Demystifying and Extracting Fault-indicating Information from Logs for Failure Diagnosis

Junjie Huang*, Zhihan Jiang*, Jinyang Liu*, Yintong Huo*, Jiazhen Gu*, Zhuangbin Chen†, Cong Feng‡, Hui Dong‡, Zengyin Yang‡, Michael R. Lyu*
*The Chinese University of Hong Kong, China, {jjhuang23, zhjiang22, jyliu, ythuo, jiazhengu, lyu}@cse.cuhk.edu.hk
†Sun Yat-sen University, China, {chenzhb36@mail.sysu.edu.cn}
‡Huawei Cloud Computing Technology Co., Ltd, China, {fengcong5, donghui25, yangzengyin}@huawei.com

***Abstract*—Logs are imperative in the maintenance of online service systems, which often encompass important information for effective failure mitigation. While existing anomaly detection methodologies facilitate the identification of anomalous logs within extensive runtime data, manual investigation of log messages by engineers remains essential to comprehend faults, which is labor-intensive and error-prone. Upon examining the log-based troubleshooting practices at CloudA[1], we find that engineers typically prioritize two categories of log information for diagnosis. These include *fault-indicating descriptions*, which record abnormal system events, and *fault-indicating parameters*, which specify the associated entities. Motivated by this finding, we propose an approach to automatically extract such fault-indicating information from logs for fault diagnosis, named LoFI. LoFI comprises two key stages. In the first stage, LoFI performs coarse-grained filtering to collect logs related to the faults based on semantic similarity. In the second stage, LoFI leverages a pre-trained language model with a novel prompt-based tuning method to extract fine-grained information of interest from the collected logs. We evaluate LoFI on logs collected from Apache Spark and an industrial dataset from CloudA. The experimental results demonstrate that LoFI outperforms all baseline methods by a significant margin, achieving an absolute improvement of 25.8~37.9 in F1 over the best baseline method, ChatGPT. This highlights the effectiveness of LoFI in recognizing fault-indicating information. Furthermore, the successful deployment of LoFI at CloudA and user studies validate the utility of our method[2].**

***Index Terms*—log analysis, failure diagnosis, fault-indicating information, cloud service system**

I. Introduction

As software systems grow in complexity, especially for online services with hundreds of distributed components serving global users, the challenge of preventing failures intensifies. Despite extensive efforts, these systems still encounter large-scale unplanned interruptions and service quality degradation [1]–[4]. To enhance user experience and minimize economic losses, IT companies must promptly and effectively respond to failures, thereby ensuring reliability of their software.

Logs, which document a variety of software runtime events, have been widely acknowledged as a crucial resource for diagnosing failures in online service systems [5]–[8]. For example, an empirical study on a commercial bank service revealed that at least 31% of failure diagnosis practices depend on logs [9]. However, the sheer scale and complexity of modern software lead to the generation of massive logs from multiple services, making it challenging for engineers to promptly examine logs and gain actionable insights about faults [10], [11].

To facilitate rapid diagnosis, various log analysis approaches have been proposed. These approaches strive to identify and extract valuable information from massive log data in diverse granularities, aiming to reduce manual efforts in fault localization and investigation [9], [12]–[14]. Although they have made significant progress, the information they yield can still be extensive or irrelevant to faults, hindering the direct usage for system troubleshooting. For example, log-based anomaly detection [12], [15]–[17] can identify anomalous log sessions (*e.g.*, an input list of logs that the model deems as anomalous) from a large volume of streaming logs. However, even though the identified sessions are small in time (*e.g.*, 10s), they could still contain hundreds of logs, with only a small portion being relevant to the fault [18]. Log clustering [8], [10], [19] takes one step further to filter out irrelevant logs and localize the incident-indicating ones. This is done by first grouping similar logs and then selecting the representatives. However, the semi-structured nature of log messages can make them intricate and include redundant information about system execution. As a result, it requires additional manual efforts to comprehend various parts of a log message. Log parsing [20]–[22], on the other hand, can extract different elements of a raw log, such as log events and parameters. Yet, it remains unknown whether an event describes a critical failure symptom or a parameter captures the faulty component (e.g., device or VM). Semantic-aware log parsers [23]–[25] take a step further to identify parameter types (*e.g.*, object ID and type indicator) during parsing. However, the identified parameter types cannot always relate to faulty components. As a result, there is still a lack of tools that can automatically extract more precise and crucial information from logs to guide engineers in taking immediate actions for fault diagnosis.

In this paper, we propose to extract *fault-indicating information* from logs, namely the parts of logs that convey direct and valuable insights into system faults. Utilizing this information, engineers can quickly understand the underlying issues and examine the faulty components for effective fault mitigation. To figure out what information is fault-indicating, we first conduct

†Zhuangbin Chen is the corresponding author.

[1]Due to the company policy, we anonymize the name as CloudA.

[2]The code and data are available at https://github.com/Jun-jie-Huang/LoFI

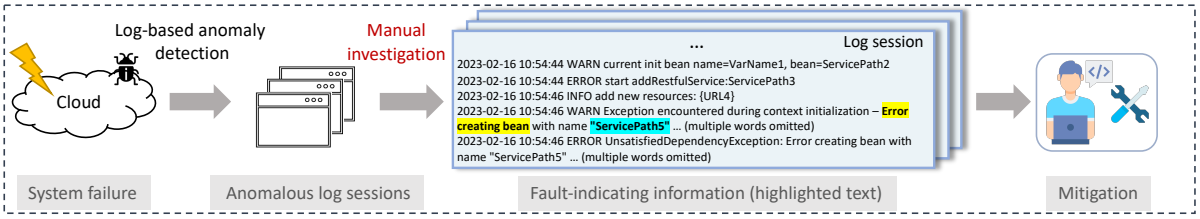

Fig. 1. The workflow of failure diagnosis with logs.

a preliminary study based on the fault mitigation practices at CloudA, a top-tier global cloud vendor. By analyzing historical fault diagnosis reports and the associated logs, we summarize two categories of fault-indicating information that are frequently employed in troubleshooting, *i.e.*, fault-indicating descriptions (FID) and fault-indicating parameters (FIP). FID describes the symptoms of a fault, which is further divided into four subtypes including error message, missing component, abnormal behavior, and wrong status. While FIP pinpoints the exact location of the fault that demands investigation, which have three typical subtypes, *i.e.*, address, component ID, and parameter name. As illustrated by the example in Figure 1, the FID of the anomalous session is "Error creating bean", and the FIP is "ServicePath5".

Designing an automated tool to extract fault-indicating information from log messages presents two key challenges. (1) The first challenge relates to the vast volume of log messages generated by modern online service systems. For example, these systems can produce up to 200 million lines of logs per hour [26]. Although anomaly detection offers potential benefits, the number of logs requiring investigation in an anomalous session can still be substantial, often ranging from hundreds to thousands [21]. Our study reveals that, for log messages in an anomalous session, only a small percentage (*i.e.*, 1.7%) contains information indicative of faults [8], [27]. (2) The second challenge involves dealing with noisy semantics in log messages, as engineers often write detailed logging statements to provide supplementary runtime information [28]. Consequently, the content of log messages tends to be verbose, with only a small portion of keywords or phrases describing the underlying issues. Our study reveals that only approximately 14.1% of the words in log messages indicate a fault or issue, posing significant challenges on the identification of target information. Overcoming these two challenges is crucial in developing an efficient and accurate tool for extracting fault-indicating information from log messages.

To address the challenges, we propose LoFI, a two-stage approach for efficient and accurate extraction of **Lo**g **F**ault-**I**ndicating information, *i.e.*, FID and FIP, to aid in fault diagnosis and reduce manual mitigation effort. LoFI mainly consists of two stages: *log selection* and *prompt-based extraction*. In the first stage, LoFI uses a coarse-grained filtering mechanism to select logs potentially relevant to faults, thereby reducing noise. This is done by first collecting logs with severe logging levels, and then incorporating more relevant logs based on their semantic similarity. By eliminating less important logs in anomalous sessions, the extraction of fine-grained information in the next stage becomes more effective and efficient. In the second stage, LoFI extracts fault-indicating information from the selected log messages by tuning a pre-trained language model (PLM). Inspired by the success of prompt learning [29], we adopt a question-answering schema to query the PLM, instructing it to iteratively yield FID and FIP. This enables us to design specific prompts that convey precise instructions about the desired information. As a result, the PLM can ignore noises within log messages and provide more accurate results.

We evaluate LoFI on logs from Apache Spark, a distributed data processing system [30], and an industrial dataset from CloudA. Given that the extraction of fault-indicating information is a novel task in log analysis, we created a benchmark dataset based on Apache Spark, namely FIBench. This is achieved by initially injecting faults [31] in to the system and then manually pinpointing fault-indicating information from the injection history. To further validate LoFI's practical significance, we also conduct experiments on an industrial dataset from CloudA, named Industry. The dataset is collected from historical postmortem reports and validated by experienced on-site engineers. Overall, LoFI achieves an F1 score of 87.4/80.6 for FID/FIP extraction on FIBench and 72.2/62.8 on Industry, significantly outperforming all baseline methods (81% on average higher than the best baseline method, ChatGPT-ICL [32]). Our ablation experiments and user study further confirm the effectiveness and practical usefulness of our method.

This work makes the following main contributions:

- We conduct a preliminary study based on six-month-long fault diagnosis reports at CloudA (§III), and summarise two categories of fault-indicating information that provides valuable insights to on-site engineers, *i.e.*, fault-indicating description (FID) and fault-indicating parameter (FIP).
- We propose LoFI, an approach to automatically extract fault-indicating information from anomalous log sessions (§IV). LoFI utilizes a novel prompt-based tuning method to effectively learn semantic information from logs with limited training data.
- Extensive experiments are conducted on datasets collected from Apache Spark and an industrial system (§V). The results show that LoFI outperforms state-of-the-art methods, including ChatGPT, by a large margin. We also demonstrate the practical value of LoFI through a user study at CloudA.

II. Background and Motivation

A. Log Analysis for Fault Diagnosis

Online service systems typically generate logs to record runtime status for troubleshooting. A log entry generally comprises three parts: a *timestamp* recording the time that the

entry is recorded, a *level* indicating the severity of a log entry (*e.g.*, INFO, WARN and ERROR), and *content* containing human-readable information that describes the specific system status. Logs serve as an important data source for failure detection and diagnosis in software systems [10].

However, manually investigating a huge number of logs is labor-intensive and time-consuming, which can lead to prolonged fault mitigation time. Thus many methods have been proposed to identify and understand useful information from logs to expedite the process. **Log parsing** is the first step to enable automated log analysis [21], which converts raw logs into *log templates* describing events and *variables* recording dynamic runtime information. Since the log parsing results cannot be directly used for diagnosis [23], **log-based anomaly detection** methods [10], [33] are proposed to identify abnormal system behaviors from logs and reduce millions of logs to a small window for engineers to investigate. However, as revealed by recent studies [34], [35], on-site engineers still have to manually investigate the runtime failure by reading tens or even hundreds of raw and noisy logs before taking further troubleshooting measures. Considering the defects of existing methods, there is an urgent need to automatically identify the detailed and crucial information from logs in order to assist in-time diagnosis and rapid mitigation.

B. A Motivating Example

In this section, we introduce the current practices of log-based fault diagnosis at CloudA, which motivate this work. Figure 1 shows a typical workflow to diagnose a fault at CloudA. The fault in this case was caused by a flawed configuration update to the job scheduling module of service X, leading to degraded performance and a decrease in successful requests. Upon detecting the failure, on-site engineers followed a systematic approach. They first identified relevant services based on their experience, including the manage-board service, monitoring service, and database. They then retrieved logs from these services for diagnosis. In this process, anomaly detection methodologies can help pinpoint anomalous log sessions. Next, they conducted a coarse-grained search to locate critical log messages containing essential information about the fault, using keywords “kill”, “fail”, “error”, and “exception”. Subsequently, a fine-grained analysis was performed by carefully reading the logs to understand the issue and determine the root cause. For example, by reading logs from the manage-board service, engineers discovered that the bean configuration creation failed, specifically in the problematic instance ServicePath5. During investigation, this process was repeated for all services to ensure a comprehensive analysis. Finally, they identified misconfiguration as the root cause and proceeded to fix the fault and restart the service.

Based on the aforementioned process, we can observe that while log-based anomaly detection expedites the coarse-grained search to identify anomalous log sessions, a more detailed and precise fine-grained investigation remains necessary for fault diagnosis. The growing scale and complexity of online service systems mean that a single fault may encompass numerous log sessions, each containing tens or even thousands of log messages. Consequently, addressing these issues requires significant manual effort. Based on the on-site engineers of CloudA, manual fault investigation through logs accounts for nearly two-thirds of the overall fault handling time. This observation aligns with the findings from recent studies [8], [36]. Thus, this work aims to address the challenge of automatically extracting fine-grained information from log sessions to enhance the process of fault diagnosis.

III. Fault-indicating Information in Logs

In this section, we aim to summarize the essential information capable of assisting engineers in taking appropriate actions and pinpointing the fault, *i.e.*, *fault-indicating information*. To this end, we conduct a preliminary study to examine how engineers leverage logs in their fault handling processes. The study comprises three steps: *dataset preparation*, *manual investigation*, and *result analysis*.

Dataset Preparation. We start by collecting historical faults and the corresponding diagnostic reports from service X, a large-scale online service system of CloudA. Service X adopts the microservice architecture with rich functionalities such as user management, analytics, resource scheduling, logging and monitoring, *etc*. The data span from 2022-09-03 to 2023-03-02, resulting in a total of 88 faults. These faults cover diverse root causes, including network disconnection, device failures, configuration errors, *etc*. For each fault, on-site engineers documented diagnosis details such as affected components, associated log files, and a fault summary that included diagnosis process, root causes, and mitigation steps. We manually investigate these reports to gain insights into how log messages are utilized during fault mitigation.

Manual Investigation. The goal of the manual investigation is to identify fault-indicating information that aids in fault diagnosis. Generally, we pinpoint such information by manually analysing diagnosis reports and corresponding log sessions to identify the log segments that are notified in the reports. The motivation of the pinpointing strategy is that only important details will be recorded in the reports, where the notified events or components are more likely to indicate faults.

Specifically, we first collect logs that are generated from around ten minutes before and after each fault given the recorded timestamps. These logs are likely to cover all system events associated with the fault. We then examine these logs and the diagnostic report to identify the specific log messages that are directly related to the fault or explicitly referred to in the mitigation step. For example, if a mitigation step mentions restarting a virtual machine with ID=6afd89eh, we will mark this ID. This process allows us to locate relevant log information used by on-site engineers in fault resolution. Two authors conduct the manual investigation separately for all collected faults. Subsequently, two senior on-site engineers from CloudA review the results for correctness. Any discrepancies are resolved through discussion until a consensus is reached.

Result Analysis. From our manual investigation, we identified two main categories of fault-indicating information frequently

TABLE I
CATEGORIES OF FAULT-INDICATING INFORMATION IN LOGS.

Category	Subtype	Example	Number
Description (FID)	Error Message	... **url detection error**!agent taskId:f292c7e596d5435d9b9e9b9f47e1f872, retCode is empty	32/88
	Missing Component	... execute template error, reason is **Host name must not be empty**	20/88
	Abnormal Behavior	... reader request line for 192.168.132.245:8080(https) failed, **read line timed-out**	24/88
	Wrong Status	... **httpCode is 404. requestEntity's type is GET**. requestEntity's url is /users/orders/task. please check!	12/88
Parameter (FIP)	Address	... **httpCode is 404. requestEntity's type is GET**. requestEntity's url is /users/orders/task. please check!	15/68
	Component ID	... **query** ... **failed**. historyid=51890bae-57c6-47a3-b37d-62df9d2f3c87	28/68
	Parameter Name	... **cannot get topicInfo for consumer by topic**: alarm_and_event_data	25/68

[1] Fault-indicating descriptions (FID) is marked in **BOLD** and fault-indicating parameters (FIP) is in UNDERLINE.

used in the mitigation process: fault-indicating description (*FID*), which primarily describes the fault, and fault-indicating parameter (*FIP*), which denotes positions or components requiring further investigation. Notably, fault-indicating parameters could only be extracted from 68 out of 88 faults, as the remaining faults do not explicitly mention parameters in the logs during mitigation. Within these two categories, we have further classified FID and FIP into four and three subtypes, respectively, based on their specific content as shown in Table I. We introduce these subtypes as follows:

- *Error Message* directly describes a failed action or an exception raised from a software stack.
- *Missing Component* means some components are unavailable such as devices, tasks and hosts.
- *Abnormal Behavior* indicates the degraded performance of an application *e.g.*, HTTP timeout, slow response time.
- *Wrong Status* means a specific response code is incorporated to explain the wrong event, *e.g.*, status code, error flags.
- *Address* includes a concrete URL of HTTP requests, IP address or paths to a folder.
- *Component ID* records the index for a system component *e.g.*, job ID, task ID, service ID.
- *Parameter Name* shows the key and value for a parameter *e.g.*, data name, user name.

Based on the identified fault-indicating information, we further investigate the presence of noise in the logs. Specifically, we compute the ratio of log messages containing either FID or FIP in their raw form. Our analysis reveals that, on average, only 1.7% of the log messages contain the target information. Additionally, the fault-indicating words account for only 14.1% of these messages' content. These findings indicate a significant amount of noise in the logs, potentially hindering on-site engineers' ability to identify precise information needed to address the underlying faults.

In summary, we have identified two common categories of fault-indicating information in fault diagnosis: fault-indicating description, reflecting fault symptoms, and fault-indicating parameters, indicating faulty components. Therefore, our focus is on extracting these two types of information.

IV. METHODOLOGY

In this section, we describe our method LoFI to automatically extract fault-indicating information from logs, *i.e.*, *FID* and *FIP* defined in § III. There are two major challenges in fault-indicating information extraction: 1) the log sessions produced by anomaly detection methods may still contain numerous, noisy log messages; 2) since each log message contains complex and redundant contents, it is hard to identify the fault-indicating information.

A. Problem Formulation

We formulate the problem of identifying fault-indicating information in logs as follows. The input is an anomaly session with n log messages $L = [l_1, l_2, \ldots, l_n]$. Each log, represented as $l_i = [w_1, \ldots, w_{m_i}]$, comprises a sequence of m_i tokens. The output is the fault-indicating information denoted as a tuple (d, p), where $d = [w_1, \ldots, w_{N_d}]$ represents the FID of system status and $p = [w_1, \ldots, w_{N_p}]$ denotes the FIP for localization. N_d and N_p are the token numbers of FID and FIP, respectively. The task aims to extract fault-indicating information (d, p) to enable engineers to understand the faults and locations so that they can take appropriate actions.

B. Overview

To overcome the aforementioned challenges, we propose a novel method called LoFI, which is based on a pre-trained language model (PLM) and is designed to extract fault-indicating information from anomalous log sessions produced by preceding anomaly detectors. The main idea behind LoFI is to utilize a large PLM to comprehend the semantics of log sessions and then extract the desired fault-indicating information by designing a prompt that accurately captures the intentions of on-site engineers. Specifically, the LoFI method is composed of two main stages: *log selection* and *prompt-based extraction*. In the log selection stage, LoFI selects candidate logs that are likely to be related to the fault in a coarse-grained manner. This stage helps to filter out less relevant logs, thus enabling a more effective and efficient extraction process. In the prompt-based extraction stage, LoFI extracts FID and FIP from candidate log messages. This is accomplished by querying a fine-tuned PLM, such as UniXcoder [37], with a well-designed prompt that precisely points to the fault-indicating information within the log messages. In this way, we allow the PLM to accurately return the desired information of on-site engineers.

C. Log Preprocessing

The preprocessing module converts raw log messages from a session into a formatted input for LoFI. The input part of Figure 2 depicts an example of processed log messages. Given a raw log, the preprocessing module first uses regular expressions to split an unstructured log message into timestamp, logging level, and log content and then deduplicates log contents. In our work, we keep the original log content for fault-indicating information extraction without parsing it into static log template and the corresponding dynamic variables [20],

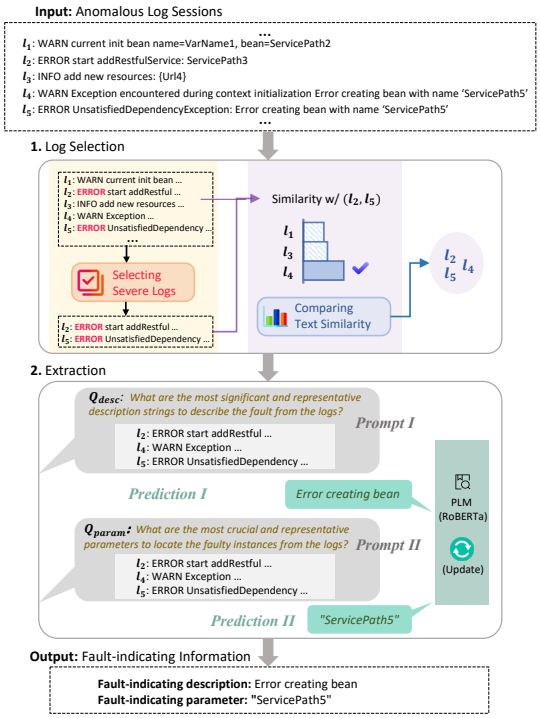

Fig. 2. The overall framework of LoFI.

since log parsing replaces common parameters with a unified identifier (*e.g.*, "$*$") which may mask informative variables (*e.g.*, device ids) and cut out the semantic relations between logs, leading to poor extraction performance.

D. Log Selection

Sessions flagged as abnormal by upstream log-based anomaly detectors often contain a large number of logs, sometimes exceeding hundreds or thousands of lines [19]. Understanding a large volume of logs is challenging for PLMs, as they often contain irrelevant information that complicates fault diagnosis. Additionally, PLMs have been shown to struggle with long texts [38], emphasizing the need for a more concise input. To address the issues, we propose a log selection approach to filter out most irrelevant log messages to the fault, which includes two steps: *level selection* (§IV-D1) and *semantic selection* (§IV-D2).

1) Level Selection.: Our study in §III shows that logs with more severe logging levels will be examined with a higher priority. Therefore, we first apply *level selection* to select logs with the highest logging level among those in a log session, which are denoted as L_{severe}; and the left logs are denoted as L_{mild}. The priority of logs is assigned by the standard ranking of logging levels defined in *Log4j* [39], *i.e.*, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, Others.

2) Semantic Selection.: However, relying solely on level selection can result in overlooking logs that, although not marked at a severe level, carry fault-indicating information. Those logs often have similar contents as severe logs, despite their less severe levels. Thus, we apply *semantic selection*, which involves calculating log embeddings and then performing similarity search to include these pertinent logs.

To produce log embeddings, we use UniXcoder [37], a PLM trained on a mixture of natural language (NL) and code corpus. While alternative PLMs such as RoBERTa [29], [40] could also be considered, we believe UniXcoder is better suited to understand log messages that contain both NL and code-like parameters, enabling producing enhanced log representations. Thus we follow [37] and use the default method to obtain log vectors by adopting the vector of the first token as the representation of the log message: $\mathbf{l_i} = \texttt{UniXcoder}(l_i)\,[0]$.

Upon obtaining log embeddings, we perform similarity search to include more relevant logs. For each log $l_i \in L_{mild}$, we compute its pairwise cosine similarity to the severe logs as $sim(l_i, l_j) = cosine(\mathbf{l_i}, \mathbf{l_j}), l_j \in L_{severe}$. We use cosine similarity as the distance metric since it measures the angle of two text vectors and is independent of their magnitude, making it suitable to measure textual semantic similarity [41]. Then we take the maximum similarity as the final score $sim(l_i, L_{severe}) = \max_{l_i \in L_{severe}}(sim(l_i, l_j))$, as we aim to highlight the most similar logs in L_{mild} to L_{severe}. Finally, we rank the logs by the score and take the top 10% logs as $L_{similar}$. We merge L_{severe} and $L_{similar}$ while preserving the original time order to create candidate logs. Despite the possibility of introducing noise, we still apply semantic selection since insufficient context could lead to further information loss. We show a comparison of selection methods in § V-C.

E. Prompt-based Extraction

After log selection, we extract fault-indicating information from the candidate logs. This is achieved by leveraging PLMs and prompt-tuning, which have been proven effective in recent log studies [29], [42]. The essence behind this idea is twofold. First, fault-indicating information has some similar semantic patterns, *e.g.*, using human-readable words to describe a failed operation or report a status code, which can be captured by PLMs through supervised signals. Second, the semantic patterns can vary across different logs, which is hard to detect by rule-based methods and cover all possible patterns. To implement this approach, we first choose a PLM (§ IV-E1) and design prompt questions for FID and FIP (§ IV-E2). Then we combine the question and logs as input to the PLM to predict position spans of target fault-indicating information (§ IV-E3). Finally, we fine-tune the PLM with a few labeled examples to improve the performance on this task (§ IV-E4).

1) Pre-trained Language Model (PLM): PLMs [40] have shown remarkable abilities to understand the semantic meaning of logs [29], [42]. In this work, we employ UniXcoder [37] as our backbone for two main reasons. Firstly, UniXcoder is trained on a blend of NL and code corpus, making it more robust and capable of comprehending logs that contain similar parametric content. Secondly, UniXcoder utilizes byte-level Byte-Pair Encoding (BPE) [43] for text tokenization, allowing it to avoid the out-of-vocabulary (OOV) problem by breaking OOV words into subwords.

2) Prompt Tuning with Questions Answering: Prompt tuning is an effective method to apply PLM to downstream tasks by adding NL instructions to the input, which can

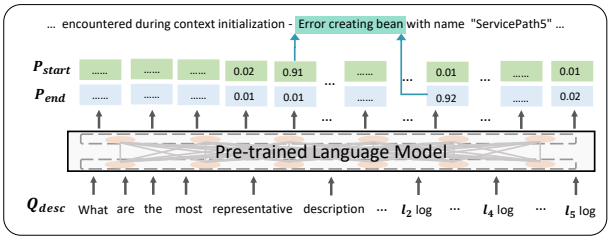

Fig. 3. Extracting fault-indicating information with the extraction module.

better utilize the knowledge in PLMs [44]. In this work, we transform fault-indicating information extraction into a question-answering task [45] by adding a prompt question. This approach enables UniXcoder to predict both FID and FIP from logs using a single unified model, eliminating the need for separate modules and enabling the sharing of common knowledge between the two types of information.

To effectively guide the PLMs, we design different prompt questions to identify FID and FIP. Specifically, for FID, the prompt question is *"What are the most significant and representative description strings to describe the fault from the logs?"* For FIP, the prompt question is *"What are the most crucial and representative parameters to locate the faulty instances from the logs?"*

3) Predicting Span to Extract Fault-indicating Information: We use a span prediction mechanism [46] to extract FID and FIP spans with the PLM. This method, commonly used in question answering tasks [45], predicts the start and end positions of the target span. By adding two additional vectors during fine-tuning, one for training a start position classifier and another for training an end position classifier, this method effectively aligns pre-training knowledge with our task requirements while maintaining a low parameter count. which is particularly beneficial when training data is scarce.

Figure 3 shows the extraction process to obtain one span. Given a prompt question and logs $L_{severe} + L_{similar}$, we concatenate them with a special delimiter token `[SEP]` to obtain a single packed input sequence X to UniXcoder:

$$X = \texttt{[CLS]}\ Question\ \texttt{[SEP]}\ l_1\ \texttt{[SEP]}\ ...\ l_i\ \texttt{[SEP]} \quad (1)$$

The input sequence X is then fed into UniXcoder to obtain hidden representations $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of tokens and d is the embedding dimension. We introduce a start vector $\mathbf{s} \in \mathbb{R}^d$ and an end vector $\mathbf{e} \in \mathbb{R}^d$ for prediction. The probability of token i being the start of the answer span is computed as a dot product between its representation $\mathbf{X}_{[i,:]}$ and $\mathbf{s}$ followed by a softmax function over all tokens: $P_{start} = \frac{e^{\mathbf{s} \cdot \mathbf{X}_{[i,:]}}}{\sum_j e^{\mathbf{s} \cdot \mathbf{X}_{[j,:]}}}$. Similarly, the probability of the end of a answer span can be computed by the analogous formula. The probability of a candidate span from position i to j is the product of start and end position probability: $P_{i,j} = P_i \cdot P_j$. Tokens in top-k scoring spans where $j >= i$ are concatenated as the prediction.

4) Training Objective: We use the start position X_{start} and end position X_{end} of ground truth FID or FIP as the target positions for each example X. The UniXcoder is trained to maximize the probability of the correct answer span, which is the sum of log-likelihoods of the correct start and end positions. The loss function is computed as follows:

$$Loss = -\frac{1}{N} \sum_N (\log P_{X_{start}} + \log P_{X_{end}}), \quad (2)$$

where N is the number of labeled training examples.

F. Online Extraction

Identifying fault-indicating information from a large volume of logs can greatly assist engineers in monitoring system status and reducing diagnosis efforts in real-world software maintenance. Here, we apply LoFI to extract FID/FIP in an online setting, where logs are continuously produced by software systems. To simulate the online phase, we first perform automatic log anomaly detection to prepare anomalous log sessions and then apply LoFI to extract FID and FIP from the sessions. We take a simple yet effective method to detect anomalies with Decision Tree (DT) [47], which can efficiently scale to millions of logs in a short period of time [15]. Specially, we parse the logs into log templates with Drain [20] and compute a count feature vector for each session of logs. The count vector is subsequently used as input for DT, which is trained with 100 manually annotated anomalous sessions as positive examples. After anomaly detection, we use LoFI to predict FID and FIP. Note that we only perform extraction for potentially anomalous log sessions, which is still efficient when scaled to a large volume of streaming logs.

V. Evaluation

We evaluate our method by answering the following research questions (RQs):

- RQ1: How effective is LoFI in the *offline* setting?
- RQ2: How log selection affects the results of LoFI?
- RQ3: How prompt-based tuning affects the results of LoFI?
- RQ4: How LoFI helps SREs to diagnose in *online* setting?

A. Experiment Designs

1) Dataset: To evaluate the effectiveness of LoFI in extracting fault-indicating information from logs, we conduct experiments on two datasets, which are collected from a public software Apache Spark and a large-scale industrial system.

FIBench. We collect FIBench from Apache Spark, a widely-used distributed framework for big data processing [30], which produces extensive log data to record its runtime information. [31] conducted fault injection to Spark to gather logs in both normal and anomalous running states for log-based anomaly detection. They injected 21 types of faults, including network issues, process killing, *etc*. However, the labeled anomalous log sections still contain numerous logs without FID and FIP. Building on the work [31], we further annotate the FID and FIP within these logs by examining the fault-injecting steps. Specifically, we focus on the logs containing entities that were first operated for fault injection. Next, by referring to the categories of fault-indicating information outlined in Table I, we can identify the FID and FIP in logs accurately. For instance, if a process in a physical machine is terminated, we can associate the process ID and machine's IP address with the logs, thereby identifying the corresponding FID and

FIP. Finally, we obtained 71 fault cases with explicit fault-indicating information.

Industry. To confirm LoFI's practical significance, we also collect an industrial dataset with 88 fault cases from 33 microservices within the production system of CloudA. The detailed collection process of Industry can be found in §III. Table II shows the statistics of our dataset.

TABLE II
DATASET STATISTICS

Dataset	Total Logs	Logs per Session	Faults	FID	FIP
FIBench	1,225,287	39.9	71	71	37
Industry	2,721,013	64.3	88	88	68

2) Baseline Methods: We compare LoFI with the following baseline methods on fault-indicating information extraction.

- TF-IDF [48] is a keyword extraction method based on word frequency. We treat a log session as a corpus and each log as a document. The importance weights of words are calculated using TF-IDF, and the top-k words with the highest weights are chosen as FID and FIP. We set $k = 6$ as is close to the average length of FID and FIP.
- TextRank [49] is a summarization method based on textual similarity. We use Log2Vec [50] to produce word embeddings, and then rank the words based on the importance scores computed by TextRank. We use the top-k words with the highest score as FID and FIP where $k = 6$.
- LogSummary [35] is a log summarization method, which first extracts summaries in triple format from log templates and then ranks the triples based on Log2Vec [50]. We obtain the highest scoring one as the FID and FIP.
- ChatGPT [32] is a cutting-edge conversational AI based on large language models, which has been widely acknowledged for its groundbreaking abilities in text understanding. Specially, we consider a zero-shot and an In Context Learning (ICL) setting, where the former construct the prompt input with questions (§ IV-E2) and logs, and the latter additionally incorporates one more example as demonstrations. We use OpenAI APIs to obtain responses. Due to space constraints, we show the prompt in our repository [51].

3) Metrics: Log fault-indicating information extraction aims to automatically identify the correct FID and FIP strings in a format of word tokens from logs, which is a new task in log analysis. To evaluate the task, we utilize the *F1 score (F1)*, which is widely used to measure the correctness of extracted answer strings with the reference [45], [52]. F1 score measures the average overlap between the predictions and ground truth FID or FIP. As the F1 score is computed the same as the Rouge-1 [53], a popular metric to evaluate textual summarization, we simply use F1 here. We treat each prediction and reference as bags of words and compute the *Precision*, *Recall*, and *F1-score* of each example as follows: $precision = \frac{\#\ shared\ tokens}{\#\ prediction\ tokens}$, $recall = \frac{\#\ shared\ tokens}{\#\ ground\ truth\ tokens}$, $F1 = \frac{2 \times precision \times recall}{precision + recall}$. Finally, the scores are averaged over all examples for FID and FIP, respectively.

4) Implementation Details: We conduct our experiments on a Linux GPU server with Intel Xeon 2.3GHz CPU and NVIDIA Tesla V100 16G GPU. We implement LoFI with Python 3.9, PyTorch 2.0 and transformers 4.26.1. For the log selection module, we set 10 seconds as the default time to collect a session of anomalous logs. When training the pre-trained language models, we use AdamW [54] optimizer with a learning rate of 5e-5 and linear scheduling with 5% warm-up. The maximum input token length for UniXcoder is 512. We set the training batch size as 8 and train the model for 100 epochs. We merge the top-3 scoring spans to get predictions. In the online stage, we set the inference batch to 32. The time period for online anomaly detection is 10 seconds.

B. RQ1: Effectiveness of LoFI

Setup. In this RQ, we evaluate the effectiveness of LoFI in extracting fault-indicating information in an offline setting by comparing it with a range of baseline models. To simulate the process, we assume the anomalies are all correctly detected and construct the log sessions with a time period of 10 seconds. We fine-tune the models on the training set with randomly sampled 32 cases of FIBench and Industry and evaluate on the remaining cases, respectively.

Results. The evaluation results in terms of precision, recall, and F1 scores are shown in Table III. From the results, we can find that: (1) LoFI achieves high accuracy in recognizing fault-indicating information, with F1 scores of 87.4/80.6 for FID/FIP on FIBench and 72.2/62.8 for FID/FIP on Industry, which show the effectiveness of our method. LoFI's superior results stem from its design, which combines an efficient log selection algorithm, a robust pretrained language model, and a context-aware prompt-based tuning mechanism. Further comparisons of the three components can be found in § V-C and § V-D. (2) LoFI significantly outperforms all baseline methods in both FIBench and Industry for FID and FIP, across all evaluation metrics. Specifically, in terms of F1, LoFI surpasses the strongest baseline *ChatGPT-ICL* by 81% (an average of 75.8 over 41.9 across two datasets and two fault-indicating information). This demonstrates LoFI's substantial superiority over existing baselines. Unsupervised methods, such as TF-IDF, TextRank, and LogSummary, which do not use pre-trained language models, struggle to understand the log semantics, leading to low F1 scores in recognizing fault-indicating information across both datasets. Among unsupervised methods, ChatGPT-Zeroshot stands out, which is probably because ChatGPT is a powerful large language models and can understand the log and question semantics. Although incorporating the in-context learning mechanism boosts ChatGPT-ICL's performance, it still falls short of LoFI by a large margin. The performance gap further emphasizes LoFI's superiority. (3) Comparing the results on FIBench and Industry, we find the overall performance on FIBench is better than that on Industry (84 versus 67.5 on average of F1 for LoFI). This is probably because Industry is collected from an industrial application which has a more diverse set of faults and more complicated logs. (4) Comparing the results of FID prediction and FIP prediction on both datasets, we find that the overall accuracy of FID is higher than FIP (79.8 versus 71.7 on average of F1). This can be attributed to the difference in word distribution between FID and FIP, where

TABLE III
EXPERIMENTAL RESULTS (%) OF LOG FAULT-INDICATING INFORMATION EXTRACTION

Method	FIBench-FID			FIBench-FIP			Industry-FID			Industry-FIP		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
TF-IDF*	3.4	2.6	2.8	2.8	1.4	1.8	5.3	4.2	4.4	2.5	2.4	2.4
TextRank*	12.8	12.8	12.3	0.0	0.0	0.0	17.9	18.0	17.1	5.2	3.6	4.2
LogSummary*	4.5	5.4	4.5	3.6	1.1	1.7	16.2	13.5	14.2	6.9	4.9	5.5
ChatGPT-Zeroshot*	59.6	30.1	38.2	9.7	1.3	2.2	47.2	29.9	33.2	32.1	33.3	32.2
ChatGPT-ICL	53.3	51.6	49.6	46.5	44.4	44.9	45.1	33.3	35.9	41.3	38.3	37.0
LoFI (ours)	**87.4**	**87.6**	**87.4**	**80.6**	**80.6**	**80.6**	**73.8**	**72.0**	**72.2**	**70.0**	**60.9**	**62.8**

[1] We use $*$ to denote unsupervised methods, others are supervised ones.

TABLE IV
RESULTS OF DIFFERENT LOG SELECTION (LS) METHODS (%)

	FIBench				Industry			
	CR	Acc.	F1-FID	F1-FIP	CR	Acc.	F1-FID	F1-FIP
Full LoFI	39.9	**100.0**	**87.4**	**80.6**	62.7	98.2	**72.2**	**62.8**
- LS=HighestCtx	38.1	87.2	77.6	78.2	59.1	91.1	67.6	58.9
- LS=Highest	34.5	76.9	69.9	72.2	48.2	91.1	67.0	60.8
- LS=ErrorWarn	7.9	71.8	65.4	59.3	86.4	**100.0**	61.0	29.5
- LS=Error	4.0	23.1	16.7	13.9	44.6	85.7	64.0	56.5
- w/o LS	-	-	76.9	75.6	-	-	50.2	43.7

the former are mainly written in natural language, and the latter often contain parametric words like HTTP requests and user IDs in hexadecimal format, which is more difficult to recognize. Despite LoFI is based on UniXcoder, a language model pretrained with mixture of natural language and code, it still performs poorer on FIP than FID.

C. RQ2: Impacts of Log Selection

Setup. In this study, we examine the impacts of log selection (LS) in LoFI on the performance of fault-indicating information extraction. The LS module first selects severe logs based on their logging levels and then collects related logs based on embedding similarity. To this end, we completely remove LS module and substitute it with four different LS methods. Next, we fine-tune the PLMs using varied inputs on the same dataset split as RQ1, and compare LoFI's performance against these LS variants. Besides F1 scores, we also report the selection accuracy (Acc.), considering an example correct if the chosen logs carry FID and FIP, and the compression ratio (CR), which is the percentage of log lines remaining after LS.

- w/o LS: The log selection module is removed.
- LS=Error: Logs with the ERROR logging level are selected.
- LS=ErrorWarn: Logs with ERROR or WARN levels.
- LS=Highest: Logs with the highest logging level.
- LS=HighestCtx: Logs with the highest level are initially selected. Contextual logs, which appear immediately before and after these logs, are then merged with the first step.

Results. The results in Table IV shows that: (1) Our proposed log selection method in the full LoFI model outperforms in five out of six metrics, highlighting its superiority. Specially, when comparing full LoFI to LoFI with LS=Highest and LS=HighestCtx, we find that mining semantic-similar logs can complement the hard level selection, allowing for more accurate fault-indicating information extraction by finding additional potential fault-related logs. (2) Despite using logs at ERROR and WARN levels (LS=ErrorWarn) can cover 100% FID and FIP in Industry, the performance still falls short as it struggle to filter out irrelevant logs, resulting in a high compression rate of 86.4%. As compression rates vary greatly with logging level-based filtering due to varying software logging styles, an adaptive LS strategy combining level and semantic selection is a more effective solution.

D. RQ3: Impacts of Prompt-based Tuning

Setup. LoFI utilizes prompt-based tuning with PLMs to understand the semantic meaning of log contents and predict the span of FID and FIP. In this RQ, we evaluate the effectiveness of prompt-based tuning with different prompt designs and tuning methods. We also investigate the influence of PLMs by replacing UniXcoder to various PLMs.

- Prompt=LessInfo: A less informative hard prompt is used [55], *i.e.*, *"Fault-indicating descriptions in the following logs: "* for FID and *"Fault-indicating parameters in the following logs: "* for FIP.
- w/o Prompt: The prompt question are removed and only the logs are used to fine-tune a UniXcoder model.
- w/o Tuning: Directly apply UniXcoder without finetuning to predict the span of FID and FIP with the input of prompt questions and logs.
- PLM=BERT: The PLM is replaced by BERT [46], the first PLM for language understanding.
- PLM=RoBERTa: The PLM is replaced by RoBERTa [40], an improved version of BERT with optimized hyperparameters and a larger training corpus.
- PLM=CodeBERT: The PLM is replaced by CodeBERT [56], the first PLM trained on a mix of natural language and code.

Results. The results shown in Table V reveal that: (1) Among the prompt variants, prompt question performs the best, suggesting that it provides important context for this task, thereby enhancing PLMs' ability to pinpoint relevant fault-indicating information. Notably, the absence of prompts (w/o Prompt) leads to a substantial performance drop, especially for FIP, which sees a 51.6%/37.0% F1 drop on FIBench/Industry compared to FID with only 28.8%/7.8% F1 drop. This difference in decline ratio can be attributed to the variances in word distribution between FID and FIP. Since FIP often contain technical terms and domain-specific vocabulary that are less common in the pre-training corpus, predicting them without the context provided by the prompt is more challenging. (2) LoFI shows poor performance without fine-tuning, especially with FIP. This result shows the importance of fine-tuning, which can significantly improve LoFI's performance, even when available training data is limited. (3) Among the PLMs, UniXcoder achieves the best performance. Owing to its training on a hybrid corpus of code and natural language with

TABLE V
RESULTS OF PROMPT-BASED TUNING VARIANTS (%)

	FIBench		Industry	
	F1-FID	F1-FIP	F1-FID	F1-FIP
Full LoFI	**87.4**	**80.6**	**72.2**	**62.8**
- Prompt=LessInfo	81.3	78.3	68.6	57.9
- w/o Prompt	58.6	29.0	64.4	25.8
- w/o Tuning	34.8	1.8	11.3	5.6
- PLM=CodeBERT	81.5	76.4	69.6	49.5
- PLM=RoBERTa	83.8	78.3	64.7	41.4
- PLM=BERT	77.5	78.7	65.4	14.6

improved training objectives, UniXcoder can better understand the semantics of mixed text types. Consequently, it can more effectively interpret logs, which comprise natural language and code-like parameters, thus underscoring its superiority.

E. RQ4: How LoFI Assists in Online Diagnosis?

In this RQ, we conduct a user study to evaluate LoFI's effectiveness and usefulness in an online setting, where streaming logs are continuously produced by the systems.

Setup. To simulate the diagnosis process in production environments, we train LoFI on all 88 examples in Industry and apply our online pipeline (§ IV-F) to new logs from 2023-03-03 to 2023-04-02 to extract fault-indicating information. The pipeline first identifies anomalous logs sessions using an anomaly detection algorithm, then employs LoFI for extraction. Due to company policy, the total number of anomalies is not disclosed. For the user study, 50 fault examples are randomly sampled and evaluated by ten experienced engineers from three different service teams within CloudA, averaging 3.6 years of experience.

We begin our study by showing participants with full raw logs in the anomalous session, accompanied by extracted FID and FIP. They are then asked to rate the accuracy of extracted FID and FIP (Q1-Q2). After judging all examples, they are asked if automatic fault-indicating information extraction would help (Q3). Specially, Q1 and Q2 are rated on a 5-point Likert scale [57], with participants encouraged to provide explanations for their scores. We then summarize the results.

Q1. Do FIDs accurately represent anomalous events? The average rating of the FID's accuracy was 4.34, with a majority of 30 examples scoring 5, 12 examples scoring 4, 4 scoring 3, 3 scoring 2, and 1 scoring 1. Overall, we find that participants highly acknowledged the effectiveness of FID in summarizing logs and aiding diagnosis. Apart from false predictions, some examples with low scores were explained by the participants, such as *containing redundant information* and *erroneous splitting*.

Q2. Do FIP accurately identify anomalous components? The average rating was 4.02, with a majority of 34 examples scoring 5, 5 scoring 4, and 11 scoring 1. Overall, participants were positive about FIP's correctness, though one participant noted a recurring issue with unnecessary predicted parameters:

"The description of "sql cannot be full or empty" is correct. But the example can have a void parameter since the root cause is not system-related, but user-related."

Q3. Would extracting fault-indicating information aid in fault diagnosis? Notably, all participants agreed that automated extraction of fault-indicating information from logs would help. In addition to reducing time and efforts for diagnosis, some participants comment on other benefits, such as implying root causes and following mitigation steps:

"The extracted fault-indicating descriptions can represent root causes and are useful for troubleshooting."

"The fault-indicating information can correlate to possible mitigation steps, e.g.*, when I see the description of "service does not exist" and the parameter of "serviceId=...", I realize I can first try to restart the service to mitigate."*

"When is this tool scheduled to launch? I used to spend 3-4 hours mitigating a fault, but with this tool, I'll be able to save time on checking hundreds of logs to find run-time behaviour."

Overall, engineers in CloudA acknowledge the value of identified fault-indicating information in assisting with diagnosis, reducing time and human efforts. These findings highlight the utility of automatic extraction of FID and FIP from logs, shedding light on future research to improve log analysis and fault diagnosis by mining more useful log information.

VI. INDUSTRIAL EXPERIENCE

In this section, we share our experience of applying LoFI to real-world cloud service systems in CloudA, aiming to show its usefulness. In CloudA, numerous services use logs to record system runtime behaviors during runtime, which are retrieved and analyzed when engineers diagnose a fault. However, when analysing logs, engineers face two main challenges. Firstly, due to the increasing scale and complexity of online service systems, a single fault can cause cascading failures which trigger sequential events in a short period of time, leading to multiple log sessions from various services [36], [58]. Secondly, to support diagnosis, software systems may generate extra log lines to record the specifics when problems arise [28], *e.g.*, software stack, resulting in tens to hundreds of logs per session. Consequently, despite the use of log anomaly detection to reduce logs for investigation, engineers still struggle with high log volumes. To address this, in CloudA, LoFI has been integrated into the intelligent log analysis system that serves hundreds of microservices to improve reliability. LoFI processes anomalous log sessions and extracts fault-indicating information to highlight symptoms and problematic positions. This allows engineers to swiftly understand the fault and grasp its essence, eliminating the need to read hundreds of logs and allowing a greater focus on troubleshooting and root cause analysis. In the following, we present two primary usage scenarios of identified fault-indicating information in CloudA: *rapid diagnosis* and *alert configuration*.

Rapid Diagnosis. Identifying fault-indicating information enables online service providers to diagnose and recover from incidents rapidly. Figure 1 shows the diagnosis workflow. Upon realizing a failure, engineers first determine relevant services and gather anomalous logs from these services. Then they manually review these logs to pinpoint incident-related information, *e.g.*, failure events and faulty devices, determine the root cause, and implement a mitigation plan. However,

this process can be time-consuming and potentially inaccurate, especially in large-scale cloud systems with numerous simultaneously events. LoFI automates this by extracting and highlighting FID and FIP, providing an immediate snapshot of anomalous actions and variables, thus significantly reducing the time to identify the root cause.

Alert Configuration. Another use case of LoFI is configuring alerts with summarized fault-indicating information from logs during system monitoring. Alerting is commonly used in software monitoring to indicate potential issues that need attention. The current monitoring raise alerts based on predefined conditions or thresholds, such as specific log occurrences or detected anomalies [36], [59]–[61]. However, the alert names are typically set by predefined templates, offering limited actionable guidance to engineers. LoFI streamlines this process by automatically creating alert contents with extracted fault-indicating information, which can specify the events that occurred, and the associated objects or variables. The summary provides a clear overview of the anomaly and actionable alerts to the SRE team, improving the overall efficiency of the entire incident response process.

VII. Threats to Validity

Internal Threats LoFI is designed to extract fault-indicating information from logs when anomalies occur. Consequently, it may struggle with normal log sessions due to different data distributions and absence of fault-indicating information, *e.g.*, error status codes, error messages, faulty devices, *etc*. Nevertheless, since normal logs are less critical and do not require investigation unless anomalies are present, we consider the model's poor performance with normal logs acceptable.

External Threats In this paper, the task and method are inspired by real-world fault mitigation practices with logs in CloudA. One may concern LoFI's applicability to other systems from different companies. Since logs have long been recognized as critical for software maintenance, and CloudA is a large-scale cloud company providing hundreds of online services for millions of users, we believe CloudA's practices are representative. Due to privacy and security concerns, we cannot access data from other companies, so LoFI is only evaluated on CloudA logs. However, the proposed approach can be trained with a small number of labeled data, making it feasible to apply LoFI to anomalous logs from other systems.

Another threat arises from the labeled data. LoFI relies on a small set of labeled log fault-indicating information for training and evaluation. The annotation requires engineers to manually inspect the log sessions and history mitigation records and mark the information span from the logs. Limited by engineers' varying experience, the label may not be entirely accurate. However, the engineers involved are in charge of cloud monitoring in CloudA and have rich experience in fault localization and mitigation using logs. They can also search online and discuss with colleagues to reach a consensus. Thus, we believe the amount of inaccurate labels is small (if any). By introducing extra annotated data to cover more anomalous scenarios, our method is expected to predict fault-indicating information more precisely.

VIII. Related Works

A. Logs Analysis for Failure Diagnosis

Logs are essential for diagnosing online service systems. To reduce diagnosis efforts, various tools have been proposed to mine informative content from large volumes of logs at different granularities [12], [13].

One research direction focuses on mining a small set of log messages for diagnosis. Anomaly detection [62] is a key task that identifies a session of logs deviating from normal behaviors. The sessions are usually split by a time period or fixed length and then judged normal or not based on various techniques, including traditional machine learning [9], [10], [33], [63]–[65], deep learning [66]–[69], and language models [70], [71]. As a session can contain numerous continuous logs and noise, some works propose to highlight lines that are most likely to reveal problems by clustering logs. LogFaultFlagger [72] flags problematic log lines for human inspection, while LogSed [73] uses a time-weighted control flow graph to identify problematic logs across multiple software threads. Onion [8] identifies three aspects of incident-indicating logs (*i.e.*, consistency, impact, and bilateral difference) and locate these logs with clustering. However, due to the complexity and verbosity of log content, manually examining lines of logs remains time-consuming for engineers to understand runtime status and diagnose system problems [34], [74].

Another line of research focuses on extracting useful information from log messages at a finer granularity. Log parsing [75], [76] identifies variables in logs by converting logs into static templates and dynamic variables [20], [26], [29], [77]–[81]. But these variables can not be directly used for diagnosis as the corresponding categories and importance levels are unaware [23]. To address this, SemParser [82] and VALB [23] explicitly identify variables and corresponding concept types (*e.g.*, concepts like "instance" and "server") during parsing. However, these two works are used for log parsing, and the identified variable and types may not relate to failures. LogSummary [35] is a closely related work to ours. They extract multiple triples from logs, *i.e.*, ("entity", "event", "relation"), to represent a log sequence and improve readability. However, these simple triples may not capture the complex semantics of logs, and users are still unaware of severe events and instances to help diagnosis.

B. Language Models for Log Analysis

Pre-trained language models (PLM) (*e.g.*, BERT [46]) have significantly advanced many software engineering tasks [83] due to the strong ability to learn the semantic information of the textual input. Based on the idea that logs are language sequences [68], many studies applied PLMs to log analysis. Swisslog [84] tunes BERT to encode log messages for anomaly detection. Ott et al. [85] examine the effectiveness of various PLMs for anomaly detection. NeuralLog [42] directly feeds raw logs to PLMs to avoid the issues of out-of-vocabulary words and semantic misunderstandings. Apart from anomaly detection, PLMs are also used for log parsing.

LogStamp [80] treats log parsing as a sequence labeling problem, fine-tuning BERT to judge whether tokens are parameters. LogPPT [29] explores prompt-based tuning on RoBERTa [40] to identify templates and parameters.

However, these studies fail to provide actionable insights to assist on-site engineers in diagnosing faults. Our work, LoFI trains a PLM to learn semantics from a few labeled examples with prompt-based tuning and can effectively identify fault-indicating descriptions and parameters from logs for diagnosis.

IX. Conclusion

In conclusion, this paper has addressed the challenge of extracting fault-indicating information from anomalous log sessions to aid fault diagnosis and reduce mitigation efforts in large-scale software systems. We conducted a preliminary study on log-based troubleshooting practices at CloudA, and identified two types of information engineers typically prioritize, *i.e.*, fault-indicating descriptions (FID) and fault-indicating parameters (FIP). Motivated by this finding, we proposed a two-stage approach, LoFI, which performs coarse-grained filtering in the first stage and leverages a pre-trained language model with a novel prompt-based tuning method in the second stage to extract fine-grained fault-indicating information. Our evaluation of LoFI on FIBench and industrial datasets demonstrated significant improvements over baseline methods. Additionally, our user study and successful deployment to CloudA provided further evidence for the usefulness of our proposed method.

X. Acknowledgement

The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund) and Australian Research Council (ARC) Discovery Projects (DP200102940, DP220103044) and Fundamental Research Funds for the Central Universities, Sun Yat-sen University (No. 76250-31610005).

References

[1] "Aws post-event summaries," https://aws.amazon.com/cn/premiumsupport/technology/pes/, [Online; accessed 31 April 2024].

[2] "Google cloud status dashboard," https://status.cloud.google.com/summary, [Online; accessed 31 April 2024].

[3] "Azure status history," https://azure.status.microsoft/en-us/status/history/, [Online; accessed 31 April 2024].

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

[5] Y. Li, Y. Huo, Z. Jiang, R. Zhong, P. He, Y. Su, and M. R. Lyu, "Exploring the effectiveness of llms in automated logging generation: An empirical study," *arXiv preprint arXiv:2307.05950*, 2023.

[6] Y. Li, Y. Huo, R. Zhong, Z. Jiang, J. Liu, J. Huang, J. Gu, P. He, and M. R. Lyu, "Go static: Contextualized logging statement generation," *FSE*, vol. 1, no. FSE, pp. 609–630, 2024.

[7] 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 *ESEC/FSE*, 2019, pp. 683–694.

[8] X. Zhang, Y. Xu, S. Qin, S. He, B. Qiao, Z. Li, H. Zhang, X. Li, Y. Dang, Q. Lin *et al.*, "Onion: identifying incident-indicating logs for cloud systems," in *ESEC/FSE*, 2021, pp. 1253–1263.

[9] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang *et al.*, "An empirical investigation of practical log anomaly detection for online service systems," in *ESEC/FSE*, 2021, pp. 1404–1415.

[10] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ICSE-Companion*, 2016, pp. 102–111.

[11] Z. Chen, Z. Jiang, Y. Su, M. R. Lyu, and Z. Zheng, "Tracemesh: Scalable and streaming sampling for distributed traces," *arXiv preprint arXiv:2406.06975*, 2024.

[12] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM computing surveys (CSUR)*, pp. 1–37, 2021.

[13] S. He, X. Zhang, P. He, Y. Xu, L. Li, Y. Kang, M. Ma, Y. Wei, Y. Dang, S. Rajmohan *et al.*, "An empirical study of log analysis at microsoft," in *ESEC/FSE*, 2022, pp. 1465–1476.

[14] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Lilac: Log parsing using llms with adaptive parsing cache," *FSE*, vol. 1, no. FSE, pp. 137–160, 2024.

[15] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *ISSRE*, 2016, pp. 207–218.

[16] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *arXiv preprint arXiv:2107.05908*, 2021.

[17] J. Liu, J. Huang, Y. Huo, Z. Jiang, J. Gu, Z. Chen, C. Feng, M. Yan, and M. R. Lyu, "Scalable and adaptive log-based anomaly detection with expert in the loop," *arXiv preprint arXiv:2306.05032*, 2023.

[18] Y. Huo, C. Lee, Y. Su, S. Shan, J. Liu, and M. Lyu, "Evlog: Evolving log analyzer for anomalous logs identification," *arXiv preprint arXiv:2306.01509*, 2023.

[19] N. Zhao, J. Chen, X. Peng, H. Wang, X. Wu, Y. Zhang, Z. Chen, X. Zheng, X. Nie, G. Wang *et al.*, "Understanding and handling alert storm for online service systems," in *ICSE-SEIP*, 2020, pp. 162–171.

[20] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS*, 2017, pp. 33–40.

[21] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *ICSE-SEIP*, 2019, pp. 121–130.

[22] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu, and M. R. Lyu, "A large-scale evaluation for log parsing techniques: How far are we?" in *ISSTA*, 2024.

[23] Z. Li, C. Luo, T.-H. Chen, W. Shang, S. He, Q. Lin, and D. Zhang, "Did we miss something important? studying and exploring variable-aware log abstraction," *arXiv preprint arXiv:2304.11391*, 2023.

[24] 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.

[25] J. Huang, Z. Jiang, Z. Chen, and M. R. Lyu, "Ulog: Unsupervised log parsing with large language models through log contrastive units," *arXiv preprint arXiv:2406.07174*, 2024.

[26] X. Wang, X. Zhang, L. Li, S. He, H. Zhang, Y. Liu, L. Zheng, Y. Kang, Q. Lin, Y. Dang *et al.*, "Spine: a scalable log parser with feedback guidance," in *ESEC/FSE*, 2022, pp. 1198–1208.

[27] E. Chuah, S.-h. Kuo, P. Hiew, W.-C. Tjhi, G. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne, "Diagnosing the root-causes of failures from cluster log files," in *2010 International Conference on High Performance Computing*. IEEE, 2010, pp. 1–10.

[28] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," *TOCS*, pp. 1–28, 2012.

[29] V.-H. Le and H. Zhang, "Log parsing with prompt-based few-shot learning," *arXiv preprint arXiv:2302.07435*, 2023.

[30] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, pp. 56–65, 2016.

[31] C. Lee, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu, "Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention," *arXiv preprint arXiv:2302.06914*, 2023.

[32] "Chatgpt," https://chat.openai.com/, [Online; accessed 31 April 2024].

[33] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," in *Proceedings of SOSP*, 2009, pp. 1–17.

[34] P. Dogga, K. Narasimhan, A. Sivaraman, and R. Netravali, "A system-wide debugging assistant powered by natural language processing," in *SoCC*, 2019, pp. 171–177.

[35] W. Meng, F. Zaiter, Y. Zhang, Y. Liu, S. Zhang, S. Tao, Y. Zhu, T. Han, Y. Zhao, E. Wang *et al.*, "Logsummary: Unstructured log summarization for software systems," *IEEE Transactions on Network and Service Management (TNSM)*, 2023.

[36] L. Li, X. Zhang, X. Zhao, H. Zhang, Y. Kang, P. Zhao, B. Qiao, S. He, P. Lee, J. Sun, F. Gao, L. Yang, Q. Lin, S. Rajmohan, Z. Xu, and D. Zhang, "Fighting the fog of war: Automated incident detection for cloud systems," in *USENIX ATC*, 2021, pp. 131–146.

[37] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," in *ACL*, 2022, pp. 7212–7225.

[38] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[39] "Apache log4j 2: Description and download," https://logging.apache.org/log4j/2.x/, 2021, [Online; accessed 31 April 2024].

[40] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

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

[42] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *ASE*. IEEE, 2021, pp. 492–504.

[43] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *ACL*, 2016, pp. 1715–1725.

[44] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys (CSUR)*, pp. 1–35, 2023.

[45] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," in *EMNLP*, 2016, pp. 2383–2392.

[46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-NLT*, 2019, pp. 4171–4186.

[47] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, pp. 1–37, 2008.

[48] S. Lee and H.-j. Kim, "News keyword extraction for topic tracking," in *2008 fourth international conference on networked computing and advanced information management*, 2008, pp. 554–559.

[49] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *EMNLP*, 2004, pp. 404–411.

[50] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *CCS*, 2019, pp. 1777–1794.

[51] "Lofi," https://github.com/Jun-jie-Huang/LoFI/.

[52] M. Glass, A. Gliozzo, R. Chakravarti, A. Ferritto, L. Pan, G. S. Bhargav, D. Garg, and A. Sil, "Span selection pre-training for question answering," in *ACL*, 2020, pp. 2773–2782.

[53] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[54] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations (ICLR)*.

[55] T. Gao, A. Fisch, and D. Chen, "Making pre-trained language models better few-shot learners," in *ACL-IJCNLP*, 2021, pp. 3816–3830.

[56] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Findings of EMNLP*, 2020, pp. 1536–1547.

[57] S. McLeod, "Likert scale definition, examples and analysis," 2008.

[58] T. Yang, J. Shen, Y. Su, X. Ling, Y. Yang, and M. R. Lyu, "Aid: efficient prediction of aggregated intensity of dependency in large-scale cloud systems," in *ASE*. IEEE, 2021, pp. 653–665.

[59] X. Li, G. Yu, P. Chen, H. Chen, and Z. Chen, "Going through the life cycle of faults in clouds: Guidelines on fault handling," in *ISSRE*. IEEE, 2022, pp. 121–132.

[60] T. Yang, J. Shen, Y. Su, X. Ren, Y. Yang, and M. R. Lyu, "Characterizing and mitigating anti-patterns of alerts in industrial cloud systems," in *DSN*. IEEE, 2022, pp. 393–401.

[61] J. Kuang, J. Liu, J. Huang, R. Zhong, J. Gu, L. Yu, R. Tan, Z. Yang, and M. R. Lyu, "Knowledge-aware alert aggregation in large-scale cloud systems: a hybrid approach," in *ICSE-SEIP*, 2024, pp. 369–380.

[62] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *ICSE*, 2022, pp. 1356–1367.

[63] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM)*, 2007, pp. 583–588.

[64] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *ICSE*, 2021, pp. 1448–1460.

[65] B. Zhang, H. Zhang, V.-H. Le, P. Moscato, and A. Zhang, "Semi-supervised and unsupervised anomaly detection by mining numerical workflow relations from system logs," *ASE*, p. 4, 2023.

[66] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017, pp. 1285–1298.

[67] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1196–1201.

[68] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *ESEC/FSE*, 2019, pp. 807–817.

[69] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *ICSE*, 2022, pp. 623–634.

[70] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *ASE*. IEEE, 2021, pp. 492–504.

[71] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "Can language models help in system security? investigating log anomaly detection using bert," in *Proceedings of the The 20th Annual Workshop of the Australasian Language Technology Association*, 2022, pp. 139–147.

[72] A. Amar and P. C. Rigby, "Mining historical test logs to predict bugs and localize faults in the test logs," in *ICSE*. IEEE, 2019, pp. 140–151.

[73] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu, "Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 447–455.

[74] Y. Sui, Y. Zhang, J. Sun, T. Xu, S. Zhang, Z. Li, Y. Sun, F. Guo, J. Shen, Y. Zhang *et al.*, "Logkg: Log failure diagnosis through knowledge graph," *IEEE Transactions on Services Computing (TSC)*, 2023.

[75] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in *ICSE*, 2022, pp. 1095–1106.

[76] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. S. Chen, and F. Pianese, "System log parsing: A survey," *TKDE*, 2023.

[77] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *KDD*, 2009, pp. 1255–1264.

[78] S. Zhang, W. Meng, J. Bu, S. Yang, Y. Liu, D. Pei, J. Xu, Y. Chen, H. Dong, X. Qu *et al.*, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 2017, pp. 1–10.

[79] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang *et al.*, "Uniparser: A unified log parser for heterogeneous log data," in *Proceedings of the ACM Web Conference 2022 (WWW)*, 2022, pp. 1893–1901.

[80] S. Tao, W. Meng, Y. Cheng, Y. Zhu, Y. Liu, C. Du, T. Han, Y. Zhao, X. Wang, and H. Yang, "Logstamp: Automatic online log parsing based on sequence labelling," *ACM SIGMETRICS Performance Evaluation Review*, pp. 93–98, 2022.

[81] H. Dai, H. Li, C.-S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using n n-gram dictionaries," *IEEE Transactions on Software Engineering (TSE)*, pp. 879–892, 2020.

[82] Y. Huo, Y. Su, C. Lee, and M. R. Lyu, "Semparser: A semantic parser for log analysis," *arXiv preprint arXiv:2112.12636*, 2021.

[83] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," in *NeurIPS*, 2021.

[84] 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 *ISSRE*. IEEE, 2020, pp. 92–103.

[85] H. Ott, J. Bogatinovski, A. Acker, S. Nedelkoski, and O. Kao, "Robust and transferable anomaly detection in log data using pre-trained language models," in *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, 2021, pp. 19–24.