the top 5 similar historical cases using SentenceTransformer [56] and cosine similarity, and provided them as in-context examples to the LLM. As shown in Table IX, the fine-tuned method significantly outperforms both alternatives. Fine-tuning enables the LLM to internalize domain-specific knowledge and follow task instructions more accurately, resulting in higher-quality and more structured outputs. In contrast, the base LLM (with or without RAG) often fails to follow instructions, generates irrelevant or unstructured content, and struggles to produce outputs in the required format. This demonstrates that, for specialized tasks like log fault diagnosis, fine-tuning is essential for reliable and actionable results.

B. Threats to Validity

Two potential threats may affect the validity of *LogInsight*:

Requirement for Labeled Data: LogInsight requires labeled fault cases for fine-tuning, which may raise concerns regarding the cost and feasibility of data annotation. However, our empirical results demonstrate that effective fine-tuning can be achieved with a relatively small number of labeled instances, provided that all major fault types are represented. In our experiments, only 50 labeled fault cases per dataset were used, with labels provided by experienced operations engineers. For these experts, categorizing faults is a routine task, and the labeling effort is minimal. Thus, while the need for labeled data is a limitation, the burden of data collection and annotation is relatively low, making the approach practical for real-world deployment.

Manual Effort: To clarify the effort required for explanation generation, *LogInsight* first uses GPT-4 to automatically generate candidate explanations, which are then reviewed by experts for accuracy. This process significantly reduces the manual workload, as experts are not required to craft explanations from scratch but merely assess and confirm the generated outputs. In our experience, two engineers were able to review 50 cases in about one hour. Moreover, this review process is a one-time effort associated with model training rather than an ongoing

TABLE IX
PERFORMANCE COMPARISON OF FINE-TUNING, PROMPT-BASED, AND RAG METHODS

LLM	Dataset 1			Dataset 2			Dataset 3		
	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1	Micro F1	Macro F1	Weighted F1
LogInsight	**0.884**	**0.883**	**0.883**	**0.998**	**0.998**	**0.997**	**0.997**	**0.997**	**0.997**
Prompt-Based	0.451	0.218	0.508	0.302	0.212	0.292	0.489	0.483	0.591
RAG-Based	0.247	0.209	0.302	0.419	0.356	0.461	0.084	0.050	0.141

requirement. As a result, the overall annotation burden is limited and practical for real-world deployment and future scaling.

C. Limitations

Based on the experimental results, we identify two key limitations of *LogInsight* and try to suggest possible solutions:

Challenges of Similar Log Patterns: Some misclassified cases arise from unclear boundaries between certain fault types. For example, in Dataset 3, which consists of network device log data, "Port Failure" and "LACP Flapping" often exhibit highly similar log patterns. This similarity occurs because a port failure can also lead to LACP instability, resulting in overlapping log features for both fault types. Consequently, the model struggles to accurately differentiate between these types using log data alone. To address this limitation, future work could explore the integration of additional data modalities, such as system performance metrics or call traces, to provide richer contextual information.

Challenges with Unknown Fault Types: In the current implementation of *LogInsight*, the large language model (LLM) is guided by a prompt that explicitly enumerates all known fault types for a given use case, enabling it to classify faults with high accuracy.However, in real-world industrial environments, previously unseen or unknown fault types frequently emerge. To mitigate this issue, the prompt includes instructions for the LLM to classify any unrecognized fault as an "unknown type." Despite these precautions, our experimental findings reveal that the model often misclassifies unknown faults as one of the predefined types, rather than correctly identifying them as unknown. This limitation underscores the challenge of deploying *LogInsight*in dynamic environments where new fault types may appear over time. To enhance the robustness of the system, future research will investigate alternative fine-tuning strategies. For instance, incorporating external knowledge bases or leveraging self-supervised learning approaches, rather than relying solely on prompt engineering, may help the model more effectively recognize and handle unknown fault types. This could reduce its dependence on predefined types and improve adaptability in evolving industrial settings.

VIII. CONCLUSION

In this paper, we present *LogInsight*, a novel framework utilizing LLMs for effective and interpretable log-based fault diagnosis. Through fine-tuning a medium-sized, open-source LLM, we integrated domain-specific knowledge. Additionally, we designed a Fault-Oriented Log Summary (FOLS) module to extract essential information from log sequences, effectively addressing LLMs' limitations in handling long input contexts, and thereby enhancing processing efficiency for long texts. Our extensive experiments on two public datasets and one production dataset validate *LogInsight*'s accuracy and interpretability.

REFERENCES

[1] W. Meng et al., "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 4739–4745.

[2] A. Mahimkar, C. E. de Andrade, R. Sinha, and G. Rana, "A composition framework for change management," in *Proc. 2021 ACM SIGCOMM Conf.*, New York, NY, USA: ACM, 2021, pp. 788–806, doi:10.1145/3452296.3472901.

[3] Y. Yuan, W. Shi, B. Liang, and B. Qin, "An approach to cloud execution failure diagnosis based on exception logs in OpenStack," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2019, pp. 124–131.

[4] J. Chen et al., "An empirical investigation of incident triage for online service systems," in *Proc. 2019 IEEE/ACM 41st Int. Conf. Softw. Eng.: Softw. Eng. Pract.*, 2019, pp. 111–120.

[5] J. Chen et al., "Continuous incident triage for large-scale online service systems," in *Proc. 34th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2020, pp. 364–375, doi:10.1109/ASE.2019.00042.

[6] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surv.*, vol. 54, no. 6, Jul. 2021, Art. no. 130, doi:10.1145/3460345.

[7] Q. Lin, H. Zhang, J.-G Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. IEEE/ACM Int. Conf. Softw. Eng. Companion*, 2016, pp. 102–111.

[8] Y. Sui et al., "LogKG: Log failure diagnosis through knowledge graph," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3493–3507, Sep./Oct. 2023.

[9] C. Zhao et al., "Robust multimodal failure detection for microservice systems," in *Proc. 29th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, New York, NY, USA: ACM, 2023, pp. 5639–5649, doi:10.1145/3580305.3599902.

[10] S. Zhang et al., "No more data silos: Unified microservice failure diagnosis with temporal knowledge graph," *IEEE Trans. Serv. Comput.*, vol. 17, no. 06, pp. 4013–4026, Nov./Dec. 2024, doi:10.1109/TSC.2024.3489444.

[11] Y. Liu, S. Tao, W. Meng, F. Yao, X. Zhao, and H. Yang, "LogPrompt: Prompt engineering towards zero-shot and interpretable log analysis," in *Proc. IEEE/ACM Int. Conf. Softw. Eng. Companion*, 2024, pp. 364–365, doi:10.1145/3639478.3643108.

[12] S. Frieder et al., "Mathematical capabilities of ChatGPT," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 27699–27744.

[13] W. Jiao, W. Wang, J.-T. Huang, X. Wang, and Z. Tu, "Is ChatGPT a good translator? A preliminary study," 2023, *arXiv:2301.08745*.

[14] Y. Peng, C. Wang, W. Wang, C. Gao, and M. R. Lyu, "Generative type inference for python," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2023, pp. 988–999.

[15] Y. Li et al., "Exploring the effectiveness of LLMs in automated logging generation: An empirical study," *IEEE Trans. Softw. Eng.*, vol. 50, no. 12.

[16] J. Qi et al., "LogGPT: Exploring ChatGPT for log-based anomaly detection," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun. Data Sci. Syst. Smart City Dependability Sensor Cloud Big Data Syst. Appl.*, 2023, pp. 273–280.

[17] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang, "LLMParser: An exploratory study on using large language models for log parsing," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.*, 2024, pp. 1–13.

[18] M. Poli et al., "Hyena hierarchy: Towards larger convolutional language models," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 28043–28078.

[19] O. Rubin and J. Berant, "Retrieval-pretrained transformer: Long-range language modeling with self-retrieval," *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 1197–1213, Sep. 2024.
[20] N. F. Liu et al., "Lost in the middle: How language models use long contexts," *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 157–173, 2024.
[21] X. Zhang et al., "Onion: Identifying incident-indicating logs for cloud systems," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1253–1263.
[22] A. Radford et al., "Improving language understanding by generative pre-training," *OpenAI*, 2018.
[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, arXiv:1810.04805.
[24] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.
[25] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.
[26] J. Achiam et al., "GPT-4 Technical report," 2023, *arXiv:2303.08774*.
[27] H. Touvron et al., "Llama: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.
[28] A. Chowdhery et al., "PaLM: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, no. 240, pp. 1–113, 2023.
[29] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[30] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.
[31] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. Natural Lang. Process.*, Aug. 2021, pp. 4582–4597. [Online]. Available: https://aclanthology.org/2021.acl-long.353/
[32] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, Nov. 2021, pp. 3045–3059. [Online]. Available: https://aclanthology.org/2021.emnlp-main.243/
[33] T. Lei et al., "Conditional adapters: Parameter-efficient transfer learning with fast inference," in *Proc. 37th Int. Conf. Neural Inf. Process. Syst.*, 2024, pp. 8152–8172.
[34] H. Zhao, J. Fu, and Z. He, "Prototype-based hyperadapter for sample-efficient multi-task tuning," 2023, *arXiv:2310.11670*.
[35] Y. Wang et al., "AdaMix: Mixture-of-adaptations for parameter-efficient model tuning," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, Abu Dhabi, United Arab Emirates: ACL, Dec. 2022, pp. 5744–5760. [Online]. Available: https://aclanthology.org/2022.emnlp-main.388/
[36] E. J. Hu et al., "LoRa: Low-rank adaptation of large language models," 2021, *arXiv:2106.09685*.
[37] V. Fomenko, H. Yu, J. Lee, S. Hsieh, and W. Chen, "A note on LoRa," 2024, *arXiv:2404.05086*.
[38] A. Vervaet, "MoniLog: An automated log-based anomaly detection system for cloud computing infrastructures," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 2739–2743.
[39] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "SwissLog: Robust and unified deep learning based log anomaly detection for diverse faults," in *Proc. IEEE Int. Symp. Softw. Rel. Eng.*, 2020, pp. 92–103.
[40] L. Liao, K. Zhu, J. Luo, J. Cai, and G. Costa, "LogBASA: Log anomaly detection based on system behavior analysis and global semantic awareness," *Int. J. Intell. Syst.*, vol. 2023, Jan. 2023, Art. no. 3777826, doi:10.1155/2023/3777826.
[41] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," in *Proc. 1999 ACM SIGMOD Int. Conf. Manage. Data*, New York, NY, USA: ACM, 1999, pp. 49–60, doi:10.1145/304182.304187.
[42] J. Xu, R. Yang, Y. Huo, C. Zhang, and P. He, "DivLog: Log parsing with prompt enhanced in-context learning," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.*, New York, NY, USA: ACM, 2024, pp. 2457–2468, doi:10.1145/3597503.3639155.
[43] Z. Jiang et al., "LILAC: Log parsing using LLMs with adaptive parsing cache," *Proc. ACM Softw. Eng.*, vol. 1, pp. 137–160, Jul. 2024.
[44] J. Liu et al., "Scalable and adaptive log-based anomaly detection with expert in the loop," 2023, *arXiv:2306.05032*.
[45] J. Pan, W. S. Liang, and Y. Yidi, "RAGLog: Log Anomaly Detection using Retrieval Augmented Generation," *2024 IEEE World Forum on Pub. Saf. Technol. (WFPST)*, Herndon, VA, USA, pp. 169–174 2024, doi:10.1109/WFPST58552.2024.00034.
[46] J. Xu et al., "UniLog: Automatic logging via LLM and in-context learning," in *Proc. IEEE/ACM 46th Int. Conf. Softw. Eng.*, New York, NY, USA: ACM, 2024, Art. no. 14, doi:10.1145/3597503.3623326.
[47] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez, "Unveiling clusters of events for alert and incident management in large-scale enterprise it," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 1630–1639.
[48] M. Ester et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 1996, pp. 226–231.
[49] J. MacQueen et al., "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, Oakland, CA, USA, 1967, pp. 281–297.
[50] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to Information Retrieval*, vol. 39. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[51] L. Zheng et al., "Judging LLM-as-a-judge with MT-bench and chatbot arena," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 46595–46623.
[52] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. 2017 IEEE Int. Conf. Web Serv. (ICWS)*, 2017, pp. 33–40.
[53] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proc. Int. Database Eng. Appl. Symp.*, 2014, pp. 366–367.
[54] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manage.*, vol. 45, no. 4, pp. 427–437, 2009.
[55] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*. Berlin, Germany: Springer, 2005.
[56] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese bert-networks," 2019, *arXiv:1908.10084*.

Yongqian Sun (Member, IEEE) received the BS degree in statistical specialty from Northwestern Polytechnical University, Xi'an, China, in 2012, and the PhD degree in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2018. He is currently an associate professor with the College of Software, Nankai University, Tianjin, China. His research interests include AIOps and service computing.

Shiyu Ma received the BS and MS degrees in software engineering from the College of Software, Nankai University, Tianjin, China, in 2022 and 2025, respectively. Her research interests include failure diagnosis and root-cause localization.

Tong Xiao received the BS degree in software engineering from the North University of China, Taiyuan, China, the MS degree in computer science from the National University of Defense Technology, Changsha, China, and the PhD degree in computer science from Hunan University, Changsha, China. He is currently a postdoctoral fellow with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include automated log analysis, AIOps, data mining, and deep learning.

Yongxin Zhao received the BS and MS degrees in software engineering from the College of Software, Nankai University, Tianjin, China, in 2021 and 2024, respectively. She is currently working toward the PhD degree with the College of Software, Nankai University, Tianjin, China. Her research interests include failure detection and diagnosis.

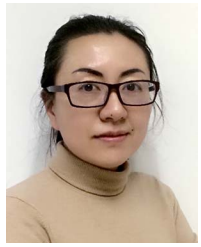
Xuhui Cai received the Master of Science degree in computer and communications from the Beijing University of Posts and Telecommunications, Beijing, China, in 1991. He is currently the manager with Network Department, China Mobile Communications Group. He has extensive experience in cloud platform maintenance.

Wei Dong received the Master of Science degree in control engineering from the Beijing Institute of Technology, Beijing, China, in 2017. He is currently the project manager with Network Department, China Mobile Communications Group. He has extensive experience in cloud platform maintenance.

Yue Shen received the Master of Science degree in computer science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2021. She is currently the project manager with Network Department, China Mobile Communications Group. She has extensive experience in cloud platform maintenance. Her research interests include cloud computing, virtualization, and algorithm design.

Yao Zhao received the BS degree in information management and information systems from the Nanjing University of Information Science and Technology, Nanjing, China, in 2015, and the MS degree in agricultural informatization from the College of Basic and Information Engineering, Yunnan Agricultural University, Kunming, China, in 2017. He is currently with Network Department, China Mobile Communications Group Jiangsu Company, Ltd. His research interests focus on the application of artificial intelligence in the telecommunications field.

Shenglin Zhang (Member, IEEE) received the BS degree in network engineering from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2012, and the PhD degree in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2017. He is currently an associate professor with the College of Software, Nankai University, Tianjin, China. His research interests include failure detection, diagnosis, and prediction for service management. He is also a member of the IEEE.



Jing Han received the master's degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China. Since 2000, she has been with ZTE Corporation. From 2000 to 2016, she was engaged in 3G/4G key technologies, and has been a technical director responsible for the intelligent operation of cloud platforms and wireless networks since 2016. Her research interests include machine learning, data mining, and signal processing.

Dan Pei (Senior Member, IEEE) received the BE and MS degrees in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1997 and 2000, respectively, and the PhD degree in computer science from the Computer Science Department, University of California, Los Angeles, Los Angeles, CA, USA, in 2005. He is currently an associate professor with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include network and service management. He is a senior member of the IEEE and ACM.