

Interpretable Online Log Analysis Using Large Language Models with Prompt Strategies

Yilun Liu¹, Shimin Tao^{1,2}, Weibin Meng¹, Jingyu Wang³, Wenbing Ma¹,
Yuhang Chen², Yanqing Zhao¹, Hao Yang¹, Yanfei Jiang¹

¹ Huawei, China

² University of Science and Technology of China, China

³ Beijing University of Posts and Telecommunications, China

{liuyilun3,taoshimin,mengweibin3,mawenbing,zhaoyanqing,yanghao30,jiangyanfei}@huawei.com
wangjingyu@bupt.edu.cn,chen97@mail.ustc.edu.cn

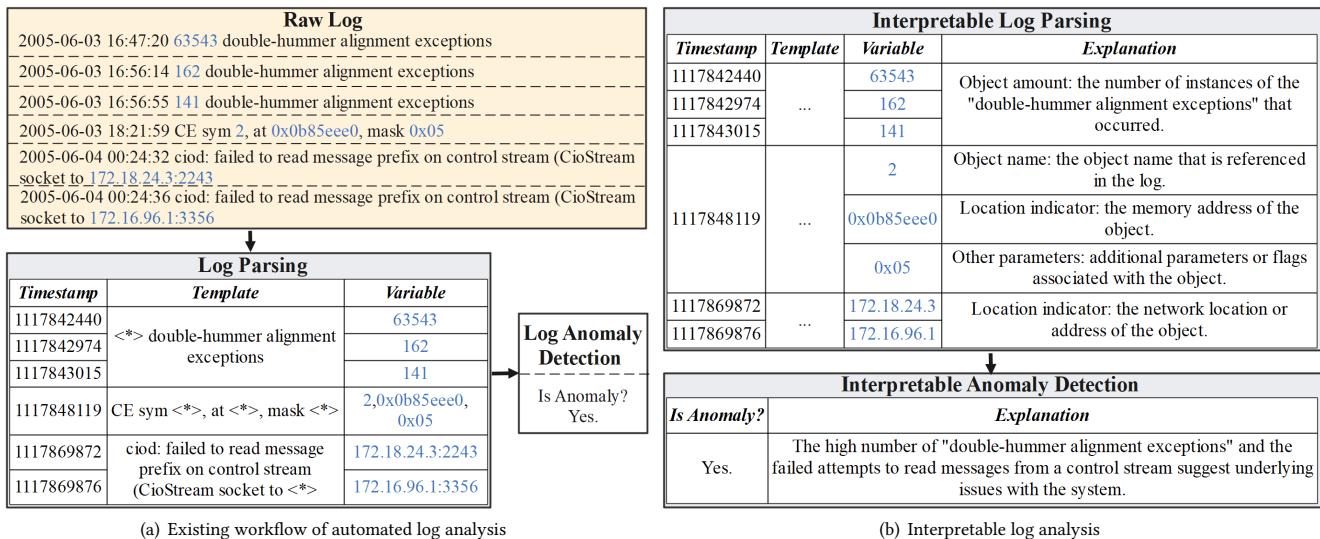


Figure 1: Illustration of automated log analysis: (a) without interpretability and (b) with interpretability. The presented explanations are generated by our approach on 2023/04/18.

ABSTRACT

Automated log analysis is crucial in modern software-intensive systems for facilitating program comprehension throughout software maintenance and engineering life cycles. Existing methods perform tasks such as log parsing and log anomaly detection by providing a single prediction value without interpretation. However, given the increasing volume of system events, the limited interpretability of analysis results hinders analysts' comprehension of program status and their ability to take appropriate actions. Moreover, these methods require substantial in-domain training data, and their performance declines sharply (by up to 62.5%) in online scenarios involving unseen logs from new domains, a common occurrence due to rapid software updates. In this paper, we propose LogPrompt, a novel interpretable log analysis approach for online scenarios. LogPrompt employs large language models (LLMs) to perform online log analysis tasks via a suite of advanced prompt strategies tailored for log tasks, which enhances LLMs' performance by up to 380.7% compared with simple prompts. Experiments on nine publicly available evaluation datasets across two tasks demonstrate that LogPrompt, despite requiring no in-domain training, outperforms existing approaches trained on thousands of logs by up to 55.9%.

We also conduct a human evaluation of LogPrompt's interpretability, with six practitioners possessing over 10 years of experience, who highly rated the generated content in terms of usefulness and readability (averagely 4.42/5). LogPrompt also exhibits remarkable compatibility with open-source and smaller-scale LLMs, making it flexible for practical deployment. Code of LogPrompt is available at <https://github.com/lunyilu/LogPrompt>.

CCS CONCEPTS

- Computer systems organization → Reliability; • Social and professional topics → Software maintenance.

KEYWORDS

large language model, prompt engineering, log analysis, interpretability, online scenario

1 INTRODUCTION

Automated log analysis is essential for ensuring the reliability and resilience of fundamental services in the development and maintenance of software-intensive systems by facilitating the comprehension of program status. For instance, a popular e-commerce website serving millions of global users daily may operate on a cluster of hundreds or thousands of servers, generating vast amounts of log data that records system failures, performance degradation, and unexpected interruptions. Traditionally, when a failure occurs in a large-scale system, Operations and Maintenance (O&M) engineers must first review copious system logs, identify abnormal events, determine root causes, and then implement appropriate mitigation measures. This process is time-consuming, requires multiple rounds of communication among engineers, and demands advanced expertise. In contrast, current automated log analysis techniques can provide efficient approaches for program comprehension, such as log abstractions (through log parsing) and anomaly detection, significantly reducing the burden on O&M engineers. As illustrated in Fig. 1(a), log parsing extracts common (static) segments (*template*) and unique (dynamic) segments (*variable*) from raw logs, minimizing log size while retaining key information; and log anomaly detection identifies anomalies in historical log sequences.

Though existing approaches have achieved remarkable performance in log analysis when combined with deep learning techniques [17, 20, 37], challenges arise when log analysis technologies are applied in real-world scenarios. First, the performance of current methods significantly declines when training samples are scarce and unseen logs are the majority, a situation common in the industry (referred as the online scenario [24, 35, 37]). In online scenarios, operators frequently perform software upgrades on services to introduce new features, resolve bugs, or enhance performance, resulting in the generation of new types of logs and the potential incompatibility of logs from older versions. Consequently, the availability of an insufficient number of historical logs often hinders effective model training. Thus, there are usually not enough historical logs to effectively train the model. In extreme cases, when a brand new service goes online, no training samples are available due to the lack of in-domain logs, necessitating an highly adaptive log analysis approach that can be applied to new domains without in-domain training.

Second, the potential benefits of log analysis techniques in improving program comprehension are hampered by their limited interpretability. Traditional methods, as depicted in Fig. 1(a), provide only prediction without further explanation. This absence of justification makes it less easier in some cases for human analysts to comprehend and act on analysis results, while an interpretable analysis output, as shown in Fig. 1(b), not only aids program comprehension with ample explanations and justifications, but also facilitates detecting false alarms, tracing root causes and taking appropriate actions.

In this paper, we aim to address two relatively unexplored challenges in log analysis, namely online scenarios and limited interpretability, by leveraging the recent advancements in large language models (LLMs). Our findings indicate that enhancing prompt strategies is crucial in unleashing the log analysis capabilities of LLMs and tackling these challenges. To this end, we propose a

novel log analysis approach called LogPrompt, which encompasses three prompt strategies: (1) self-prompt, (2) chain-of-thought (CoT) prompt, and (3) in-context prompt. Experiments on nine public evaluation datasets across log parsing and log anomaly detection tasks reveal that LogPrompt not only achieves SOTA performance without in-domain training process, but also generates useful justifications for its predictions to enhance the interpretability of log analysis. These generated justifications are rated by six senior practitioners with over 10 years of experience in software and system O&M. Our contributions can be summarized as follows:

- We present a highly adaptive online log analysis approach that requires nearly no in-domain training. While most existing approaches require training on thousands of in-domain logs, our approach outperforms them by up to 55.9% without requiring substantial in-domain training.
- We take the first step towards achieving interpretability in log analysis. In the human evaluation with six senior practitioners, over 80% of the automatically generated explanations for variables and anomalies in log analysis are considered highly useful and readable, which could aid O&M engineers in swiftly comprehending program status, composing analysis reports and ruling out false alarms.
- We propose a collection of prompt strategies designed for log analysis tasks, which consistently improve LLMs' performance by up to 380.7% compared to simple prompts. Furthermore, the compatibility of these strategies with open-source and smaller-scale LLMs suggests their potential for use in industrial applications.

Additionally, we release the code of LogPrompt, which not only encourages further exploration of advanced prompting strategies when LLMs are employed in log analysis within the research community but also facilitates actual deployments by engineers.

2 BACKGROUND

2.1 Large Language Models

Large language models (LLMs) are trained using a probabilistic approach to generate text based on diverse training corpora. More specifically, these models focus on estimating the probability of the next word given a preceding sequence of words, employing an auto-regressive process whereby tokens are predicted one at a time until a designated end token is encountered [31]. The recent advancement of LLMs has brought a significant impact across multiple domains. For instance, ChatGPT [29], a particularly powerful LLM, not only excels at various Natural Language Processing (NLP) tasks [14, 15, 30], but also provides natural language explanations to justify its decision-making process. The activation of LLMs' capabilities relies on prompts, which typically take the form of textual templates with blank slots for LLMs to fill in [21]. Studies have demonstrated that advanced prompt strategies can significantly enhance the performance of LLMs [39, 40, 44].

2.2 Pre-training & Prompting

The introduction of BERT [13] heralded a pre-training & fine-tuning paradigm within the NLP community. Such language models are pre-trained to extract robust, general-purpose language features

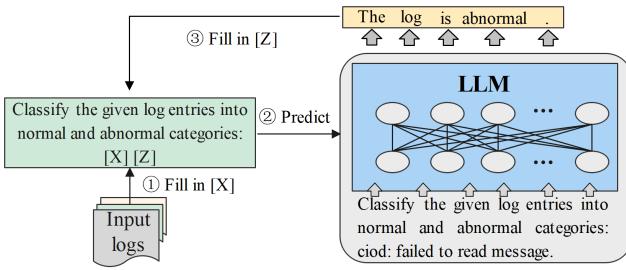


Figure 2: Illustration of utilizing a simple prompt for log anomaly detection with a large language model. [X] represents the input slot, while [Z] denotes the answer slot.

and subsequently adapted to various downstream tasks by introducing additional parameters and fine-tuning them using task-specific objective functions. However, with the substantial increase in language model scale (e.g., from 220M parameters in BERT to 175B in ChatGPT) and their enhanced capabilities, this paradigm has evolved towards pre-training & prompting [21], due to the expensive cost of fine-tuning entire models. As depicted in Fig. 2, this approach involves designing a task-specific prompt to guide the LLM in generating the desired output, which is directly input to the model without altering its parameters.

A textual prompt can be viewed as a template comprised of three components: (1) the prompt prefix, a fixed string conveying task-specific information (e.g., the prompt prefix for the simple prompt in Fig. 2 for anomaly detection is *Classify the given log entries into normal and abnormal categories:*); (2) the input slot [X], which will later incorporate one or multiple task inputs (e.g., log entries); and (3) the answer slot [Z]. Once the input slot is populated with actual input, the only dynamic component of the prompt is [Z], typically positioned at the end of the prompt. The LLM then predicts answers in an auto-regressive manner, populating the [Z] slot accordingly (e.g., with a binary word choice between “normal” and “abnormal”).

3 DESIGN OF LOGPROMPT

3.1 Motivation

(1) Insufficient interpretability. The goal of program comprehension is to facilitate engineers in understanding the program status, thereby reducing the cost for addressing breakdowns. However, as highlighted by recent studies [3, 11], existing approaches only provide simple predictions without sufficient justification and semantics, thereby impeding engineers in comprehending the analysis results easily. For instance, in Fig. 1(a), traditional methods report only the positions of variables in log parsing and binary Yes/No responses for anomaly detection, which still require efforts from practitioners to interpret and verify analysis results, especially for unfamiliar logs in online scenarios. In contrast, Fig. 1(b) demonstrates the advantages of providing explanations along with parsed variables in logs, facilitating the interpretation of program status (e.g., the variable “141” is correctly interpreted by our approach as the number of occurrences of the “double-hummer exceptions”). The justifications for detected anomalies also help comprehend the

conclusion. For example, the explanation along with the anomaly conclusion in Fig. 1(b) further interprets the anomaly in the system, attributing it to the high number of “double-hummer exceptions” and the failed read attempt from a control stream, thereby validating the credibility of the prediction.

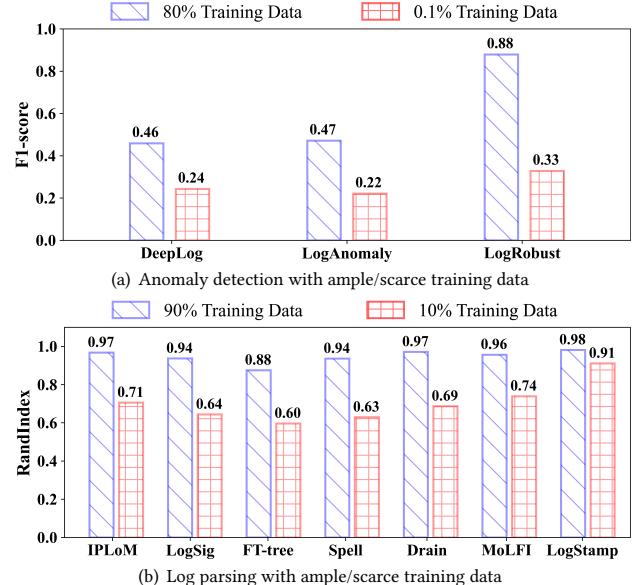


Figure 3: The average performance of existing log analysis approaches deteriorates when in-domain logs for training are limited in availability.

(2) Weak adaptability caused by data inefficiency. In modern software systems, the generation of massive logs from devices across various domains has become increasingly common [36]. These logs are highly heterogeneous in format and terminology, and are frequently updated with new log events in the online scenario. To analyze such logs, an automated log analysis approach with strong adaptability is essential to ensure steady performance when being faced with new log events. However, previous studies [37] and our preliminary experiments have shown that existing approaches perform well only when ample historical logs with correct human labels are given for training, and their performance deteriorates drastically when in-domain training data is scarce. Fig. 3 presents the mean performance of existing approaches in the two log analysis tasks, averaged across two and seven public datasets for each respective task. When the ratio of training data (chronologically splitting [35, 37]) is 80% or 90% (tested on the remaining 20% or 10% logs in each dataset), existing approaches achieve nearly 100% accuracy on both analysis tasks. However, when the available in-domain log data for training is limited to 10% or less, and unseen logs are present in the test set, their performance drops drastically (e.g., the F1-score for LogRobust drops from 0.88 to 0.33). To effectively handle new log events, existing approaches require continuous training with newly labelled logs, and some may even need to be retrained from scratch. Therefore, an log analysis approach that can directly analyze unseen logs from any domain

without the need for manually labeling massive historical logs and expensive training processes is desirable for the online scenario.

(3) Prompt strategies matter. Despite the promising potential of leveraging LLMs such as ChatGPT for interpretable online log analysis due to their powerful language understanding abilities, directly applying a simple prompt to LLMs can result in poor performance. In our preliminary experiments, ChatGPT with the simple prompt shown in Figure 2 achieved an F1-score of only 0.189 in anomaly detection. However, our best prompt strategy outperformed the simple prompt by 0.195 in F1-score. Similarly, in log parsing, we observed a performance fluctuation of up to 380.7% among different prompt strategies. Recent studies have also reported performance differences between simple and advanced prompts when LLMs are used for NLP tasks [12, 14, 30]. An unregulated prompt can lead to inaccurate results and even produce invalid content (e.g., hallucinations). Therefore, designing a task-specific prompt strategy is critical to effectively leverage LLMs to address various challenges in log analysis tasks with minimal efforts.

3.2 Overview of LogPrompt

The primary objective of LogPrompt is to enhance the correctness and interpretability of log analysis in the online scenario, through properly prompting LLMs. Firstly, LogPrompt standardizes the formats of input logs, by setting the context for slot $[X]$, and the formats of output answers ($[Z]$), in order to mitigate uncertainty in the generated solutions. Subsequently, a tailored prompt strategy is employed to construct the prompt prefix. The assembly of the prompt is completed by concatenating the prompt prefix, input slot, and answer slot, with the final response acquired from the LLM, following the procedure illustrated in Fig. 2.

We conducted experiments using three distinct strategies for constructing the prompt prefix, each founded on unique underlying principles: (1) *self-prompt*, which aims to leverage the intrinsic capabilities of the LLM to generate pertinent prompt candidates for log analysis tasks; (2) *chain-of-thought (CoT) prompt*, which emphasizes a systematic, step-by-step reasoning process by compelling the LLM to address log analysis problems through a sequence of intermediate steps, both explicitly and implicitly; (3) *in-context prompt*, which supplies multiple log examples to establish a contextual understanding for the specific task.

3.3 Format Controls

To mitigate transmission failures due to internet fluctuations when invoking LLMs, such as ChatGPT, via an API, we insert multiple input logs into the $[X]$ slot of a prompt until the length limitation for a single query is reached. However, our preliminary experiments revealed that prompts without format control statements often resulted in omitted logs, inconsistent formats, and unrelated content in the query results. Consequently, LogPrompt employs the following two functions, $f_x([X])$ and $f_z([Z])$, to establish the context for the input slot $[X]$ and the answer slot $[Z]$, respectively.

3.3.1 Input Control Function $f_x([X])$. Given N input logs, f_x constructs a context for the input slots $[X]_1 \dots [X]_N$.

$f_x([X]) = \text{There are } N \text{ logs, the logs begin: (1)}[X]_1 \backslash n (2)[X]_2 \backslash n \dots \backslash n (N)[X]_N$

3.3.2 Answer Control Function $f_z([Z], S)$. The answer control function adds prefix texts to $[Z]$, regulating the output answer format for each input log in a query. S is a text string describing the desired answer value range. For instance, in anomaly detection, S might be “a binary choice between abnormal and normal”, while in log parsing, it could be “a parsed log template”.

$f_z([Z], S) = \text{Organize your answer to be the following format: (1)}x-y \backslash n (2)x-y \backslash n \dots \backslash n (N)x-y, \text{ where } x \text{ is } S \text{ and } y \text{ is the reason.}$

3.4 Prompt Strategies

KE The task of log parsing aims to extract common segments (template) and unique segments (variable) from raw logs. For example: raw log “visible is system.call.count gt 0” is parsed into a template “visible is * gt *”. Now please specify several prompts that can make large language models precisely perform this task, where a list of input logs is parsed into their correspondent templates using * to replace the variables.

Q Here are some prompts that could be used to train a large language model to perform log parsing:

1. Parse the following log into a template format, replacing variable parts with a wildcard (*): ` [log] `
2. Convert the following log into a standardized template by identifying and replacing the variable parts with a *: ` [log] `
3. Transform the raw log ` [log] ` into a log template by replacing variable segments with *
4. Given the following log, extract its common structure and replace variable parts with * to create a log template: ` [log] `
5. Create a log template from the raw log ` [log] ` by replacing dynamic segments with *

Figure 4: Prompts advised by ChatGPT for the task of log parsing (Dialogue date: 2023/02/25).

3.4.1 Self-prompt. Drawing inspiration from the work of Jiao *et al.* [12], we seek to create prompts that activate ChatGPT’s log analysis capabilities by soliciting advice directly from ChatGPT itself. We prompt the LLM to generate several prompts that can guide LLMs in performing a specific task, as demonstrated in the log parsing task in Fig. 4. First, the self-prompt strategy prompts the LLM to generate a pool of prompt candidates C using a prompt that consists of a precise task description and output requirements. The pool C contains multiple prompts that ChatGPT believes can be used for log analysis tasks. Next, we evaluate the prompts in C on a small task-specific log dataset, selecting the best prompt based on a metric score $s(p)$, where $p \in C$. Finally, with S_p = “a parsed log template” representing the answer range string, the complete self-prompt for log parsing, combined with format control functions, can be expressed as:

Self-prompt: $\arg \max_{p \in C} \{s(p)\} f_x([X]) f_z([Z], S_p)$

Specifically, we evaluated the log parsing performance of the five prompt candidates in Fig. 4, using the first 100 logs of the Android

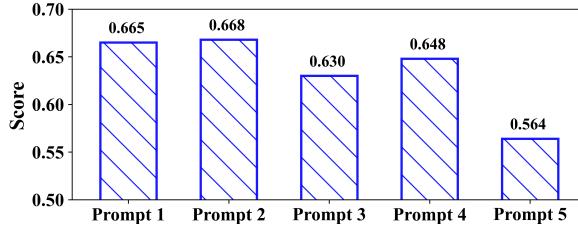


Figure 5: Performance comparison of the five raised prompt candidates in Fig. 4 for Log Parsing.

dataset as a test set. The Android dataset has the highest number of log templates (as shown in Table 1), making it suitable for the selection of prompt candidates. The metric score $s(\cdot)$ was set to be the average of RandIndex and F1-score. As shown in Fig. 5, although ChatGPT considered all five candidate prompts to be suitable for log parsing, a significant fluctuation in performance was observed among the candidates. For example, prompt 2 achieved the best performance, outperforming prompt 5 by 18.4%. Thus, prompt 2 was chosen to form the complete self-prompt in the main experiment. Note that since Prompt 1 and Prompt 2 achieved similar scores, they both can be regarded as optimal prompts for log parsing in practical applications. A further finding is that prompts that use more accurate language and provide step-by-step instructions tend to produce better outcomes. For example, compared to prompt 5, prompt 2 provides more formal and accurate words (such as “standardized” and “convert”) and clearly outlines the intermediate steps for the task (identifying and replacing variables, then converting to a template). Similar reasoning can be applied to the relative performance of prompts 3 and 4, which outperformed prompt 5 by using more accurate words (such as “common structure”, “transform” and “variable”).

3.4.2 CoT Prompt. The concept of chain of thought (CoT), a series of intermediate reasoning steps, is introduced by Wei *et al.* [39]. The CoT prompt emulates the human thought process when addressing complex problems and can enhance the performance of LLMs in challenging tasks, such as solving mathematical problems. In manual log analysis, practitioners also engage in a series of reasoning steps to reach a conclusion. For instance, without further definitions, the boundary between a normal log and an abnormal log is unclear. Our preliminary experiments revealed that when the simple prompt in Fig. 2 is used, ChatGPT often reports anomalies based on hallucinatory content (e.g., 0x00000000, a string of zeros, leads ChatGPT to believe this log recorded an invalid memory access) or reports anomalies based on the belief that a log is abnormally corrupted. Therefore, the intermediate thought process should be regulated in the prompt to suppress the divergence of generated answers.

The CoT prompt guides LLMs in performing log analysis with constrained intermediate steps, both *implicitly* and *explicitly*. (1) *Implicitly*: We require the LLM to generate a reason for each normal/abnormal answer, justifying its decisions. During the pre-training process, the model learned to follow correct and sound logic chains widely existing in cleaned real-world corpora (e.g., wiki pages and books) [2, 29, 38]. Our intuition is that, by requesting reasons, the

model “recalls” the logic chains in training data and implicitly creates a reasonable chain of thoughts, resulting in more logical answers. (2) *Explicitly*: We explicitly define intermediate steps in the prompt. For example, in the task of anomaly detection, we constrain the definition of anomaly to be only “alerts explicitly expressed in textual content” to prevent LLMs from “overthinking” (e.g., when information is lacking, ChatGPT assumes that the key information is abnormally erased). It is important to note that we include a step to ignore $(*)$, the wildcard symbol representing variable parts in a parsed log template, since inputs of this task are log templates.

Assuming S_a = “a binary choice between abnormal and normal” as the answer range string, the complete CoT prompt for anomaly detection with format controls is shown as:

CoT prompt: Classify the given log entries into normal and abnormal categories. Do it with these steps: (a) Mark it normal when values (such as memory address, floating number and register value) in a log are invalid. (b) Mark it normal when lack of information. (c) Never consider $(*)$ and missing values as abnormal patterns. (d) Mark it abnormal when and only when the alert is explicitly expressed in textual content (such as keywords like error or interrupt). Concisely explain your reason for each log. $f_x([X]) f_z([Z], S_a)$

3.4.3 In-context Prompt. LLMs are capable of in-context learning, performing a new task through inference alone by conditioning on a few input-label pairs (demonstrations) and making predictions for new inputs [2, 27]. The input-label pairs are directly integrated into the prompt, without requiring an iterative training process (*i.e.*, multiple epochs of parameter updating) and additional resource costs, thereby enabling efficient adaptions in the online scenario. The in-context prompt in LogPrompt adheres to this idea, leveraging a few labelled logs at the beginning of each prompt to create the task context. LLMs are then prompted to provide predictions for logs in the slot $[X]$ based on the given demonstrations in the context. For instance, in the task of anomaly detection, we provide a few randomly sampled normal/abnormal logs and their corresponding categories (1 for abnormal logs and 0 for normal logs) and expect LLMs to implicitly create a task context for anomaly detection and correctly classify input logs. Similarly for log parsing, randomly sampled log-template pairs are used to build the prompt.

Given a small number m , m log-label pairs $L_1 \dots L_m$ are randomly sampled. For anomaly detection, with S_c = “a binary choice between 0 and 1”, the in-context prompt is expressed as:

In-context prompt: Classify the given log entries into 0 and 1 categories based on semantic similarity to the following labelled example logs: (1) Log: L_1 Category: 1 (2) Log: L_2 Category: 0 ... (m-1) Log: L_{m-1} Category: 1 (m) Log: L_m Category: 0. $f_x([X]) f_z([Z], S_c)$

For log parsing, the prompt can be built in a similar manner with the corresponding template attached behind each example log and S_c replaced with the one in Section 3.4.1.

4 EXPERIMENTS AND EVALUATIONS

In this section, we first discuss the experimental setup. We then propose four research questions and discuss the results.

4.1 Experimental Setup

Table 1: The Statistics of Log Datasets in the Experiments.

Dataset	#Messages	#Templates (2k)	#Anomalies	L.P. ^a	A.D. ^a
HDFS	82,293,702	14	-	✓	-
Hadoop	394,308	114	-	✓	-
Zookeeper	74,380	50	-	✓	-
BGL	4,713,493	120	348,460	✓	✓
HPC	433,489	46	-	✓	-
Linux	25,567	118	-	✓	-
Proxifier	21,329	8	-	✓	-
Android	30,348,042	166	-	✓	-
Spirit	7,958,733	-	768,142	-	✓

^a L.P. denotes the task of log parsing. A.D. denotes anomaly detection.

✓ means the dataset is used as a task evaluation set with human labels.

4.1.1 Datasets. As shown in Table 1, the effectiveness of LogPrompt is evaluated on large-scale real-world log datasets from nine different domains, including supercomputers, distributed systems, operating systems, mobile systems, and server applications [10]. Two of the datasets were annotated by domain experts to identify anomalous events for the purpose of anomaly detection [28]. The number of labelled anomalies can be found in Table 1. To evaluate the log parsing performance, eight of the datasets have log templates manually extracted from a subset of 2000 log messages in each domain [45]. All log data were timestamped, enabling the datasets to be split into train/test sets chronologically.

4.1.2 Implementation Details. In our primary experiments, the underlying LLM is ChatGPT and is accessed via APIs provided by OpenAI (model version: *gpt-3.5-turbo-0301*). In Section 4.2.4, an open-source LLM with fewer parameters is deployed using 1 NVIDIA A100 GPU. The initial temperature coefficient of ChatGPT is set to 0.5, maintaining a balance by increasing the model’s reasoning capabilities through diverse token exploration while limiting detrimental randomness [41]. If the response format is invalid, the query is resubmitted with an increased temperature coefficient of 0.4 until the response format is correct. The format failure rate is less than 1%, which is consistent with [14]. The train/test datasets are split chronologically to simulate online scenarios.

4.2 Research Questions and Results

4.2.1 RQ1: How effective is LogPrompt compared with existing methods in the online scenario, when being faced with scarce training data?

Motivation. As shown in Figure 3, the performance of existing log analysis methods is often reduced in the online scenario, where available in-domain training logs are often limited. The aim of this Research Question (RQ) is to investigate whether LogPrompt can address this challenge. In this comparison experiment, LogPrompt stands out for its unique approach of not requiring in-domain training, while existing approaches are trained on a small portion (less

than 10%) of in-domain logs to simulate the online scenario. The amount of training data we provided for existing approaches aligns with previous studies on online log analysis [24, 35, 37]. It should be noted that, as will be discussed in Section 6, the performance of some approaches may be underestimated in this setting.

Approach. To evaluate the performance of LogPrompt for log parsing, we compare it against 10 existing methods, namely LogPPT [19], Logstamp [37], LogParse [24], LogSig [34], Spell [6], IPLoM [23], Drain [9], FT-tree [42], MoLFI [26], and LKE [8]. The datasets used in the evaluation are described in Table 1. For each dataset, most baseline methods are trained on the first 10% logs and evaluated on the remaining 90% logs, while LogPrompt is directly tested on the remaining 90% logs. Note that LogPPT [19] is a recent log parsing approach specialized for few-shot scenarios. Thus, we set the ratio of its training data to 0.05%.

In our evaluation of log parsing, we adopt the F1-score as the metric. While RandIndex [32] is commonly used, it only assesses the accuracy of log clustering and does not consider the correctness of variables in the extracted templates. Recent studies [35, 36] have highlighted that RandIndex is not sensitive to parsing errors and may assign a high score even to random clustering. In contrast, the F1-score presents a more challenging metric as it requires precise identification of variable parts in logs [11, 20], which is particularly crucial for parsing unseen logs in online scenarios. To calculate the F1-score, we tokenize the predicted log template into a list of tokens. We then define the terms *TP*, *TN*, *FP*, and *FN* as follows: (1) *TP*: the token is a variable and is correctly predicted as such, (2) *TN*: the token is not a variable and is predicted not to be a variable, (3) *FP*: the token is not a variable but is mistakenly predicted as a variable, and (4) *FN*: the token is a variable but is predicted not to be a variable. The F1-score is computed as the harmonic average of Recall ($Recall = \frac{TP}{TP+FN}$) and Precision ($Precision = \frac{TP}{TP+FP}$).

For the task of anomaly detection, we compare LogPrompt with three representative existing methods: DeepLog [7], LogAnomaly [25], and LogRobust [43]. Following the paradigm of anomaly detection raised in [17], we first parse each input log into a log template and form log sessions from templates, using fixed-window grouping with a length of 100 templates. To ensure a fair comparison, for all methods, the templates are extracted using Drain [9], a fast log parsing algorithm. For baselines, the first 4000 logs in each dataset are used for training. Both LogPrompt and the trained baselines are then tested on the remaining logs. For LogPrompt, the LLM first predicts a normal/abnormal tag for each log template in a session and the session is considered abnormal if any abnormal tag exists. In this experiment, since existing methods output only session-level anomalies, we use the F1-score for session anomaly as the evaluation metric (See template-level performance in Table 7), where *TP* represents the successful identification of an abnormal session (similarly for *TN*, *FP*, and *FN*). We also report the Recall and Precision values. For the forecast-based methods (DeepLog and LogAnomaly), we report the top-4 F1-score.

Result of Log Parsing. The results of the log parsing task are presented in Table 2. Utilizing the self-prompt strategy in Section 3.4.1, LogPrompt achieved the best F1-score on six of the eight datasets. Moreover, despite not being trained on in-domain logs,

Table 2: F1-score of Log Parsing in the Online Scenario.

Methods ^a	HDFS	Hd ^b	Zk	BGL	HPC	Linux	Px	Andro	Avg.
IPLoM [23]	0.389	0.068	0.225	0.391	0.002	0.225	0.500	0.419	0.277
LKE [8]	0.424	0.198	0.225	0.379	0.381	0.388	0.309	0.000	0.288
LogSig [34]	0.344	0.050	0.225	0.333	0.002	0.146	0.339	0.116	0.194
FT-tree [42]	0.385	0.046	0.186	0.497	0.002	0.211	0.420	0.581	0.291
Spell [6]	0.000	0.058	0.045	0.536	0.000	0.091	0.000	0.245	0.122
Drain [9]	0.389	0.068	0.225	0.397	0.002	0.225	0.500	0.413	0.277
MoLFI [26]	0.000	0.095	0.000	0.333	0.000	0.026	0.000	0.208	0.083
LogParse [24]	0.632	0.502	0.348	0.665	0.330	0.588	0.334	0.233	0.454
LogStamp [37]	0.523	0.594	0.275	0.818	0.434	0.658	0.438	0.899	0.580
LogPPT [19]	0.838	0.526	0.795	0.982	0.287	0.423	0.638	0.313	0.600
LogPrompt	0.863	0.763	0.889	0.865	0.759	0.766	0.653	0.819	0.797

^a LogPrompt uses **no training data**, unlike others trained with up to 10% logs.^b Hd, Zk, Px and Andro denotes Hadoop, Zookeeper, Proxifier and Android.

LogPrompt outperformed LogPPT by 32.8% and LogStamp by 37.4% in average, which require resources for in-domain training. The advantage of LogPrompt indicates its strong ability in accurately recognizing variables from unseen logs. It is worth noting that, according to recent studies [11, 20], accurately identifying fine-level variables in logs is of greater significance in the analysis of breakdowns and failures in many industrial scenarios, as opposed to solely relying on coarse-level clustering of logs. The effectiveness of LogPrompt in recognizing fine-level variables in logs makes it a suitable choice for parsing unseen logs in online scenarios.

Table 3: Log Anomaly Detection in the Online Scenario.

Methods	#Train ^a	BGL ^c			Spirit		
		P	R	F	P	R	F
DeepLog [7]		0.156	0.939	0.268	0.249	0.289	0.267
LogAnomaly [25]	4000	0.016	0.056	0.025	0.231	0.141	0.175
LogRobust [43]		0.095	0.425	0.156	0.109	0.135	0.120
LogPrompt^b	0	0.249	0.834	0.384	0.290	0.999	0.450

^a #Train denotes the number of logs utilized for training.^b P denotes Precision. R denotes Recall. F1 denotes F1-score.^b LogPrompt here is constructed via the strategy of CoT-prompt.^c We automatically extracted 1766 templates from BGL; 1297 from Spirit.

The reported F1-score is session level. See template level in Table 7.

Result of Anomaly Detection. The results of the anomaly detection task are presented in Table 3. Despite the existing methods being trained on thousands of logs, LogPrompt still achieved strong performances in both datasets without utilizing any in-domain training data, with an average improvement of 55.9% in terms of F1-score. This indicates the effectiveness of LogPrompt in addressing the performance reduction in anomaly detection when training data is scarce. However, even with LogPrompt, the absolute performance of anomaly detection in the online scenario remains low, highlighting the challenging nature of performing anomaly detection with extremely limited historical in-domain logs.

4.2.2 RQ2: Does LogPrompt’s interpretability yield helpful and comprehensible content for practitioners during real-world log analysis?

Motivation. As discussed in Section 3.1, the insufficient interpretability of existing approaches is a major pain point for practitioners during log analysis. Although justifications and explanations, as shown in Fig. 1(b), can be generated by LogPrompt, it is still hard to determine the quality of such content in industrial scenarios. To assess the impact of LogPrompt on enhancing the interpretability of log analysis, we established a set of criteria for evaluating interpretability, presented in Table 4. We cooperated with a top-tier ICT & software company and recruited six of their experienced practitioners (with full-time commitment to this project) to evaluate LogPrompt’s interpretability in terms of usefulness and readability. The recruited experts possess over 10 years of experience of O&M in various domains such as distributed systems, mobile operating systems, and software services.

Approach. A total of 200 logs were randomly sampled for the human evaluation, accompanied by LogPrompt’s actual outputs. Specifically, 100 logs were related to log parsing, evenly distributed across the eight domains that have human-labelled variables and templates in Table 1. While the remaining 100 were related to anomaly detection, evenly distributed across BGL and Spirit, the two domains with human labels. As the correctness of LogPrompt was already evaluated in Section 4.2.1, incorrectly predicted logs (*i.e.*, FPs and FNs) were not included in this evaluation. To ensure a comprehensive evaluation, an equal number of normal and abnormal samples were included for anomaly detection, and each selected log for log parsing was required to contain at least one variable. Fig. 1(b) presents two example formats of logs with LogPrompt’s outputs, which were rated by the reviewers. For anomaly detection, we utilized the CoT prompt (implicit part only) to prompt ChatGPT to generate task outputs along with justifications. For log parsing, we constrained the variables to the eight general categories defined by Li *et al.* [20] and prompted ChatGPT to classify and generate explanations for each variable. Following Ahmed *et al.* [1], the evaluation dimensions focused on the following two aspects:

- **Usefulness.** The reviewers were asked to rate the extent to which the provided interpretation of the analysis decision is detailed, specific, relevant, logically sound and helpful in actual log analysis.
- **Readability.** The reviewers were asked to rate the ease with which a reader could understand the provided text. A text was considered readable if it was grammatically correct, meaningful, and professional.

For each evaluation dimension, a Likert scale ranging from one to five is utilized to score the interpretability level, with Table 4 providing detailed definitions for each level. Every reviewer independently rated all the 200 samples. For each log, the reviewers were asked to provide a usefulness score and a readability score based on their understanding of the scoring criteria. To mitigate bias, the reviewers were unaware of the source of the generated explanations during evaluations, and there was no overlap between co-authors and reviewers. We reported two metrics: (1) The average score assigned by each reviewer across all samples. (2) The High Interpretability Percentage (HIP), which represents the proportion of samples that received a score higher than four from a reviewer.

Table 4: Human Scoring Criteria for Evaluating the Interpretability of Large Language Models in Log Analysis. Score Ranking: 1 - Not Interpretable, 2 - Low Interpretable, 3 - Moderately Interpretable, 4 - Highly interpretable, 5 - Very Highly interpretable.

Scores	Usefulness				Readability	
	Log Parsing		Anomaly Detection			
1	No variable is extracted, or explanations of those variables are irrelevant.		No justification on the anomaly beyond a simple prediction label.		The text contains numerous unintelligible elements or grammatical mistakes.	
2	Explanations are meaningless or logically wrong, hindering engineers from interpreting the logs.		The Justification for the prediction is irrelevant, or logically inconsistent with the facts.		Most of the generated text is readable, but it may have grammar errors or unclear phrases.	
3	Variables with appropriate classes are partially extracted, and explanations are somewhat relevant.		The justification well supports the predictions, but may lack clarity and details.		The text has few grammar errors, although some terms may need refinement.	
4	Variables with appropriate classes are mostly extracted, and explanations are specific and relevant. Engineers can understand the logs with less effort.		Specific, accurate, and relevant justification is presented, which positively assist engineers in eliminating false alarms and further analysis.		The text is clear, grammatically correct, with only a minimal number of technical terms possibly needing refinement.	
5	Variables are fully and correctly extracted, and explanations are detailed, specific and relevant, enabling easy and precise understanding of logs.		Detailed, relevant and clear justification that significantly assists engineers in ruling out false alarms and locating the root cause.		The text is clear, detailed, grammatically perfect, and professional for software engineering.	

Table 5: Interpretability of LogPrompt Rated by Experts.

Raters	Log Parsing				Anomaly Detection			
	Usefulness		Readability		Usefulness		Readability	
	Mean ^a	HIP ^b	Mean	HIP	Mean	HIP	Mean	HIP
R1	4.19	76.00%	4.70	94.00%	4.27	73.00%	4.50	84.00%
R2	4.36	81.00%	4.78	95.00%	4.42	79.00%	4.63	90.00%
R3	4.13	91.00%	4.18	95.00%	4.12	86.00%	4.35	94.00%
R4	4.20	73.00%	4.74	98.00%	4.40	85.00%	4.71	95.00%
R5	4.06	91.00%	4.41	100.00%	4.12	94.00%	4.24	98.00%
R6	4.46	86.00%	4.77	97.00%	4.48	90.00%	4.78	99.00%
Avg.	4.23	83.00%	4.60	96.50%	4.30	84.50%	4.54	93.33%

^a Mean is the average score of the ratings by a reviewer for all samples.

^b HIP is the percentage of samples received a rating higher than four.

Result. The results of the human evaluation are presented in Table 5. For each reviewer, the average scores on the samples for both tasks in terms of usefulness and readability consistently exceeded four. For the HIP, there were some fluctuations ranging from 70% to 90%, which can be attributed to the subjective nature of the criteria. However, in every evaluation dimension, the average HIP was consistently above 80%, indicating that most generated content was considered helpful and readable by experienced log analysts.

Feedbacks from Practitioners. Alongside the ratings, we obtained valuable comments and feedback from the reviewers. One practitioner states that: “I appreciate the ability of LogPrompt in supporting the interpretation of logs from various domains. As our system continuously incorporating third-party services, sometimes I have to read through the manuals to decipher logs from unfamiliar domains. The explanations generated by LogPrompt can provide a swift grasp of the logs before I found the official definitions.”

Another comments said: “LogPrompt can definitely help in the compose of analysis reports, especially shortly after system crashes, where I often need to quickly interpret the events to non-technical colleagues in meetings.”

Another reviewer commented: “In the realm of software O&M, false alarms are an inescapable reality, with false positives imposing substantial time costs on engineers and false negatives causing severe ramifications. Accompanying explanations with automatic analysis outcomes enables engineers to more promptly ascertain the credibility of the purported anomaly, thereby reducing the time spent on subsequent actions.”

Table 6: Analysis on low-rated samples in human evaluation

Cause	Ratio	Example	
		Raw log	Interpretation
Domain Knowledge	45.95%	CE sym 33, at 0x1ff2fc60, mask 0x04.	Sym 33 is a Object Name that refers to an object named “sym 33”.
Semantic insufficiency	40.54%	fpr8=0x3883a497 bfe016b5 8da3311e 3f06d5a7.	Normal - Register values.
Other	13.51%	-	-

Bad Case Analysis. We investigate the samples that received ratings averagely lower than four by practitioners. As shown in Table 6, a major factor is the LLM’s lack of domain knowledge, which leads to overly general interpretations of some domain-specific terms. For example, it may refer to specific parameters as simply an object. Another cause is the lack of semantic contents in certain input logs, which can be attributed to their brevity or richness of non-NLP patterns (*i.e.*, digits, codes and addresses). This insufficiency makes it challenging for the model to generate sound explanations.

The practitioners’ feedback on LogPrompt indicated its potential for real-world applications and provided valuable insights for optimizing the model, including the need to improve the model’s domain-specific knowledge and its understanding of non-NLP patterns in logs through domain adaptation technologies.

4.2.3 RQ3: How do varying prompt strategies within LogPrompt impact the performance of LLMs in log analysis tasks?

Motivation. Given the prompt strategies discussed in Section 3.4, this RQ aims to investigate the influence of these strategies on the performance of LLMs in log analysis tasks. Specifically, this RQ seeks to explore the impact on performance by examining various settings and modules in LogPrompt, which will help guide the selection of optimal settings for LogPrompt in practical applications and enhance understanding of the mechanism of LogPrompt.

Table 7: Ablation Study of the CoT Prompt Strategy in Anomaly Detection.

Methods ^b	BGL		Spirit	
	S-F1 ^a	T-F1	S-F1	T-F1
ChatGPT (simple prompt)	0.189	0.115	0.445	0.086
+Implicit CoT	0.307	0.183	0.443	0.096
+Explicit CoT	0.384	0.321	0.450	0.116

^a S-F1 denotes the session-level F1-score.

^b T-F1 denotes the template-level F1-score.

^b The implicit CoT requires ChatGPT to report justifications.
The explicit CoT further adds intermediate steps in prompts.

Impact of Modules in the CoT Prompt. As detailed in Section 3.4.2, the CoT prompt consists of two modules: implicit CoT and explicit CoT. Starting from the simple prompt in Fig. 2, we successively added each module to evaluate its impact on the performance of ChatGPT in the task of anomaly detection. In addition to the session-level F1-score reported in Table 3, we also report the template-level F1-score, which reflects the performance in predicting the anomaly tag for each log template. The results of our evaluation are presented in Table 7, which demonstrate that incorporating chains of thought in the prompt continuously improves the performance of ChatGPT in anomaly detection (e.g., for the BGL dataset, the session-level F1-score improved by 0.195 and the template-level F1-score improved by 0.206). This supports the idea that a more precise and step-by-step prompt leads to better performance from LLMs.

An interesting observation from our evaluation is that even when only the implicit part of the CoT prompt was utilized, the performance of ChatGPT was still significantly improved (e.g., for the BGL dataset, the session-level F1-score improved by 62.4% and the template-level F1-score improved by 59.1%). The only difference between the simple prompt and the prompt loaded with the implicit CoT was that the LLM was asked to generate justifications for its analysis decisions. This improvement in performance can be attributed to the fact that the LLM implicitly generates reasonable intermediate steps for the task, leading to a more logical result when being prompted to justify its decisions. Given that language models are trained to fit sound logical chains from various real-world corpora, generating extra explanations in log analysis results reduces the likelihood of the language model inferring incorrect logical chains. This further emphasizes the importance of interpretability in log analysis.

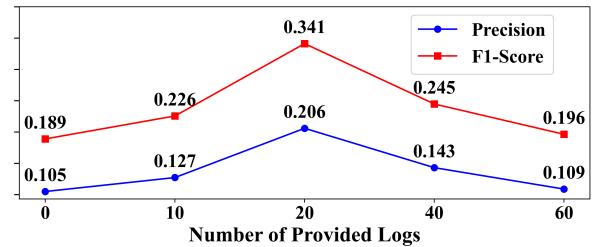


Figure 6: Variation in F1-score and Precision of the in-context prompt with an increase in the number of provided logs.

Impact of Number of Labelled Logs in the In-context Prompt. Fig. 6 presents the performance of the in-context prompt in the anomaly detection task on the BGL dataset when we vary the number of labelled logs provided in the prompt. The Recall is mostly steady across the number of logs (consistently achieving 0.99+ with the only exception of 0.847 observed when the number of logs is 40), and thereby is not presented in the figure. For each number of logs, the number of positive samples is equal to the number of negative samples. The results demonstrate that even with a few labelled logs provided in the prompt, the performance of ChatGPT is significantly improved compared to the simple prompt (*i.e.*, where no labelled logs are provided), indicating the effectiveness of the in-context prompt. Once given a few log examples that are divided into positive and negative clusters, even without explicitly specifying the context of anomaly, the LLM can classify new input logs into their corresponding clusters, by implicitly creating a task context for anomaly detection.

As seen from Fig. 6, both the F1-score and precision reach their peak when 20 labelled logs are provided in the prompt, and then decrease with the further increase of labelled logs. This phenomenon can be attributed to the idea that, instead of learning the input-label correspondences from large-scale training data like typical supervised learning, LLMs model other data components (*i.e.*, example inputs, example labels, and the example format) from the examples in the in-context prompt, which can be efficiently learned from only a few examples [27]. Additionally, an overly long context prefixed to the prompt may cause LLMs to pay less attention to the new input logs, thereby deteriorating the task performance.

4.2.4 RQ4: Is LogPrompt Compatible With Open-source and Smaller-scale LLMs?

Motivation. Although LogPrompt exhibits promising qualities, it is important to address two key concerns. Firstly, the deployment of large, proprietary, and API-dependent LLMs, such as ChatGPT, in local environments can be a challenging task. Secondly, the reproducibility of LogPrompt may be compromised if the API services of LLMs become unavailable. Therefore, it is crucial for LogPrompt to have compatibility with alternative open-source, privacy-protected, and smaller-scale LLMs. In this RQ, we explore the compatibility of LogPrompt with a smaller-sized and open-source LLM.

Approach. In addition to ChatGPT, the proprietary model with 175B parameters developed by OpenAI, we further evaluate the performance of LogPrompt under Vicuna [4], an open-source LLM

with only 13B parameters. *Vicuna* [4] can be easily deployed on MacBook and mobile platforms, requiring minimally only CPU and less than 7GB of memory¹. We investigate the performance of log parsing on the eight datasets in Table 2 without in-domain training, which is the same setting as Section 4.2.1.

Table 8: Performances of LogPrompt in Online Log Parsing with Varying Underlying LLMs.

LLMs	Size	HDFS	Hd ^a	Zk	BGL	HPC	Linux	Px	Andro
GPT Model									
<i>ChatGPT</i>	175B	0.863	0.763	0.889	0.865	0.759	0.766	0.653	0.819
Vicuna^a (with)									
<i>Simple Prompt</i>	13B	0.244	0.018	0.273	0.048	0.022	0.109	0.014	0.041
<i>LogPrompt</i>		0.779	0.468	0.364	0.266	0.243	0.715	0.614	0.248

^a **Hd, Zk, Px, Andro** denotes Hadoop, Zookeeper, Proxifier, Android.

^b Vicuna with LogPrompt uses the in-context prompt strategy with $m=3$.

Result. The results in Table 8 demonstrate that, although Vicuna has only 13B parameters, when equipped with LogPrompt, it achieves a comparable performance of log parsing with the 175B GPT model in the datasets of HDFS, Linux and Proxifier. Additionally, when the prompt strategy transitions from a simple prompt to LogPrompt, Vicuna exhibits significant improvements on performance, with an average increase of 380.7% in F1-score. The effectiveness of LogPrompt with the significantly smaller-scale LLM highlights its reproducibility and compatibility. Furthermore, as Vicuna is open-source and requires fewer resources for training and deployment, the success of LogPrompt on Vicuna holds promising implications for building in-domain industrial applications.

5 DISCUSSION

Based on our experimental findings, we identify several implications for LLM-based log analysis.

(1) Challenges in online anomaly detection. Without in-domain training data, the F1-score achieved by ChatGPT for anomaly detection is only 0.450, making it unsuitable for direct application in industry. The difficulty of online anomaly detection lies in the varied and flexible definitions of anomalies across different systems, which makes accurate detection through textual information in logs alone challenging. One potential solution is to incorporate general knowledge of existing software systems into LLMs.

(2) Significance of log interpretation. The positive feedback from practitioners regarding the generated explanations indicates the potential for improved user experiences in using log interpretation tools for practical log analysis. Interestingly, the results in Table 7 demonstrate that when LLMs are asked to generate both predictions and interpretations simultaneously, there is an additional improvement in the accuracy of analysis outcomes. We attribute this to the assumption that by demanding interpretations, an implicit CoT is created in the prompt, causing LLMs to "recall" the clean general CoT samples from the pre-training stage. Therefore, introducing domain-related CoT samples (e.g., log interpretations) in the pre-training stage of LLMs may lead to even greater improvements.

¹An approach of efficient deployment: <https://github.com/ggerganov/llama.cpp>

(3) Complexity alignment of prompt strategy. Certain strategies discussed in this paper, such as the CoT prompt for log parsing, have not been extensively explored for other tasks due to concerns about complexity. The mainly used prompt in log parsing, the self-prompt, already achieves good performance (F1-score of 0.797), which suggests that log parsing is a relatively easier task compared to anomaly detection (best F1-score of only 0.450). Therefore, implementing a more complex prompt like CoT is unnecessary for log parsing, as CoT is more suitable for tasks involving mathematical or logical reasoning, such as anomaly detection [39].

6 THREATS TO VALIDITY

Our study faces several potential threats:

(1) Labelling errors in public datasets. Despite the use of publicly available datasets that contain logs from various domains, the manually labelled contents in the logs (*i.e.*, log templates and anomaly tags) are still susceptible to errors, as reported in previous studies [11, 16]. The sheer volume of raw logs makes it infeasible to identify and correct all errors, which may result in lower absolute performance in analysis tasks. For example, We have identified logs in the BGL dataset that are labeled as abnormal but appear to be semantically normal (*e.g.*, the log message "exited normally with exit code" is marked as abnormal), which might explain the lower Recall score of LogPrompt in the BGL dataset. However, relatively speaking, LogPrompt still demonstrates remarkable performance, and the results of the ablation study indicate its efficacy.

(2) Black box nature of LLMs. The pre-training dataset used for LLMs consists primarily of textual data derived from web pages, books, and Wikipedia [2, 29, 38]. Due to the black box nature of LLMs, it is possible that these LLMs, including BERT [13], have been exposed to certain log fragments and subsequently acquired relevant concepts (*e.g.*, from Wiki pages related to log analysis) during pre-training. However, both recent studies [12, 14, 30] and our ablation studies (Table 7 and 8) emphasize that, without an appropriate prompt strategy, it is hard to elicit the domain knowledge stored within LLMs and further facilitate their log analysis capabilities, which may be the reason why LogPrompt outperforms other LLM-based approaches, such as LogPPT [19] and LogStamp [37].

(3) Underestimated Baselines. To emulate the online scenario, we trained all baseline methods on a limited portion of each in-domain dataset (*e.g.*, 10%), which may have led to lower performance compared to the results reported in the original literature. Notably, most baselines are not specifically designed for online scenarios and require substantial training data (*e.g.*, 90%) to achieve optimal performance. Therefore, the underestimated baselines pose a fairness concern in the comparisons, particularly considering the extensive amount of data used to pre-train LLMs.

(4) Insignificance of human evaluations. The limited number of sampled logs for the human evaluation in our study makes it difficult to achieve statistical significance in terms of usefulness and readability scores. Furthermore, it is uncertain whether the working experience of the human evaluators sufficiently qualifies them to judge the usefulness of the explanations generated for log analysis. However, scaling the sample size can be difficult due to the nature of human evaluations, and we have made efforts to ensure that all participants possess substantial experience (*i.e.*, at least 10 years)

in the field of software O&M. Nevertheless, the positive comments and feedback from these practitioners revealed promising merits of LogPrompt.

7 RELATED WORK

7.1 Log Analysis Tasks

7.1.1 Log Parsing. Log parsing is a well-established technique for reducing the size of large log volumes by extracting meaningful templates from raw logs. This enables more effective analysis, such as anomaly detection, by simplifying the log data. Traditional approaches focus on the aggregation of lexically similar logs, such as cluster-based methods [8, 45], heuristic methods [6, 23], and tree structure mining methods [5, 9, 42].

More recently, there has been a growing recognition of the importance of variables in logs. Variables are no longer viewed as the parts in logs to be “abandoned”, but the key to achieve deeper understanding of logs. LogParse [24] and LogStamp [37] use a word-level classifier to learn variable patterns and recognize variables directly from raw logs. Huo *et al.* [11] model both instance-level and message-level semantics for variables in logs, leading to improved downstream analysis. Li *et al.* [20] manually derive eight general categories for variables in software logs and achieves high accuracy in variable classification.

In addition, LogPPT [19] incorporates prompt tuning techniques and employs a BERT-like language model for the purpose of log parsing. However, LogPPT’s dependence on fine-tuning and predicting virtual prompt tokens using domain-specific logs constrains its adaptability and interpretive capabilities.

7.1.2 Log Anomaly Detection. Anomaly detection in logs can be performed using either forecast-based methods or classification-based methods. Classification-based methods require both normal and abnormal training logs, while forecast-based methods only require normal historical logs for training.

Examples of classification-based methods include LogRobust [43], which uses an LSTM model and a general-purpose word embedding as features, and Lu *et al.* [22], who use a convolutional neural network (CNN) to capture sequential log features. Le *et al.* [16] use a BERT encoder to directly recognize anomalies in raw logs without the need for log parsing. Forecast-based methods include DeepLog [7], which extracts normal log patterns and considers a new log anomalous if it violates these patterns, and LogAnomaly [25], which uses domain-specific synonyms and antonyms to match new logs with existing templates.

7.2 LLMs in Software O&M

We also noticed the concurrent work by Le *et al.* [18], who evaluate the effectiveness of ChatGPT for log parsing, with less emphasis on developing intricate prompts. Our research distinguishes itself from other LLM-based approaches and prior studies in log analysis in several ways. Firstly, we explore LLMs-based approaches for both log parsing and anomaly detection tasks, unlike the single supported task of log parsing in [18, 19]. Secondly, we provide scientific descriptions of prompt construction and validate the effectiveness of three advanced prompt strategies in log analysis, in contrast to the simple prompt strategies employed in [18, 19]. Thirdly, this is

the first work to address the challenge of limited interpretability in log analysis, which is an area that has been overlooked in previous research. To evaluate interpretability, we conduct a rigorous human evaluation guided by detailed interpretability evaluation criteria. Lastly, our proposed approach is highly adaptive and does not require in-domain training, unlike existing work that still relies on resources for an in-domain training stage [19].

Although this is among the very first studies to apply LLMs to log analysis, there have been several studies exploring the use of LLMs in software O&M. Ahmed *et al.* [1] fine-tune GPT-3.x models to recommend root causes and mitigation steps for cloud incidents and propose a scoring schema for human evaluation of the generated recommendations. Somashekhar *et al.* [33] propose adapting language models to the task-specific domain through training on system configuration files to enhance the configuration tuning pipeline for distributed applications. In contrast to these studies, which fine-tune LLMs with large amounts of training data in a specific domain, our work explores general prompt strategies for LLMs to perform log analysis that can be applied to any domain.

8 CONCLUSION

In this study, our objectives transcend the simple application of the latest advancements in LLMs within log analysis. Instead, we aim to offer a scientific and comprehensive engineering methodology to accurately discern how to optimize prompts to enhance the performance of LLMs within log analysis, and, crucially, to comprehend the potential areas where LLMs can aid engineers in software O&M and program comprehension. Although the advanced measures of LogPrompt in online scenarios augment adaptability in log analysis by superseding the labor-intensive in-domain training process, we believe that LogPrompt’s primary and unique advantage resides in facilitating program comprehension by furnishing justifications alongside predictions. In our expert study, practitioners provided positive anticipations on the application of LogPrompt in real-world scenarios, including event comprehension in unfamiliar domains, composition of analysis reports, and ruling out false alarms. In addition, we constructed a systematic and intricate criteria to assess the usefulness and readability of these generated explanations in log analysis, which may inspire future research aimed at improving interpretability scores and facilitate industrial applications. Lastly, we substantiate that LogPrompt is compatible with open-source and smaller-scale LLMs, thereby validating its feasibility for actual deployment.

Future research directions include advancing the ability of LLMs through domain adaptation techniques, validating LogPrompt on more LLMs and expanding prompt strategies to more tasks.

REFERENCES

- [1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *ICSE 2023*.
- [2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [3] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R Lyu. 2021. Experience report: Deep learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908* (2021).
- [4] Wei-Lin Chiang, Zhuohai Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Liannmin Zheng, Siyuwan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eri P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. <https://lmsys.org/blog/2023-03-30-vicuna/>
- [5] Guojun Chu, Jingyu Wang, Qi Qi, Haifeng Sun, Shimin Tao, and Jianxin Liao. 2021. Prefix-Graph: A Versatile Log Parsing Approach Merging Prefix Tree with Probabilistic Graph. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2411–2422.
- [6] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.
- [7] Min Du, Feifei Li, Guineeng Zheng, and Vivek Srikanth. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.
- [8] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. 149–158.
- [9] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.
- [10] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448* (2020).
- [11] Yintong Huo, Yuxin Su, Cheryl Lee, and Michael R Lyu. 2023. SemParser: A Semantic Parser for Log Analytics. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 881–893.
- [12] Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu. 2023. Is ChatGPT a good translator? A preliminary study. *arXiv preprint arXiv:2301.08745* (2023).
- [13] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.
- [14] Tom Kocmi and Christian Federmann. 2023. Large language models are state-of-the-art evaluators of translation quality. *arXiv preprint arXiv:2302.14520* (2023).
- [15] Tiffany H Hung, Morgan Cheatham, Arielle Medenilla, Czarina Sillos, Lorie De Leon, Camille Elepaio, Maria Madriaga, Rimel Aggabao, Giezel Diaz-Candido, James Maningo, et al. 2023. Performance of ChatGPT on USMLE: Potential for AI-assisted medical education using large language models. *PLoS digital health* 2, 2 (2023), e0009198.
- [16] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 492–504.
- [17] Van-Hoang Le and Hongyu Zhang. 2022. Log-based Anomaly Detection with Deep Learning: How Far Are We?. In *2022 IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 1356–1367.
- [18] Van-Hoang Le and Hongyu Zhang. 2023. An Evaluation of Log Parsing with ChatGPT. *arXiv preprint arXiv:2306.01590* (2023).
- [19] Van-Hoang Le and Hongyu Zhang. 2023. Log Parsing with Prompt-based Few-shot Learning. In *2023 IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE.
- [20] Zhenhao Li, Chuan Luo, Tse-Hsun Peter Chen, Weiyi Shang, Shilin He, Qingwei Lin, and Dongmei Zhang. 2023. Did We Miss Something Important? Studying and Exploring Variable-Aware Log Abstraction. In *ICSE 2023*.
- [21] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.
- [22] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. 2018. Detecting anomaly in big data system logs using convolutional neural network. In *2018 IEEE 16th Intl Conf on Dependable (DASC/PICOM/DataCom/CyberSciTech)*. IEEE, 151–158.
- [23] Adetokunbo AO Makanku, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1255–1264.
- [24] Weibin Meng, Ying Liu, Federico Zaiter, et al. 2020. Logparse: Making log parsing adaptive through word classification. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 1–9.
- [25] Weibin Meng, Ying Liu, Yichen Zhu, et al. 2019. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.
- [26] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 167–1710.
- [27] Sewon Min, Xinxin Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? *arXiv preprint arXiv:2202.12837* (2022).
- [28] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 575–584.
- [29] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.
- [30] Keqin Peng, Liang Ding, Qihuang Zhong, Li Shen, Xuebo Liu, Min Zhang, Yuanxin Ouyang, and Dacheng Tao. 2023. Towards Making the Most of ChatGPT for Machine Translation. *arXiv preprint arXiv:2303.13780* (2023).
- [31] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
- [32] William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association* 66, 336 (1971), 846–850.
- [33] Gagan Somashkar and Rajat Kumar. 2023. Enhancing the Configuration Tuning Pipeline of Large-Scale Distributed Applications Using Large Language Models (Idea Paper). In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 39–44.
- [34] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 785–794.
- [35] Shimin Tao, Yilun Liu, Weibin Meng, Zuomin Ren, Hao Yang, Xun Chen, Liang Zhang, Yuming Xie, Chang Su, Xiaosong Oiao, Weinan Tian, Yichen Zhu, Tao Han, Ying Qin, and Yun Li. 2023. Biglog: Unsupervised Large-scale Pre-training for a Unified Log Representation. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. 1–11. <https://doi.org/10.1109/IWQoS57198.2023.10188759>
- [36] Shimin Tao, Yilun Liu, Weibin Meng, Jingyu Wang, Yanqing Zhao, Chang Su, Weinan Tian, Min Zhang, Hao Yang, and Xun Chen. 2023. DA-Parser: A Pre-trained Domain-aware Parsing Framework for Heterogeneous Log Analysis. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMP-SCA)*. 322–327. <https://doi.org/10.1109/COMPSCA57700.2023.00050>
- [37] Shimin Tao, Weibin Meng, Yiming Cheng, Yichen Zhu, Ying Liu, Chunling Du, Tao Han, Yongpeng Zhao, Xiangguang Wang, and Hao Yang. 2022. LogStamp: Automatic Online Log Parsing Based on Sequence Labelling. *ACM SIGMETRICS Performance Evaluation Review* 49, 4 (2022), 93–98.
- [38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
- [39] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).
- [40] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).
- [41] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.
- [42] Shenglin Zhang, Weibin Meng, et al. [n.d.]. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *IEEE/ACM 25th International Symposium on Quality of Service (IWQoS'17)*. 1–10.
- [43] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinshe Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.
- [44] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).
- [45] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.