LogEval: A Comprehensive Benchmark Suite for LLMs in Log Analysis

Tianyu Cui[1], Shiyu Ma[1], Ziang Chen[1], Tong Xiao[2], Chenyu Zhao[1], Shimin Tao[3], Yilun Liu[3], Shenglin Zhang[1*], Duoming Lin[1], Changchang Liu[1], Yuzhe Cai[1], Weibin Meng[3], Yongqian Sun[1], Dan Pei[2]

[1]Nankai University, Tianjin, China.
[2]Tsinghua University, Beijing, China.
[3]Huawei, Beijing, China.

*Corresponding author(s). E-mail(s): zhangsl@nankai.edu.cn;
Contributing authors: cuitianyu@mail.nankai.edu.cn;
mashiyu@mail.nankai.edu.cn; 2120240792@mail.nankai.edu.cn;
xiaotong@tsinghua.edu.cn; zhaochenyu@mail.nankai.edu.cn;
taoshimin@huawei.com; liuyilun3@huawei.com;
2114010@mail.nankai.edu.cn; 2113411@mail.nankai.edu.cn;
2212113@mail.nankai.edu.cn; m_weibin@163.com;
sunyongqian@nankai.edu.cn; peidan@tsinghua.edu.cn;

Abstract

Log analysis is vital in Artificial Intelligence for IT Operations (AIOps) and plays a crucial role in ensuring software reliability and system stability. However, challenges such as the absence of comprehensive evaluation standards, inconsistencies in benchmarking practices, and limited exploration of Large Language Models (LLMs) in log-related tasks persist. To address these issues, we introduce *LogEval*, a comprehensive benchmark designed to systematically evaluate LLMs' performance across four key log analysis tasks: log parsing, log anomaly detection, log fault diagnosis, and log summarization. *LogEval* systematically tackles these challenges through the following aspects: (i) it incorporates 4,000 publicly available log entries, spanning diverse tasks and providing a strong foundation for evaluating LLM performance; (ii) it utilizes standardized prompts in both English and Chinese to ensure consistent and objective evaluations, this benchmark covers two experimental paradigms: Naive question-answering (Q&A) and self-consistency (SC) Q&A, under both zero-shot and

few-shot settings, while also considering inference efficiency and average token usage; (iii) it features an open-source, continuously updated platform (https://nkcs.iops.ai/LogEval/) that integrates new LLMs and user-uploaded production data, fostering reproducibility and adaptability in performance comparisons. The experimental results provide valuable insights into the varying strengths of LLMs across different tasks, highlighting opportunities for further optimization and innovation for LLMs in log analysis. Our code repository is available at https://github.com/LinDuoming/LogEval.

Keywords: Log Analysis, Benchmark Suite, Large Language Models, Prompt Engineering

1 Introduction

With the rapid advancement of information technology, software systems have become essential to the operations of businesses and organizations [1]. These systems generate vast amounts of log data that capture operational behavior, status changes, and potential failures [2]. As software systems grow in both scale and complexity, the manual log analysis performed by experts is becoming increasingly difficult and error-prone [3–6]. It is not only time-consuming but also inefficient, often leading to delayed responses to critical system failures [7–9]. Consequently, there is an urgent need for automated log analysis tools capable of quickly providing insights to ensure the reliability and stability of modern large-scale systems [10].

To meet these needs, various automated log analysis tools have been developed, focusing on four primary tasks: log parsing [11–18], log anomaly detection [19–25], log fault diagnosis [3, 26–32], and log summarization [7, 33]. In recent years, deep learning (DL) methods have been widely applied in log analysis to address the limitations of traditional approaches [10, 34]. Unlike traditional machine learning (ML) methods, deep learning is more effective at handling large-scale and complex log data [8, 22, 24], and can automatically extract features in an end-to-end manner, avoiding the constraints of manual feature engineering and fixed rule-based methods [35]. While deep learning-based approaches offer significant improvements over traditional ML methods, they also come with certain challenges [36]. One major limitation is that deep learning models, especially PLMs, require substantial computational resources and large datasets for both pre-training and fine-tuning [8]. Additionally, these models may struggle with domain-specific terminologies, such as the abbreviations commonly found in logs, log events of the same type exhibit different semantics and different log events share many similar words but exhibit opposite [37], which are not typically present in general language corpora. Furthermore, the variability of logs across different systems poses another challenge, as DL models often need to be retrained or fine-tuned frequently to effectively handle new log types.

To address the generalization challenge, data requirements, and retraining issues of DL-based methods, researchers have begun to explore the use of Large Language Models (LLMs) in log analysis tasks [6, 38], as LLMs have demonstrated outstanding

performance in various natural language processing (NLP) tasks such as text generation, language translation, and sentiment analysis. LLMs like GPT-4 [39], LLaMA-2 2023, ChatGLM-4 [41], and Qwen-1.5 [42] have shown promising performance in these tasks. Nevertheless, with the diversification of LLMs, their performance varies across different tasks. As a result, selecting the most appropriate LLM for a given log analysis task has become an important consideration in both research and practice. However, in the field of log analysis, there is currently no comprehensive and systematic evaluation standard or toolkit to help researchers and developers understand and compare the performance of different LLMs across various log analysis tasks. The diverse architectures, model sizes, and applicability of LLMs make selecting the optimal model a complex task that lacks scientific guidance. Therefore, there is an urgent need to construct a unified benchmark that can scientifically assess the performance of different LLMs and provide objective, comprehensive comparisons. To address this, we propose and develop a comprehensive benchmark suite called *LogEval*, designed to evaluate LLMs' performance across various log analysis tasks. The main contributions of this paper are as follows:

- **Diverse Log Dataset Collection**: *LogEval* incorporates log datasets from multiple sources, covering core tasks such as log parsing, log anomaly detection, log fault diagnosis, and log summarization. This curated dataset provides a robust foundation for evaluating LLM performance comprehensively.
- **Unified and Reliable Evaluation**: *LogEval* utilizes standardized English and Chinese prompts to ensure consistent and objective assessments of LLMs. A unified prompt design is introduced for all tasks, minimizing variations caused by differing prompt styles and ensuring fair comparisons. The evaluation spans two paradigms: Naive Q&A and Self-Consistency Q&A under zero-shot setting and few-shot setting, while also considering inference efficiency and token usage.
- **Dynamic Benchmarking Platform**: *LogEval* features an open-source, continuously updated online platform (https://nkcs.iops.ai/LogEval/) that allows dynamic integration of new LLMs and user-uploaded production log data. This platform promotes reproducibility, fairness, and adaptability in performance comparisons, ensuring long-term relevance and scalability. Our code repository is available at https://github.com/LinDuoming/LogEval.

2 Background

Automated log analysis typically involves four core tasks—log parsing, log anomaly detection, log fault diagnosis, and log summarization. Each task addresses specific challenges and plays a crucial role in transforming raw log sequences into actionable insights. Fig. 1 illustrates the sequence of these tasks in the log analysis pipeline. Below, we describe each task, the results depicted in the figure, and the evaluation metrics used to measure their performance.

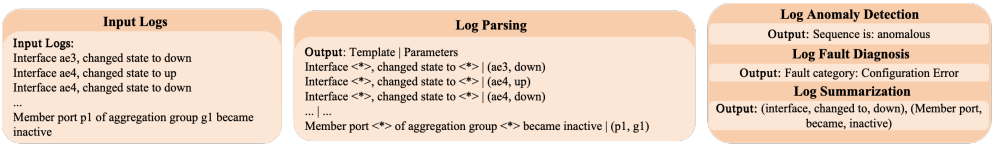

Fig. 1: A demonstration of the log analysis tasks.

Log Parsing: Log parsing is the initial task in the log analysis pipeline. It involves transforming raw log data into a structured format that can be processed by subsequent tasks. The goal of log parsing is to extract relevant components (such as interface states, error messages, or system events) from unstructured logs and represent them in a consistent format, often as key-value pairs or predefined templates. In Fig. 1, the first section illustrates log entries such as interface state changes or member port status updates, which are parsed and organized into templates (e.g., "Interface `<*>`, changed state to `<*>`") in the following table. This structured output enables the identification of recurring patterns or anomalies. **Metrics and Formula**: The parsing performance is evaluated using two key metrics:

- *Parsing Accuracy*: The formula for parsing accuracy is given by

$$\text{Accuracy} = \frac{\text{Correctly Parsed Entries}}{\text{Total Entries}} \times 100\% \tag{1}$$

  where **Correctly Parsed Entries** refers to log lines that exactly match predefined templates, and **Total Entries** refers to all log entries in the dataset.
- *Edit Distance*: The formula for edit distance is given by

$$\text{Edit Distance} = I + D + S \tag{2}$$

  where **Insertions (I)** is the number of characters added to match the template, **Deletions (D)** is the number of characters removed to match the template, and **Substitutions (S)** is the number of character replacements needed.

Log Anomaly Detection: Once logs are parsed, the next step is log anomaly detection. This task aims to identify unusual log entries that may indicate potential issues or system faults. Anomalies are detected based on patterns that deviate from normal system behavior, such as unexpected state changes, errors, or performance issues. In Fig. 1, the second section of the diagram shows the log anomaly detection task. After parsing, each log sequence is examined for anomalous behavior. For example, an unexpected interface state change or an error message could be flagged as anomalous. The detected anomalies are then passed on for further investigation in the fault diagnosis stage. **Metrics and Formula**: The performance of anomaly detection is commonly assessed using the following metrics: - *Precision* (the proportion of detected anomalies that are true positives):

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3}$$

- *Recall* (the proportion of actual anomalies that are correctly detected):

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4}$$

- *F1-score* (the harmonic mean of precision and recall):

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

Where True Positives (TP) are correctly identified anomalies, False Positives (FP) are incorrectly flagged as anomalies, and False Negatives (FN) are missed anomalies.

Log Fault Diagnosis: The log fault diagnosis task aims to identify the root cause of the detected anomalies by analyzing the correlations between log entries. This step often involves mapping anomalies to known fault categories or failure modes. In Fig. 1, the third section illustrates this task, where the system correlates parsed and anomalous log entries to diagnose faults. For example, a change in interface state from "up" to "down" might correlate with hardware failure or misconfiguration. **Metrics**: This task shares the same evaluation metrics with anomaly detection.

Log Summarization: Finally, log summarization condenses large volumes of log data into a concise and interpretable format, highlighting key events that require attention. The goal is to present a summary of the most critical log entries in a format that is easy for system administrators to understand and act upon. In Fig. 1, the log summarization task is shown in the final section, where the relevant log entries identified during the previous stages are summarized. For example, entries like "interface changed state to down" and "member port became inactive" are distilled into a more concise format that provides key insights for further analysis. **Metrics and Formula**: The performance of summarization is commonly assessed using the following metrics:

- *ROUGE-L F1*: The formula for ROUGE-L is given by

$$\text{ROUGE-L} = \frac{|LCS(S,R)|}{|R|} \quad \text{(Recall)} \tag{6}$$

  and the formula for F1 is given by

$$\text{F1} = 2 \times \frac{\text{ROUGE-L} \times P_{\text{LCS}}}{\text{ROUGE-L} + P_{\text{LCS}}} \tag{7}$$

  where **LCS** refers to the Longest Common Subsequence between summary (S) and reference (R), and **P__LCS** refers to the Precision of LCS, which is $\frac{|LCS(S,R)|}{|S|}$.
- *Threshold Accuracy*: The formula for threshold accuracy is given by

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} \mathbb{I}(\text{ROUGE-L}_i \geq \theta)}{n} \times 100\% \tag{8}$$

where θ is the similarity threshold (typically 0.7-0.9), and $\mathbb{I}$ is the indicator function (1 if condition met, 0 otherwise).

In this study, we carefully selected evaluation metrics tailored to the nature of each log analysis task to ensure a comprehensive, objective, and practically meaningful assessment. For tasks such as log anomaly detection and log fault diagnosis, we adopt Accuracy and F1-score as the primary evaluation metrics. Accuracy reflects the overall correctness of predictions, especially when class distributions are relatively balanced. In contrast, F1-score, which harmonizes precision and recall, is more suitable for scenarios with class imbalance, a common challenge in log analysis where normal logs significantly outnumber anomalies . Although metrics like ROC-AUC [43] and PR-AUC were considered, they were ultimately excluded due to their reliance on probabilistic outputs and threshold variation, which are not directly applicable to classification-style outputs generated by LLMs through prompt engineering. For tasks such as log parsing and log summary, where multiple valid outputs may exist, we use ROUGE-L [44] and Edit Distance to measure semantic and structural similarity between the generated and reference texts. ROUGE-L evaluates the longest common subsequence between two texts, capturing the structural overlap, which is ideal for assessing key information extraction in templates or summaries. Edit Distance quantifies the number of character-level operations needed to transform the generated output into the reference text, making it especially useful for tasks with strict format constraints such as log parsing. We also considered BLEU [45], a common n-gram based metric, but it was not chosen due to its sensitivity to word order and reduced robustness in tasks with high output variability like summarization and parsing. To comprehensively assess the efficiency and usability of LLMs, we additionally incorporate Average Number of Tokens and Inference Time. These two metrics are critical for real-world applications—longer outputs often imply higher computational and memory costs, and longer inference times directly affect system responsiveness. While alternative system-level metrics were considered, they were excluded due to cross-platform inconsistency and difficulty in reproducibility. Instead, token count and time are universally measurable and meaningful across different LLMs and environments. The metrics and formulas presented above help evaluate the performance of each task, ensuring that the system can quickly identify issues, diagnose faults, and generate actionable summaries for system operators.

3 Related Work

In this section, we first discuss the existing evaluations of LLMs in general NLP tasks, as our research also focuses on evaluating LLMs. These evaluations highlight the broad applications of LLMs in NLP. However, there is currently no systematic evaluation of LLMs specifically in the field of log analysis. Therefore, we also examine the applications of LLMs in log analysis tasks, providing context for our research and emphasizing the potential of LLMs in this area, as well as the current lack of standardized evaluation frameworks.

3.1 Evaluation of LLMs in General NLP Tasks

The evaluation of LLMs in general NLP tasks has diversified, as these models have become capable of handling increasingly complex and varied tasks. Such evaluations now not only measure basic linguistic understanding and generation, but also delve into nuanced capabilities such as reasoning, adaptability to different tasks, and the use of domain-specific knowledge. We categorize these evaluations into two main areas: general domain evaluations and specialized domain evaluations.

Evaluations in General Domain: Comprehensive assessments are designed to evaluate the broad capabilities of LLMs across multiple dimensions. For instance, HELM [46] utilizes a diverse set of metrics to assess LLMs in 42 unique scenarios, providing insights into their general linguistic abilities and reasoning skills. BIG-bench [47] extends this by including tasks that challenge the models' understanding of common sense, logic, and even creativity.

Evaluations in Specialized Domains: These assessments focus on evaluating LLMs' performance in domains requiring specialized knowledge. For example, FinEval [48] measures financial acumen, while MultiMedQA [49] tests medical knowledge using datasets derived from professional exams and consultation records. Similarly, Huatuo-26M [50] evaluates medical consultation capabilities, reflecting real-world medical inquiry handling. NetOps [51] focuses on network operations, and tests LLMs with tasks that mimic real-world challenges in network management. OpsEVAL [52] assesses the ability of LLMs to manage IT operations, through a set of structured tasks, in both Chinese and English. RepairBench [53] establishes an execution-based leaderboard for program repair, evaluating LLMs on real-world Java bugs through test-suite validation and syntactic analysis, providing standardized assessment for AI-driven code repair.

3.2 Applications of LLMs in Log Analysis Tasks

With the growing application of LLMs in log analysis tasks, researchers are increasingly exploring how these models can improve key processes such as log parsing and anomaly detection.

Log Parsing: LILAC [54] leverages the in-context learning (ICL) capabilities of LLMs by employing a hierarchical candidate sampling algorithm to select high-quality examples for log template generation. It also introduces an adaptive parsing cache to store and refine templates generated by LLMs, reducing query frequency and ensuring template consistency. LogParser-LLM [6] combines the semantic understanding capabilities of LLMs with a prefix tree clustering approach. It utilizes LLMs to process the semantic information of logs and performs online log parsing without requiring hyperparameter tuning or labeled data. DivLog [55] uses the ICL capabilities of LLMs to select diverse offline log samples as candidate examples, it then constructs prompts to generate log templates for target logs, enabling unsupervised log parsing. ECLIPSE [56] integrates the semantic understanding capabilities of LLMs with data-driven template matching algorithms to handle cross-lingual log parsing. LLMs are used to extract semantic information from log keywords, improving parsing efficiency in cross-lingual environments. LogPrompt [38] employs the zero-shot capabilities of

LLMs and advanced prompt strategies to perform log parsing tasks, it enhances LLM interpretability and flexibility, enabling log analysis without relying on training data.

Log Anomaly Detection: LogExpert [5]) integrates LLMs with domain knowledge from technical forums such as Stack Overflow. LLMs are utilized to parse relevant technical solutions and automatically generate recommended resolutions for anomalous logs, reducing the need for manual intervention. SeaLog [57] employs LLMs, such as ChatGPT, to provide expert-level feedback that enhances the accuracy of its Trie-based Detection Agent (TDA) for real-time anomaly detection, allowing the system to adapt to evolving log data more effectively. LogGPT [58] utilizes ChatGPT's language understanding and knowledge transfer capabilities through prompt-based techniques for log anomaly detection, exploring the application of large-scale corpora knowledge to the processing of complex log data. LogPrompt [38] everages the zero-shot capabilities of LLMs through a set of advanced prompting strategies specifically designed for log anomaly detection tasks. This approach enables LLMs to perform detection without relying on training data, while also offering interpretability of the results.

Other Applications: In addition to log parsing and anomaly detection, LLMs have potential applications in various aspects of log analysis. For example, Face It Yourselves [59] introduces a two-stage, LLM-based framework for diagnosing configuration errors through log analysis. This framework, called LogConfigLocalizer, leverages LLMs to help end-users, particularly those without source code access, identify the root causes of configuration issues by analyzing logs. UniLog [60] employs the ICL paradigm of LLMs to automatically generate appropriate log statements without requiring any fine-tuning. By using prompts with a few demonstration examples, LLMs can determine log positions, verbosity levels, and generate log messages, thus aiding in system maintenance and troubleshooting. LLM4Sec [61] utilizes various LLM architectures, such as BERT, RoBERTa, and GPT-2 [62], to analyze log files for cybersecurity purposes. These LLMs are fine-tuned for specific log types to enhance security log analysis. Summary Cycles [63] applies LLMs, specifically ChatGPT, to summarize interaction logs in collaborative intelligence analysis. LLMs are used iteratively with recursive summarization techniques to extract key entities, topics, and summaries from user interaction sequences.

However, currently there is no dedicated benchmark for evaluating the performance of different LLMs in various log analysis tasks. This work bridges this gap and proposes an evaluation framework for LLMs in log analysis. Our evaluation efforts intend to understand the strengths and limitations of different LLMs in various log analysis tasks, while providing valuable resources and guidance for the log analysis domain, promoting the effective application of LLMs in real-world scenarios.

4 *LogEval* Benchmark

In this section, we first introduce the platform architecture and the technical stack behind *LogEval*, which provide the foundation for its operation and scalability. The architecture is designed to ensure flexibility and extensibility, supporting a wide range of log analysis tasks. Following this, we describe the key components of our benchmark and its specific evaluation process.

4.1 Platform Architecture and Technology Stack

The design and implementation of the *LogEval* platform rely on a powerful and flexible technology stack that ensures high scalability, efficient processing, and easy extensibility. Below, we highlight the core aspects of the platform's architecture and the tools chosen for log analysis.

4.1.1 Platform Architecture

The *LogEval* platform is primarily developed using Python 3.9.6 and runs on Amazon EC2 servers. Flask 3.0.3 is used for building API interfaces, enabling the platform to handle concurrent requests efficiently. The modular architecture allows components to be updated or replaced without disrupting the entire system, ensuring the platform's scalability. Key features of the architecture include:

- **Multilingual Support:** Integration of Flask-Babel 4.0.0 enables bilingual support (Chinese and English).
- **Flexible Data Access:** The platform uses the json 2.0.9 package for managing data in JSON format and Pandas 2.2.2 for data processing. These tools are chosen for their robustness and ability to handle large-scale data efficiently.

4.1.2 Extensibility and Scalability Design

To ensure flexibility, the *LogEval* platform is designed to scale and integrate with new features. The following mechanisms support its extensibility:

- **Plugin Mechanism:** Users can easily integrate new LLM models or log processing techniques by adding custom plugins. This allows for seamless adaptation to future requirements.
- **Modular Architecture:** The platform's core functionalities, such as log parsing and fault diagnosis, are designed as independent modules. New modules can be added as needed without modifying the underlying system.
- **API Interfaces:** The platform provides open API interfaces to enable users to integrate with external systems and extend functionality. For example, new LLM models can be integrated via simple API calls, allowing users to switch models based on task requirements.
- **Hardware Configurations for Performance Testing:** The platform's performance across various tasks may be influenced by the underlying hardware configurations. For local deployments, the platform uses a high-performance setup, including 8 NVIDIA A6000 GPUs, each equipped with 48GB of memory, and Intel Xeon processors. For external API calls, the platform uses the official recommended API interfaces provided by the API provider, ensuring consistency and fairness in performance evaluations.

This scalable and modular design ensures that *LogEval* can adapt to future needs, whether it involves adding new features, models, or data sources.

4.2 Evaluation Benchmark

In this section, we introduce the evaluation benchmark *LogEval*, which is designed to assess the performance of various LLMs in performing log analysis tasks. As shown in Fig. 2.

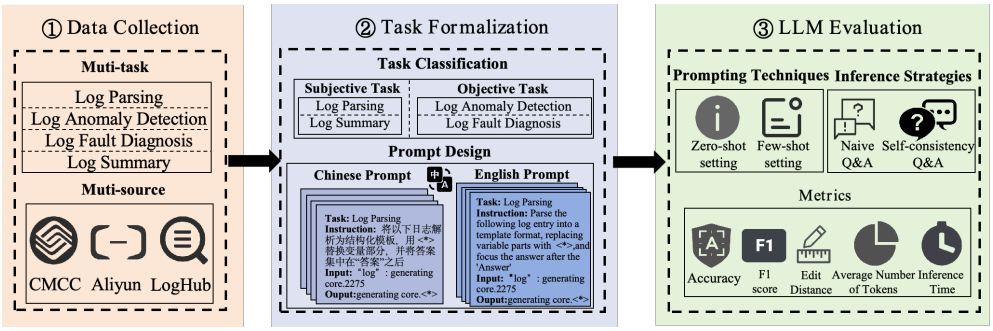

Fig. 2: The framework of *LogEval*

4.2.1 Data Collection

Data Collection is the foundational step that supports the entire evaluation process. To ensure comprehensive assessment, we curated datasets from diverse sources and tasks, covering a wide range of log processing needs. We designed four core tasks to evaluate LLM capabilities across different log analysis scenarios. In addition, we integrated multi-source datasets to enhance the framework's adaptability and generalizability:

- **Aliyun:** The dataset contains a total of 299,817 logs, which are grouped by serial number and sorted chronologically. The dataset captures three main fault categories, and their root causes, as flagged by operation and maintenance staff, include issues such as high CPU temperature, memory leaks, and hardware crashes. The dataset provides a real-world perspective on server failures, enhancing its value for anomaly detection and fault diagnosis tasks.
- **CMCC:** The dataset consists of 482,515 logs collected from OpenStack's [64] OpenVSwitch services, distributed across 493 nodes in a high-performance computing cluster. This dataset spans six fault categories, with root causes ranging from software bugs to resource underprovisioning and unexpected process restarts. The dataset's scale and complexity, derived from an industrial OpenStack environment, make it an excellent benchmark for evaluating anomaly detection methods.
- **LogHub:** [65] This open-source repository contains large-scale logs from multiple open-source projects, covering real-world scenarios in industries such as server management and cloud computing. These datasets not only feature extensive diversity but also include detailed annotations, providing a reliable foundation for evaluating the performance of LLMs in log processing tasks.

By combining multi-task and multi-source datasets, the *LogEval* framework simulates real-world production environments, providing a solid foundation for comprehensive evaluation of LLM performance in log processing.

4.2.2 Task Formalization

The purpose of this step is to structure the log analysis tasks to match the input requirements of LLMs, thereby achieving effective LLM evaluation and comparison. Task classification is a core step in the formalization process. Based on the nature of task evaluation, we categorize log analysis tasks into two main types:

- **Subjective Tasks:** These include Log Parsing and Log Summary. These tasks do not have a unique correct answer and rely on semantic understanding and content generation for assessment.
- **Objective Tasks:** These include Log Anomaly Detection and Log Fault Diagnosis. These tasks have definite answers, allowing for straightforward quantitative evaluation.

Prompt design is another key aspect to ensure that LLMs can understand and effectively complete each task, each prompt consists of the following four elements:

- **Task:** Clearly specifies the log analysis task to be evaluated, such as Parsing (Log Parsing), Detection (Log Anomaly Detection), Diagnosis (Log Fault Diagnosis), and Summary (Log Summary).
- **Instruction:** Thoroughly describes the task requirements, guiding the LLM's behavior, for example, instructing the LLM on how to transform a log entry into a structured format.
- **Input:** Provides the log entry or sequence to be analyzed, presented in a uniform format, prefixed with explicit labels like "log entry:" or "log sequence:".
- **Output:** Defines the format of the response to ensure that the LLM's output meets the expected standards.

To evaluate LLMs' performance across different languages, we have prepared prompts in both Chinese and English for each task. Additionally, we provide each task with 15 different prompts to minimize the influence of prompt variations. Table 1 gives three different English prompts for each task.

4.2.3 LLM Evaluation

This section evaluates LLMs' capabilities in log analysis through systematic benchmarking. We first introduce the evaluation strategies, then detail the selected models. Our benchmarking framework combines two evaluation strategy:

Inference Strategy

We employ two different inference strategies to process and interpret the responses generated by LLMs: Naive Q&A and Self-Consistency Q&A. These strategies aim to investigate the stability of LLM outputs.

- Naive Q&A: This strategy involves a single model invocation per query, and the generated answer is directly treated as the final prediction. Naive Q&A is simple and efficient, and it is especially suitable for tasks with subjective nature and diverse valid answers, such as log parsing and summarization.

Table 1: Three Different English Prompts for Each Task

Tasks	English Prompts
Log Parsing	1. Parse the following log into a template format, replacing variable parts with <*>: [log] 2. Convert the following log into a standardized template by identifying and replacing the variable parts with <*>: [log] 3. Transform the raw log [log] into a log template by replacing variable segments with <*>
Log Anomaly Detection	1. Review and mark the log entry as "normal" or "anomalous", only output "normal" or "anomalous" 2. Analyze the log content, classify it as "normal" or "anomalous", only output "normal" or "anomalous" 3. Check the log entry, and determine if it belongs to the "normal" or "anomalous" category, only output "normal" or "anomalous"
Log Fault Diagnosis	1. In our data scenario, there are several types of faults {fault types}. Analyze the log [log] and identify the type of fault that occurred. Only output the fault type 2. In our data scenario, there are several types of faults {fault types}. Based on the information in the log [log], determine which type of fault the log represents. Only output the fault type 3. In our data scenario, there are several types of faults {fault types}. Use the detailed information provided by the log [log] to conduct an in-depth analysis to determine the category of the fault. Only output the fault type
Log Summary	1. Analyze the following 20 logs [log], extract key information, phrases, sentences, or recurring content to generate a summary, and only output the summary 2. Extract the most important events, phrases, and activities or recurring content from the following 20 logs [log], create a concise log overview, only output the summary 3. Extract key events, sentence phrases, or recurring information from the following 20 logs [log] to form a comprehensive summary, only output the summary

- Self-Consistency Q&A: To enhance the stability and accuracy of model outputs, Self-Consistency Q&A performs multiple model invocations on the same query (set to 5 times in our study), generating multiple answers. The most frequent answer among these is selected as the final result through a voting mechanism. This approach effectively reduces the randomness of single-shot outputs and is particularly well-suited for tasks with objective ground truth, such as log anomaly detection and fault diagnosis.

Prompting Technique

We use various settings to evaluate LLMs on *LogEval* to get a comprehensive overview of their performance. We evaluate LLMs in zero-shot and few-shot (5-shot) settings.

- **Zero-shot setting**: This technique involves presenting the LLM with a task without prior examples, thereby testing its ability to adapt to new situations based on its pre-existing knowledge. It is a measure of the LLM's capacity to generalize from its training data to unseen tasks. The examples for the zero-shot setting can be found in Table 2.
- **Few-shot setting**: The LLM is provided with a limited number of exemplars before being asked to perform the task. Few-shot prompting helps the model better capture task-specific patterns or structures within the log data, often leading to improved performance compared to zero-shot. The examples for the few-shot setting can be found in Table 3.

We select 12 state-of-the-art LLMs covering diverse architectures and accessibility modes, as summarized in Table 4.

Table 2: Zero-shot prompts for the four log analysis tasks.

Task	Parsing	Anomaly Detcetion	Diagnosis	Summary
Instruction	Parse the following log entry into a template format, replacing variable parts with <*>, and focus the answer after the keyword 'Answer'	Please review the log entry and explicitly mark it as 'normal' or 'anomalous' , only output 'normal' or 'anomalous'	In our data scenario, there are three types of faults: Processor CPU Caterr, Memory Throttled Uncorrectable Error Correcting Code, Hard Disk Drive Control Error Computer System Bus Short Circuit Programmable Gate Array Device Unknown. Analyze the log entry and identify the type of fault that occurred. Only output the fault type.	Analyze the following 20 logs, extract key information, phrases, sentences, or recurring content to generate a summary, only output the summary.
Input	log entry: synchronized to 10.100.28.250, stratum 3	log entry: instruction cache parity error corrected	log entry: Processor #0xfa | Configuration Error| Asserted	log sequence:[blockMap updated: 10.251.193.175:50010 is added to blk_-3864576029521084501 size 3540711, ...
Output	synchronized to <*>, stratum <*>"	normal	Processor CPU Caterr	blockMap updated; PacketResponder terminating; Received block;

Table 3: Few-shot prompts for the four log analysis tasks.

Task	Parsing	Diagnosis
Instruction	Parse the following log entry into a template format, replacing variable parts with <*>, and focus the answer after the keyword 'Answer'. For example:log entry: no floppy controllers found, answer: 'no floppy controllers found'; log entry: 13 tree receiver in re-synch state event(s) (dcr 0x0185) detected over 4562 seconds, answer: '<*> tree receiver <*> in re-synch state event(s) (dcr <*>) detected over <*> seconds'; log entry: ... autorun DONE., answer: '... autorun DONE.'; log entry: 2 L3 EDRAM error(s) (dcr0x0157) detected and corrected over 282 seconds, answer: '<*> L3 EDRAM error(s) (dcr <*>) detected and corrected over <*> seconds'; log entry: probe of vesafb0 failed with error-6, answer: 'probe of vesafb0 failed with error <*>'."	In our data scenario, there are three types of faults {Processor CPU Caterr, Memory Throttled Uncorrectable Error Correcting Code, Hard Disk Drive Control Error Computer System Bus Short Circuit Programmable Gate Array Device Unknown}. Analyze the log entry and identify the type of fault that occurred. Only output the fault type. For Example: log entry: Temperature CPU0_ Margin_Temp | Lower Critical going low | Asserted| Reading -16 < Threshold 0 degrees C answer: 'Processor CPU Caterr'; log entry: MemoryCPU1E0_ DIMM_Stat | Correctable ECC | Asserted answer: 'Memory Throttled Uncorrectable Error Correcting Code'; log entry: System Boot Initiated BIOS_Boot_ Up | Initiated by power up |Assertedanswer: 'Hard Disk Drive Control Error Computer System Bus ShortCircuit Programmable Gate Array Device Unknown
Input	log entry: synchronized to 10.100.28.250, stratum 3	log entry: Processor #0xfa | Configuration Error| Asserted
Output	synchronized to <*>, stratum <*>"	Processor CPU Caterr

Table 4: LLMs Chosen for Evaluation

Model	Creator	Size	Access
GPT-4 [39]	OpenAI	undisclosed	Commercial
GPT-3.5 [66]	OpenAI	undisclosed	Commercial
Claude-3-Sonnet [67]	Anthropic	undisclosed	Commercial
Gemini-Pro [68]	Google	undisclosed	Commercial
Mistral [69]	Mistral	7B	Open-source
InternLM2-Chat [70]	Shanghai AI Laboratory	7B/20B	Open-source
DevOps-Model-Chat [71]	CodeFuse	7B/14B	Open-source
AquilaChat [72]	BAAI	7B	Open-source
ChatGLM-4 [41]	Tsinghua	undisclosed	Commercial
LLaMA-2 [40]	Meta	7/13/70B	Open-source
Qwen-1.5-Chat [42])	Alibaba Cloud	7/14/72B	Open-source
Baichuan2-Chat [73]	Baichuan Intelligence	13B	Open-source

5 Evaluation Results

In this section, we aim to explore the following key aspects of LLMs' performance in log analysis tasks:

- **RQ1:** What is the overall performance of different LLMs when applied to various log analysis tasks?
- **RQ2:** How do LLMs perform under Naive Q&A settings across different log analysis tasks?

- **RQ3:** How do LLMs perform under Self-Consistency Q&A settings in the context of log analysis tasks?
- **RQ4:** What is the impact of inference time and the average number of tokens on the performance of LLMs?
- **RQ5:** How do factors such as parameter size and language choice influence the performance of LLMs in log analysis tasks?

5.1 RQ1: Overall Performance

To evaluate the performance of various LLMs on different log analysis tasks, we conducted a comparative analysis of their Naive Q&A accuracy under both zero-shot and few-shot settings. The results are shown in Fig. 3 and Fig. 4, respectively. For the sake of simplicity, we use the abbreviation of each task in these two and subsequent figures, *i.e.*, we use "Parsing" instead of "Log Parsing", "Detection" instead of "Log Anomaly Detection", "Diagnosis" instead of "Log Fault Diagnosis", and "Summary" instead of "Log Summary". From Fig. 3 and Fig. 4, we have the following findings for each task:

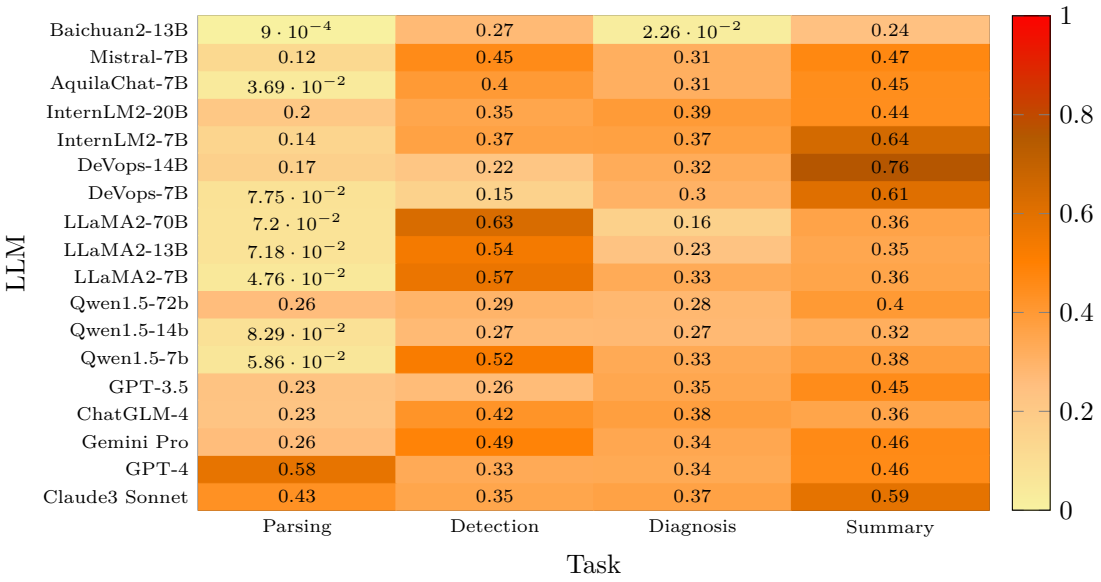

Fig. 3: Accuracy in zero-shot Naive Q&A across four tasks.

- **Log Parsing**: For log parsing, GPT-4 and Claude3 Sonnet demonstrate outstanding performance in both zero-shot and few-shot settings, with GPT-4 achieving the highest parsing accuracy under the few-shot condition, showcasing its exceptional parsing capabilities. Gemini Pro also exhibits strong adaptability in the few-shot setting, achieving a high level of parsing accuracy, which positions it as a competitive and promising LLM for this task.

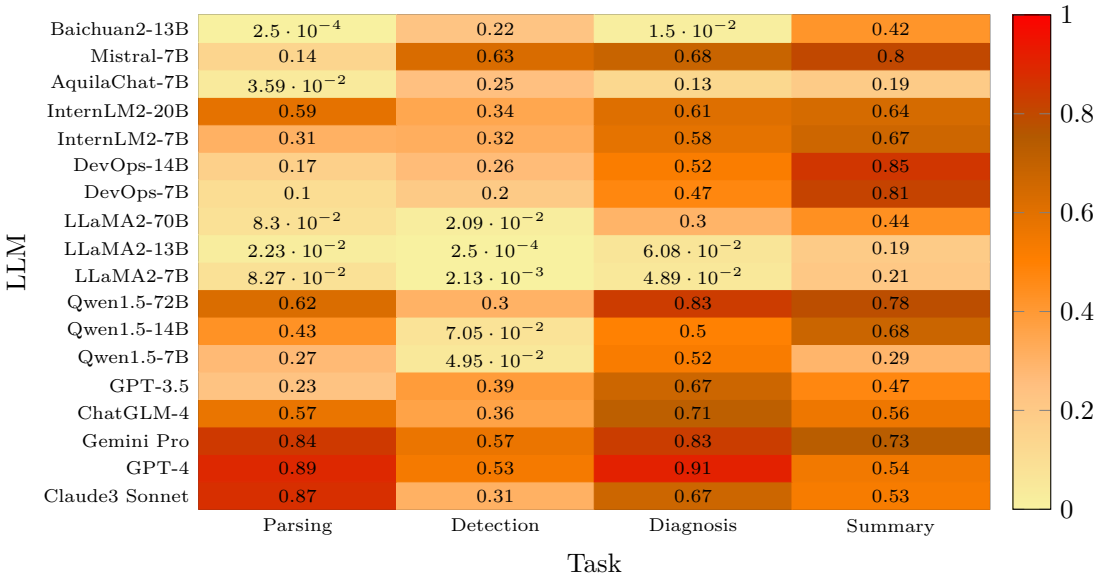

Fig. 4: Accuracy in few-shot Naive Q&A across four tasks

- **Log Anomaly Detection**: In the log anomaly detection task, LLaMA2-70B performs better than other LLMs in the zero-shot setting, but it still lags slightly behind GPT-4 and Claude3 Sonnet in overall performance. In the few-shot setting, Mistral-7B shows a significant improvement, demonstrating strong contextual learning abilities, making it the standout LLM in this task. Gemini Pro also performs well in the few-shot setting, showcasing its adaptability to different prompt conditions, making it suitable for applications in dynamic data environments.
- **Log Fault Diagnosis**: In the log fault diagnosis task, performance differences among LLMs in the zero-shot setting are relatively small; however, Baichuan2-13B and the LLaMA2 series LLMs show relatively weaker performance in this task. In the few-shot setting, GPT-4 shows a marked improvement, establishing itself as the best choice for this task, while Gemini Pro and Qwen1.5-72B also exhibit excellent diagnostic capabilities. These results suggest that GPT-4 can effectively enhance fault diagnosis accuracy under few-shot conditions, making it an ideal LLM for complex diagnostic tasks.
- **Log Summary**: In the log summarization task, the DeVops series LLMs perform well in both zero-shot and few-shot settings, showing their advantage in summary generation. In the few-shot setting, Mistral-7B and Qwen1.5-72B show significant improvement, demonstrating the ability to generate high-quality log summaries with limited prompts. These LLMs have application potential in scenarios requiring high-quality log summarization, especially where limited data is available.

We further compare the accuracy of Commercial and Open-source LLMs, in accordance with the access types listed in the "Access" column of Table 4. The results are shown in Fig. 5 and Fig. 6, from which we can draw the following key findings:

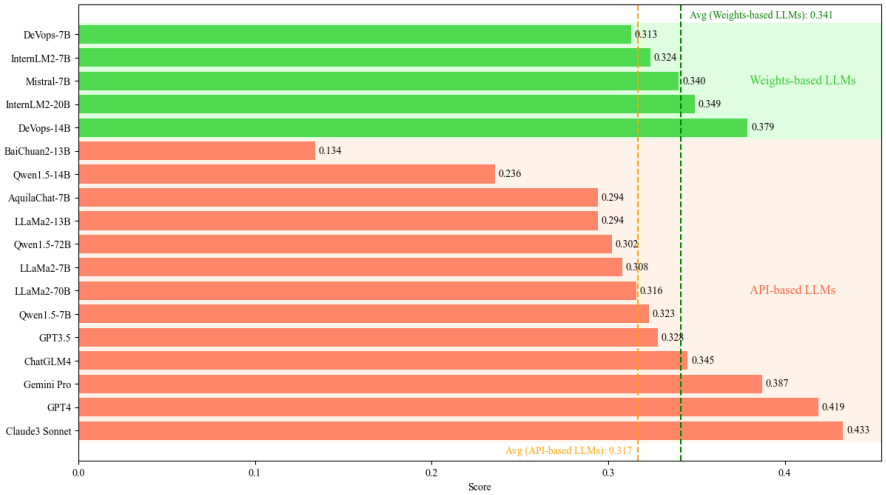

Fig. 5: Overall Performance in zero-shot Naive Q&A

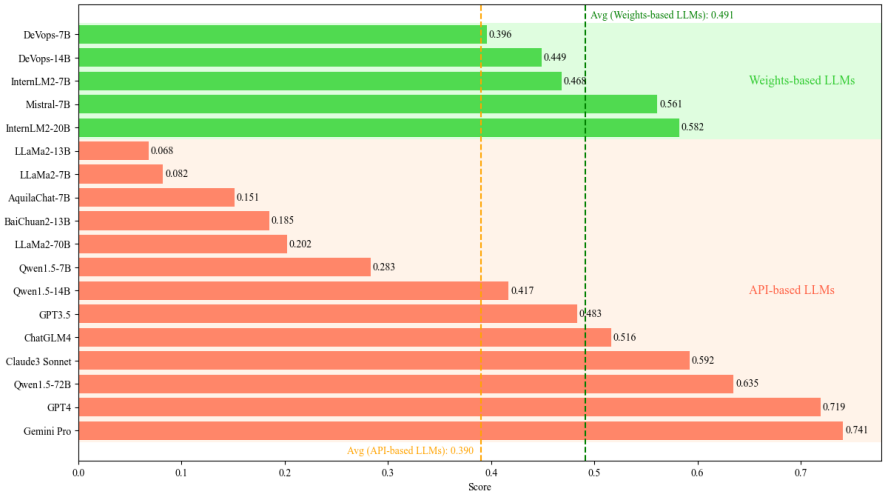

Fig. 6: Overall Performance in few-shot Naive Q&A

- **Zero-shot Setting Analysis**: In zero-shot scenarios, weight-based LLMs generally outperform API-based LLMs, with InternLM2-20B and Mistral-7B standing out for their high accuracy, demonstrating the stability and superior performance of weight-based LLMs in local runtime environments. Among the API-based LLMs, Claude3 Sonnet and GPT-4 show relatively stable performance, indicating that in multi-task scenarios, these LLMs can deliver reliable performance under zero-shot conditions, making them suitable for generic task applications that do not require fine-tuning.

- **Few-shot Setting Analysis**: In few-shot settings, weight-based LLMs show significant performance improvements, with InternLM2-20B and Mistral-7B exhibiting high adaptability with few-shot prompts. API-based LLMs also see noticeable improvement in the few-shot setting, especially with Gemini Pro and GPT-4 achieving high few-shot accuracy, demonstrating strong adaptability. However, weight-based LLMs demonstrate a more pronounced capacity for adaptation in few-shot learning, making them well-suited for complex task scenarios requiring frequent updates and optimizations. In contrast, API-based LLMs, with limited fine-tuning flexibility, are better suited for applications requiring stability and immediate responsiveness.

5.2 RQ2: Naive Q&A Performance

To investigate the performance of various LLMs in Naive Q&A across different log analysis tasks, we conducted a comparative analysis of their performance under both zero-shot and few-shot settings. This section examines the results for each task, highlights the strengths and weaknesses of different models, and explores the potential factors influencing their performance,we present the following findings for each task.

5.2.1 Naive Q&A results on Log Parsing

We evaluated the performance of various LLMs on Naive Q&A log parsing task in both zero-shot and few-shot settings. The following conclusions can be drawn:

- **Few-shot learning consistently boosts LLM accuracy:** Across most LLMs, few-shot learning substantially improves accuracy compared to zero-shot settings. This improvement is particularly notable in high-performing LLMs such as GPT-4 and Claude3-Sonnet, indicating that few-shot learning can effectively enhance LLM adaptability to complex log parsing tasks.
- **GPT-4 and Claude3-Sonnet excel in multiple parsing tasks:** Among the evaluated LLMs, GPT-4 and Claude3-Sonnet consistently deliver high performance across both Chinese and English log parsing tasks in zero-shot and few-shot settings. Their robust accuracy and low Edit Distance suggest strong generalization and adaptability across languages and parsing scenarios.
- **LLM performance scales with LLM size and architecture:** The performance data reveals that larger, more sophisticated LLMs, such as GPT-4 and Claude 3-Sonnet, consistently outperform smaller LLMs, including BaiChuan2-13B and AquilaChat-7B. This scaling effect underscores the advantage of larger LLMs with advanced architectures in capturing complex patterns in log parsing tasks, while smaller LLMs struggle to generalize and adapt effectively.

5.2.2 Naive Q&A results on Log Anomaly Detection

We evaluated the performance of various LLMs on Naive Q&A log anomaly detection task in both zero-shot and few-shot settings. The following conclusions can be drawn:

- **Limited Impact of Few-shot Learning**: In both Chinese and English tasks, few-shot learning does not significantly outperform zero-shot learning in log anomaly

detection. This may be due to the LLMs' tendency to reproduce the answers found in the few-shot examples, thereby failing to effectively identify new anomaly cases. Table 5 illustrates some specific examples of anomaly detection tasks where the LLM repeats answers given in the samples, reflecting the LLM's performance in real-world applications. This indicates that simply increasing the number of samples may not significantly enhance LLM performance, especially when dealing with imbalanced datasets where anomaly classes are underrepresented.

- **Sensitivity to Prompt Language**: The differences in performance between Chinese and English prompts are not substantial overall, but some LLMs do exhibit varying performance levels depending on the prompt language. For example, certain LLMs, such as LLaMA2-70B, show lower F1-Score under Chinese prompts, particularly in the few-shot setting, where an F1-Score of 0 was observed. This suggests that these LLMs have limited generalization capabilities when processing Chinese prompts, highlighting the sensitivity of the LLM to the prompt language.
- **Challenges with Complex or Domain-Specific Questions**: In zero-shot settings, LLMs often struggle with addressing complex or domain-specific questions, resulting in vague or uncertain outputs. As illustrated by the BaiChuan2-13B model's performance on log analysis tasks (Table 6), even high-performing models may fail to accurately classify log entries without sufficient domain knowledge. Integrating domain-specific information into the training process can significantly improve comprehension and response accuracy for specialized tasks like log analysis.

Table 5: Few error examples on Log Anomaly Detection

"Task": Detection	**"Task":** Detection	**"Task":** Detection	**"Task":** Detection
"label": "normal"	**"label":** "normal"	**"label":** "normal"	**"label":** "normal"
"predict": "log1: normal log2: normal log3: anomalous 4: anomalous 5: normal 6 anomalous"	**"predict":** "normal normal anomalous normal"	**"predict":** "'normal' 'normal' 'normal' 'abnormal'"	**"predict":** "normal normal normal abnorma anomalous"

5.2.3 Naive Q&A results on Log Fault Diagnosis

We evaluated the performance of various LLMs on Naive Q&A log fault diagnosis task in both zero-shot and few-shot settings. The following conclusions can be drawn:

- **Effectiveness of Few-shot Learning:** Few-shot learning markedly enhances accuracy and F1-Score across most LLMs. High-performing LLMs, such as GPT-4 and Qwen1.5-72B, show significant improvements in the few-shot setting, highlighting the value of providing examples in fault diagnosis. However, some smaller LLMs,

Table 6: A failed zero-shot example by BaiChuan2-13B

"Task": "Detection"	**"Task":** "Detection"
"label":"anomalous"	**"label":**"normal"
"predict": "anomalous"	**"predict":** "Based solely on the information provided in the log entry, it is not possible to definitively classify it as 'normal' or 'anomalous'. The log entry mentions "double-hummer alignment exceptions", but without additional context or information about what this means within the specific system or application being monitored, it is difficult to determine whether this is expected behavior or an indication of a problem. Therefore, I would classify this log entry as 'unclear' or 'inconclusive' rather than 'normal' or 'anomalous'. It may be necessary to gather more information or consult additional logs or documentation in order to make a more accurate classification."

like the LLaMA series, exhibit limited benefits from few-shot learning, indicating their difficulty in adapting to complex tasks through minimal examples.

- **Superiority of GPT in Complex Tasks:** Among the evaluated LLMs, GPT-3.5 and GPT-4 perform exceptionally well in few-shot log fault diagnosis, with both LLMs achieving an F1-Score above 0.9. GPT-4's superior performance in both Chinese and English tasks suggests that it can effectively generalize in complex fault diagnosis scenarios, making it ideal for critical applications requiring high accuracy.
- **Limitations of Smaller LLMs:** Smaller LLMs with fewer parameters exhibit significant limitations in handling complex fault diagnosis tasks. Their reduced capacity to capture diverse fault types, particularly in zero-shot settings, leads to inaccurate predictions. Table 7 showcases several failure examples from 7B models. For instance, when tasked with diagnosing "Processor CPU Caterr," the model inaccurately predicts "Processor CPUR," and for "Mirror does not specify output," the model erroneously outputs "Mirror specified output." These issues demonstrate that smaller models struggle to capture the deeper, intricate features of log data, resulting in lower reliability of their fault diagnosis predictions.

5.2.4 Naive Q&A results on Log Summary

We evaluated the performance of various LLMs on Naive Q&A log summary task in both zero-shot and few-shot settings. The following conclusions can be drawn:

- **Effectiveness of Few-shot Learning:** The majority of LLMs show significant improvement in log summarization performance when transitioning from zero-shot to few-shot settings. This suggests that few-shot learning enables LLMs to better grasp and adapt to the summarization context, enhancing both accuracy and F1-Score. However, certain LLMs, especially in the LLaMA series and AquilaChat-7B, display a counterintuitive trend, with performance sometimes higher in zero-shot than in few-shot settings, possibly due to noise introduced by few-shot examples.

Table 7: Few error examples on Log Fault Diagnosis

"Task": Diagnosis	**"Task":** Diagnosis	**"Task":** Diagnosis
"label":"Processor CPU Caterr"	**"label":** "Mirror does not specify output"	**"label":** "Processor CPU Caterr"
"predict": "answer:'Processor CPUR';"	**"predict":** "Mirror specified output"	**"predict":** "1. Processor CPU Cater 2. Memory Throttled Uncorrectable Error Correcting Code 3. None 4. Hard Disk Drive Control Error Computer System Bus Short Circuit Programmable Gate Array Device Unknown"

- **Strong Performance of DeVops:** The DeVops-7B and DeVops-14B models consistently outperform others across both zero-shot and few-shot settings. DeVops-14B, in particular, demonstrates exceptional summarization capabilities, making it well-suited for applications where accuracy and robustness in log summarization are critical.
- **Task-Specific LLM Performance Diversity:** In different tasks and language settings, specific LLMs exhibit notable performance variations, highlighting their adaptability and limitations in particular tasks or languages. For example, Gemini Pro performs exceptionally well in few-shot English tasks, demonstrating high adaptability, but shows weaker performance in zero-shot Chinese tasks. Similar trends are observed in LLMs like Claude 3-Sonnet. These results suggest that variations in LLM performance across tasks may reflect the impact of optimization focus and training data.

5.3 RQ3: Self-Consistency Q&A Performance

To evaluate the capability of various LLMs in Self-Consistency Q&A for log anomaly detection and log fault diagnosis, as well as self-consistency in LLM robustness performance, we conducted experiments under zero-shot and few-shot settings, and provide a corresponding analysis of these findings, we present the following findings for each task.

5.3.1 Self-Consistency Q&A results on Log Anomaly Detection

From the overall performance results, we can draw the following scientifically conclusions:

- Few-shot learning does not outperform zero-shot learning in log anomaly detection tasks, highlighting its limitations in this context. In the Self-Consistency Q&A test, which involves multiple inquiries to the LLM and taking the most frequent answer, few-shot learning did not significantly surpass zero-shot learning. This outcome

may be because the provided few-shot examples still do not sufficiently cover all patterns, thus failing to improve LLM consistency. LLMs in this setup tend to repeat examples rather than effectively learn new anomaly detection patterns from limited samples.

- The BaiChuan model shows a significant improvement in the Self-Consistency mode, indicating potential for more consistent responses, though its performance remains volatile. Compared to the Naive Q&A test, the BaiChuan model improved notably in the Self-Consistency Q&A test, suggesting a greater likelihood of generating consistent answers in repeated queries. However, it also shows considerable variability in responses across rounds, revealing a lack of stability in multi-turn interactions. Further optimization is needed to enhance the BaiChuan model's consistency and coherence in continuous query settings.
- The LLaMA2 series of models demonstrates poor performance and lack of stability in Self-Consistency Q&A test, suggesting the need for further improvements and optimizations. In multiple queries, the LLaMA2 models continue to produce low and inconsistent performance, indicating deficiencies in generating stable responses. This result may stem from limited generalization capabilities in handling complex tasks or a lack of optimization for log anomaly detection tasks. Enhancing the consistency of the LLaMA2 models in multi-turn Q&A may require architectural improvements or additional fine-tuning on relevant data to improve robustness in repeated queries.

5.3.2 Self-Consistency Q&A results on Log Fault Diagnosis

From the overall performance results, we find that the few-shot results are better than zero-shot results, similar to the Naive Q&A results. This indicates stable output in the log fault diagnosis task, with GPT-3.5 and GPT-4 showing far superior results. The Baichuan model performs poorly under both Self-Consistency and Naive Q&A, while other LLMs do not change much relative to the Naive Q&A results. The zero-shot and few-shot performance of the LLMs are examined for English and Chinese test sets by comparing the results of the Naive and Self-Consistency Q&A experiment. The following conclusions can be drawn from the results:

- For most LLMs, performance does not change much from Naive Q&A to Self-Consistency Q&A. In the anomaly detection task, the performance under few-shot conditions is inferior to zero-shot. Conversely, in the fault diagnosis task, the performance under few-shot conditions exceeds zero-shot scenarios.
- In these settings, Self-Consistency prompts relatively minor improvements to the LLM. In repeated questions, the LLM's answers were consistent.

5.4 RQ4: Performance on Inference Time and Average Number of Tokens

To investigate the reasoning efficiency of LLMs and whether they are in generating responses, we summarized the inference time for different LLMs and the average number of tokens output per log. The inference time and Average Number of Tokens

used for each task on the English dataset in the zero-shot case of the Naive Q&A are shown below.

5.4.1 Inference Time

Fig. 7 presents the inference time of 18 mainstream LLMs across four log analysis tasks, measured under the English dataset and zero-shot Naive Q&A setting. We first analyze the inference performance by task and model, and then discuss their potential in high-throughput scenarios.

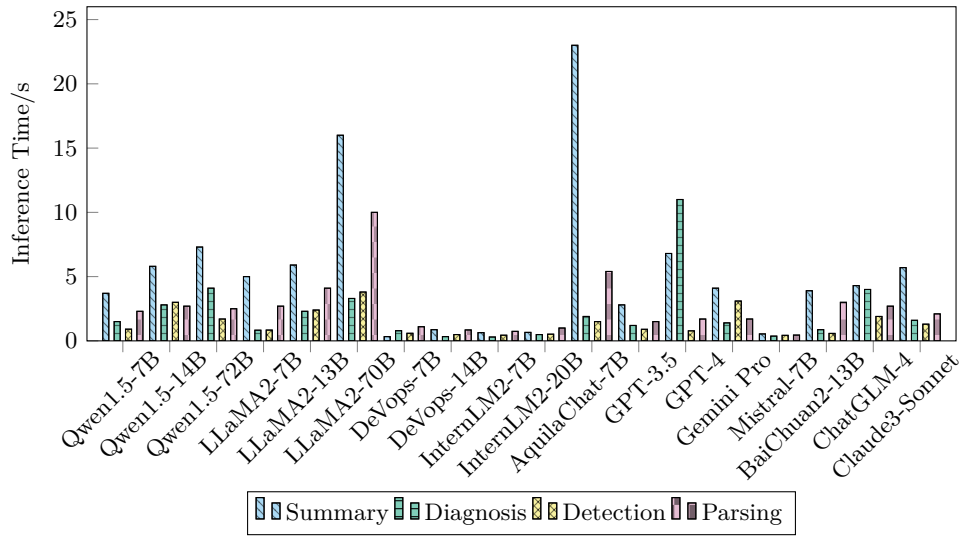

Fig. 7: The Inference Time in the Naive Q&A in log analysis

Task-wise Inference Time Comparison.

The log summarization task generally exhibits the longest inference time, with some models reaching 5–7 seconds. This is primarily due to the longer input length and the need for the model to integrate and rewrite information across multiple sentences. Log fault diagnosis and log parsing tasks show moderate inference time (mostly 1–3 seconds), indicating a relatively structured reasoning path and lower computational demand. Log anomaly detection, the only real-time task, achieves the shortest inference time. Lightweight models like DevOps-7B and InternLM2-7B maintain consistent latency between 0.4–0.7 seconds, demonstrating their suitability for real-time applications.

We also observe a clear correlation between model size and inference latency. 70B-scale models (e.g., LLaMA2-70B, Qwen1.5-72B) show significantly higher latency and are more appropriate for offline tasks, while 7B/14B models provide excellent responsiveness suitable for latency-sensitive deployments.

Analysis of Scalability under High Log Volumes.
Inference time directly affects a model's capacity to handle large-scale log streams. Based on our measurements, we further analyze the models' applicability in high-throughput industrial scenarios. For example, typical production systems generate approximately 100,000 logs/hour (28 logs/sec). Among the four tasks, only log anomaly detection requires real-time processing. DevOps-7B, with an average latency of 0.43s, can theoretically support over 2,000 logs/sec, exceeding real-world demands and ensuring both low latency and system stability. The remaining three tasks can be processed in offline batches, allowing for the use of larger models (e.g., Qwen1.5-72B) that trade latency for improved accuracy. A practical solution involves a two-stage architecture, Stage 1 (Light Filtering): Rule-based filters or lightweight LLMs remove 90% of normal logs. Stage 2 (LLM Analysis): The remaining 10% (2.8 logs/sec) are processed by more capable LLMs.

Furthermore, LogGPT [58] and LogPrompt [74] have demonstrated the ability to process log anomaly detection, further validating the scalability of LLM-based log analysis pipelines. In summary, inference time serves as a practical indicator not only for real-time responsiveness but also for guiding the architectural design of LLM-based solutions to meet high-throughput industrial requirements.

5.4.2 Average Number of Tokens

Fig. 8 shows the Average Number of Tokens of the four classes of tasks on the English data set with zero-shot setting for Naive Q&A.

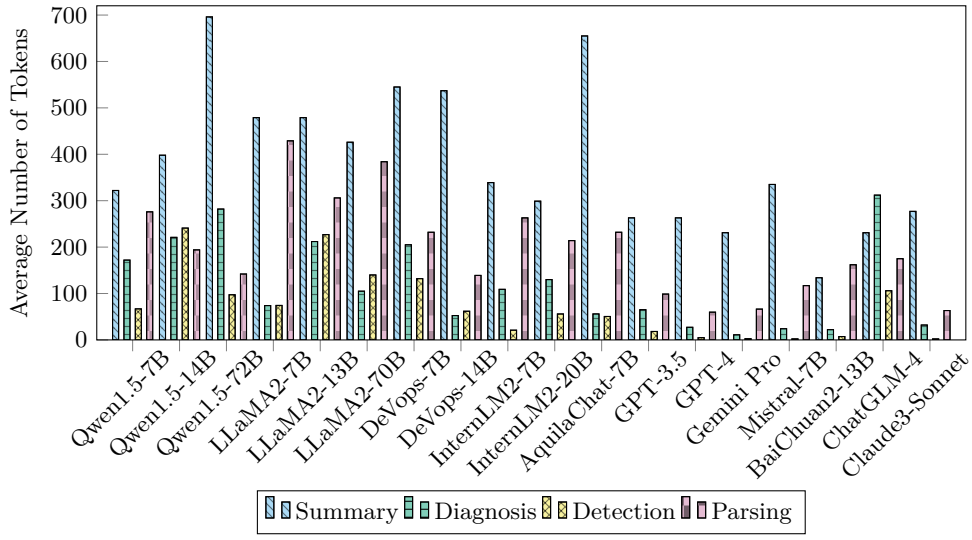

Fig. 8: The Average Number of Tokens in the Naive Q&A in log analysis

From the overall performance evaluation results, the log summary task outputs the highest average number of tokens among the four tasks. This phenomenon is

mainly determined by the nature of the task because the log summary task requires the LLM to generate a concise summary, which usually requires more tokens to accurately represent the main content of the log. However, our evaluation results show that ChatGLM-4, GPT, and Mistral models output a lower average number of tokens, indicating that their answers are more concise, without excessive redundant information, and their outputs are cleaner. Conversely, LLaMA and Qwen models output more tokens on average, meaning their answers contain more extraneous content. In practice, this can result in users spending more time and effort sifting useful information from responses, which reduces efficiency.

5.5 RQ5: Performance on Different parameters and Language

To provide the impact of different parameter sizes on models, this section conducts a comparative analysis of the performance of LLaMA-2 and Qwen-1.5, each evaluated with three different parameter sizes, offering insights into their adaptability and potential use cases.

Fig. 9 shows the accuracy of LLaMA-2 and Qwen-1.5 with different parameter sizes. We used a zero-shot Naive Q&A assessment on English prompts.

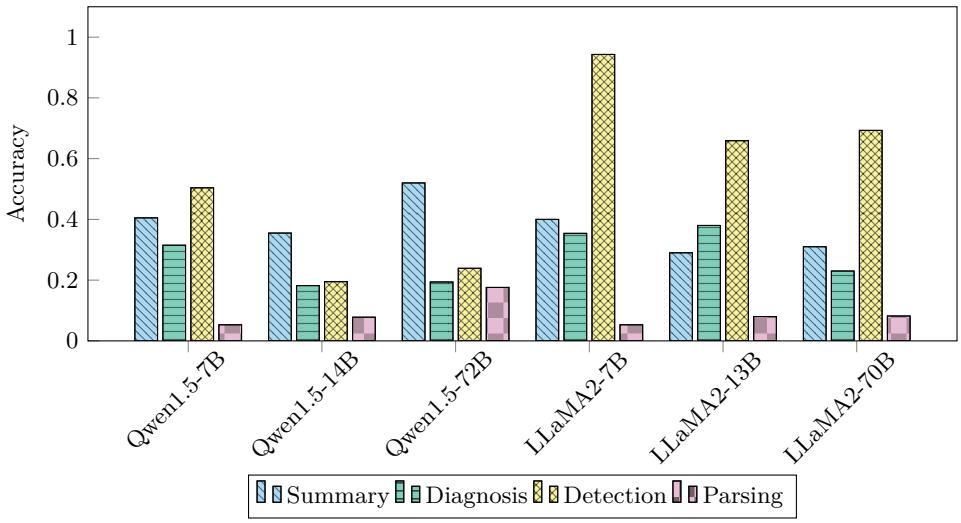

Fig. 9: The Accuracy of LLaMA-2 and Qwen-1.5 in zero-shot English Naive Q&A

From the comparison of results, most LLMs achieve better performance with a parameter size of 7B across the majority of tasks. This finding suggests that LLM size is not a determining factor for log analysis tasks. While an increase in the number of parameters generally means that the LLM can capture more features and patterns, a large number of parameters can also cause the LLM to be too complex to process log data quickly and accurately in real-world applications. Therefore, we can conclude that for log analysis tasks, choosing the right number of parameters is crucial, not simply "bigger is better." Future research should focus on how to optimize the size of

the LLM for a more efficient and cost-effective log analysis solution without sacrificing performance.

To provide the impact of different language on models, this section conducts a comparative analysis , as illustrated in Fig. 10, reveals notable differences. LLMs such as LLaMA series, GPT-4, ChatGLM4, and Claude3-Sonnet excel in English tasks, while LLMs like Qwen and DevOps, trained with a substantial amount of Chinese data, outperform in Chinese tasks. This performance disparity is attributable to the linguistic distribution in the LLMs' pretraining datasets. Therefore, task-specific language requirements must guide LLM selection. For Chinese-focused applications, LLMs like Qwen and DevOps are recommended, whereas English-dominant tasks may benefit from the LLaMA series or GPT-4.This discussion outlines the specific performance of LLMs in different languages.

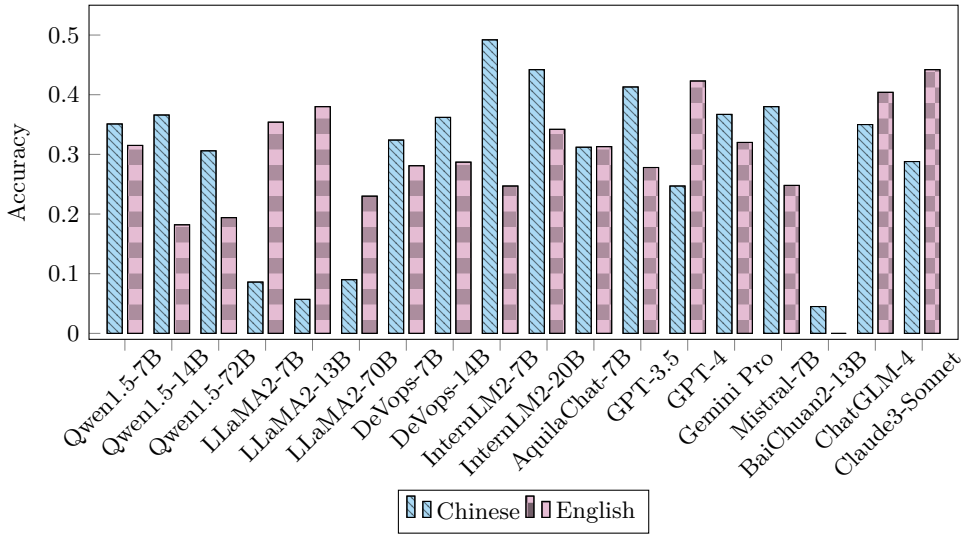

Fig. 10: The performance of LLMs under the "zero-shot" Naive Q&A in both Chinese and English test sets

This section provides a comprehensive performance evaluation of several LLMs. Through comparative analysis of these LLMs, we find significant differences in their performance on log analysis tasks. These differences may be due to differences in LLM design philosophy, training strategies, and LLM architecture. For example, some LLMs may perform better in parsing, while others may show greater efficiency in generating summaries or detecting anomalies. Additionally, the number of parameters and training objectives of the LLM are also important factors affecting its performance in the log analysis task. Our evaluation highlights the need to consider these factors when selecting and customizing a log analysis LLM to ensure that the LLM effectively meets the needs of real-world applications.

6 Conclusion

In this study, we have addressed a significant gap in the field of log analysis, where a standardized and systematic evaluation framework for assessing LLMs across multiple tasks has been lacking. The heterogeneity in LLM architectures, varying parameter sizes, and diverse applicability in log analysis complicate the decision-making process for selecting the most suitable LLM. To overcome this challenge, we introduced *LogEval*, a unified and comprehensive benchmarking suite designed to rigorously evaluate the performance of different LLMs across key log analysis tasks, including log parsing, anomaly detection, fault diagnosis, and summarization.

LogEval provides a robust framework that facilitates consistent comparisons among LLMs. The benchmark is complemented by a real-time, dynamically updating platform, accessible at https://nkcs.iops.ai/LogEval/, which serves as a valuable resource for both researchers and practitioners in the domain. This platform enables users to stay up-to-date with the latest advancements in LLM technology and understand how different LLMs perform in practical log analysis scenarios. Our code repository is available at https://github.com/LinDuoming/LogEval. Our evaluation results have highlighted the strengths and limitations of various LLMs, underscoring the importance of task-specific LLM selection and the impact of zero-shot versus few-shot prompting techniques. *LogEval* not only offers a clear performance overview but also provides insights that can guide the design and deployment of LLM-based log analysis systems.

Declarations

Funding

Not applicable.

Ethical approval

Not applicable.

Informed consent

Not applicable.

Author Contributions

All authors contributed to the study conception, design, implementation, and manuscript preparation. All authors read and approved the final manuscript.

Data Availability Statement

To facilitate further work by other researchers in this area, all of our scripts and datasets are available online. The material and data from this study are available from the following URL: https://nkcs.iops.ai/LogEval/.

Conflict of Interest

The authors declare that they have no conflict of interest.

Clinical trial number

Not applicable.

References

[1] Cito, J., Leitner, P., Fritz, T., Gall, H.C.: The making of cloud applications: an empirical study on software development for the cloud. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2015, pp. 393–403. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2786805.2786826

[2] Li, Y., Jiang, Z.M.J., Li, H., Hassan, A.E., He, C., Huang, R., Zeng, Z., Wang, M., Chen, P.: Predicting node failures in an ultra-large-scale cloud computing platform: An aiops solution. ACM Trans. Softw. Eng. Methodol. **29**(2) (2020) https://doi.org/10.1145/3385187

[3] Zhang, X., Xu, Y., Qin, S., He, S., Qiao, B., Li, Z., Zhang, H., Li, X., Dang, Y., Lin, Q., Chintalapati, M., Rajmohan, S., Zhang, D.: Onion: identifying incident-indicating logs for cloud systems. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2021, pp. 1253–1263. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3468264.3473919

[4] Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., Kao, O.: Self-attentive classification-based anomaly detection in unstructured logs. In: 2020 IEEE International Conference on Data Mining (ICDM), pp. 1196–1201 (2020). https://doi.org/10.1109/ICDM50108.2020.00148

[5] Wang, J., Chu, G., Wang, J., Sun, H., Qi, Q., Wang, Y., Qi, J., Liao, J.: Logexpert: Log-based recommended resolutions generation using large language model, 42–46 (2024) https://doi.org/10.1145/3639476.3639773

[6] Zhong, A., Mo, D., Liu, G., Liu, J., Lu, Q., Zhou, Q., Wu, J., Li, Q., Wen, Q.: Logparser-llm: Advancing efficient log parsing with large language models. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. KDD '24, pp. 4559–4570. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3637528.3671810

[7] Locke, S., Li, H., Chen, T.-H.P., Shang, W., Liu, W.: Logassist: Assisting log analysis through log summarization. IEEE Transactions on Software Engineering **48**(9), 3227–3241 (2022) https://doi.org/10.1109/TSE.2021.3083715

[8] Ma, L., Yang, W., Xu, B., Jiang, S., Fei, B., Liang, J., Zhou, M., Xiao, Y.: Knowlog: Knowledge enhanced pre-trained language model for log understanding. In: ICSE, pp. 32–13213 (2024). https://doi.org/10.1145/3597503.3623304

[9] Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 102–111 (2016)

[10] He, S., He, P., Chen, Z., Yang, T., Su, Y., Lyu, M.R.: A survey on automated log analysis for reliability engineering. ACM Comput. Surveys **54**(6), 1–37 (2021)

[11] Meng, W., Liu, Y., Zaiter, F., Zhang, S., Chen, Y., Zhang, Y., Zhu, Y., Wang, E., Zhang, R., Tao, S., Yang, D., Zhou, R., Pei, D.: Logparse: Making log parsing adaptive through word classification. In: 2020 29th International Conference on Computer Communications and Networks (ICCCN), pp. 1–9 (2020). https://doi.org/10.1109/ICCCN49398.2020.9209681

[12] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and benchmarks for automated log parsing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 121–130 (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00021

[13] Liu, Y., Zhang, X., He, S., Zhang, H., Li, L., Kang, Y., Xu, Y., Ma, M., Lin, Q., Dang, Y., Rajmohan, S., Zhang, D.: Uniparser: A unified log parser for heterogeneous log data. In: Proceedings of the ACM Web Conference 2022. WWW '22, pp. 1893–1901. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3485447.3511993

[14] Coustié, O., Mothe, J., Teste, O., Baril, X.: Meting: A robust log parser based on frequent n-gram mining, pp. 84–88 (2020). https://doi.org/10.1109/ICWS49710.2020.00018

[15] Le, V.-H., Zhang, H.: Log parsing with prompt-based few-shot learning. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 2438–2449 (2023). https://doi.org/10.1109/ICSE48619.2023.00204

[16] Xiao, T., Quan, Z., Wang, Z.-J., Zhao, K., Liao, X.: Lpv: A log parser based on vectorization for offline and online log parsing. In: 2020 IEEE International Conference on Data Mining (ICDM), pp. 1346–1351 (2020). https://doi.org/10.1109/ICDM50108.2020.00175

[17] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and benchmarks for automated log parsing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 121–130 (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00021

[18] Wang, X., Zhang, X., Li, L., He, S., Zhang, H., Liu, Y., Zheng, L., Kang, Y.,

Lin, Q., Dang, Y., Rajmohan, S., Zhang, D.: Spine: a scalable log parser with feedback guidance. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2022, pp. 1198–1208. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3540250.3549176

[19] Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17, pp. 1285–1298. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134015

[20] Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., Zhou, R.: Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: International Joint Conference on Artificial Intelligence (2019)

[21] Guo, H., Yuan, S., Wu, X.: Logbert: Log anomaly detection via bert. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2021). https://doi.org/10.1109/IJCNN52387.2021.9534113

[22] Le, V.-H., Zhang, H.: Log-based anomaly detection with deep learning: how far are we? In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, pp. 1356–1367. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3510003.3510155

[23] Zhao, N., Wang, H., Li, Z., Peng, X., Wang, G., Pan, Z., Wu, Y., Feng, Z., Wen, X., Zhang, W., Sui, K., Pei, D.: An empirical investigation of practical log anomaly detection for online service systems. ESEC/FSE 2021. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3468264.3473933

[24] Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., Chen, J., He, X., Yao, R., Lou, J.-G., Chintalapati, M., Shen, F., Zhang, D.: 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. ESEC/FSE 2019, pp. 807–817. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338906.3338931

[25] Du, Q., Zhao, L., Xu, J., Han, Y., Zhang, S.: Log-Based Anomaly Detection with Multi-Head Scaled Dot-Product Attention Mechanism, pp. 335–347 (2021). https://doi.org/10.1007/978-3-030-86472-9_31

[26] Jia, T., Wu, Y., Hou, C., Li, Y.: Logflash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems. In: 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), pp.

80–90 (2021). https://doi.org/10.1109/ISSRE52982.2021.00021

[27] Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Liu, D., Xiang, Q., He, C.: Latent error prediction and fault localization for microservice applications by learning from system trace logs. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2019, pp. 683–694. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338906.3338961

[28] He, S., Lin, Q., Lou, J.-G., Zhang, H., Lyu, M.R., Zhang, D.: Identifying impactful service system problems via log analysis. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2018, pp. 60–70. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3236024.3236083

[29] Liu, Y., Yang, H., Zhao, P., Ma, M., Wen, C., Zhang, H., Luo, C., Lin, Q., Yi, C., Wang, J., Zhang, C., Wang, P., Dang, Y., Rajmohan, S., Zhang, D.: Multi-task hierarchical classification for disk failure prediction in online service systems. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. KDD '22, pp. 3438–3446. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3534678.3539176

[30] Ma, M., Liu, Y., Tong, Y., Li, H., Zhao, P., Xu, Y., Zhang, H., He, S., Wang, L., Dang, Y., Rajmohan, S., Lin, Q.: An empirical investigation of missing data handling in cloud node failure prediction. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2022, pp. 1453–1464. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3540250.3558946

[31] Luo, C., Zhao, P., Qiao, B., Wu, Y., Zhang, H., Wu, W., Lu, W., Dang, Y., Rajmohan, S., Lin, Q., Zhang, D.: Ntam: Neighborhood-temporal attention model for disk failure prediction in cloud platforms. In: Proceedings of the Web Conference 2021. WWW '21, pp. 1181–1191. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3442381.3449867

[32] Zhou, P., Wang, Y., Li, Z., Wang, X., Tyson, G., Xie, G.: Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning, pp. 1–10 (2020). https://doi.org/10.1109/IWQoS49365.2020.9212954

[33] Meng, W., Zaiter, F., Zhang, Y., Liu, Y., Zhang, S., Tao, S., Zhu, Y., Han, T., Zhao, Y., Wang, E., Zhang, Y., Pei, D.: Logsummary: Unstructured log summarization for software systems. IEEE Transactions on Network and Service Management **20**(3), 3803–3815 (2023) https://doi.org/10.1109/TNSM.2023.3236994

[34] Sui, Y., Zhang, Y., Sun, J., Xu, T., Zhang, S., Li, Z., Sun, Y., Guo, F., Shen, J., Zhang, Y., Pei, D., Yang, X., Yu, L.: Logkg: Log failure diagnosis through knowledge graph. IEEE Transactions on Services Computing **16**(5), 3493–3507 (2023) https://doi.org/10.1109/TSC.2023.3293890

[35] Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., Meng, D.: Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. CCS '19, pp. 1777–1794. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3363224

[36] Le, V.-H., Zhang, H.: Prelog: A pre-trained model for log analytics. Proc. ACM Manag. Data **2**(3) (2024) https://doi.org/10.1145/3654966

[37] He, M., Jia, T., Duan, C., Cai, H., Li, Y., Huang, G.: Llmelog: An approach for anomaly detection based on llm-enriched log events. In: 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE), pp. 132–143 (2024). IEEE

[38] Liu, Y., Tao, S., Meng, W., Wang, J., Ma, W., Chen, Y., Zhao, Y., Yang, H., Jiang, Y.: Interpretable online log analysis using large language models with prompt strategies. In: Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, pp. 35–46 (2024)

[39] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., al., S.B.: GPT-4 Technical Report (2024). https://arxiv.org/abs/2303.08774

[40] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)

[41] THUDM: Thudm/chatglm4. https://github.com/THUDM/ChatGLM4 (2024)

[42] Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., Hui, B., Ji, L., al., M.L.: Qwen Technical Report (2023). https://arxiv.org/abs/2309.16609

[43] Fawcett, T.: An introduction to roc analysis. Pattern recognition letters **27**(8), 861–874 (2006)

[44] Lin, C.-Y.: Rouge: A package for automatic evaluation of summaries. in text summarization branches out. association for computational linguistics. Association for Computational Linguistics, Barcelona, Spain (2004)

[45] Papineni, K., Roukos, S., Ward, T., Zhu, W.-J.: Bleu: a method for automatic

evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA (2002)

[46] Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al.: Holistic evaluation of language models. arXiv e-prints (2022)

[47] Srivastava, A., Rastogi, A., Rao, A., Shoeb, A.A.M., Abid, A., Fisch, A., Brown, A.R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al.: Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. arXiv e-prints (2022)

[48] Zhang, L., Cai, W., Liu, Z., Yang, Z., Dai, W., Liao, Y., Qin, Q., Li, Y., Liu, X., Liu, Z., al.: Fineval: A chinese financial domain knowledge evaluation benchmark for large language models. arXiv e-prints (2023)

[49] Singhal, K., Azizi, S., Tu, T., Mahdavi, S.S., Wei, J., Chung, H.W., Scales, N., Tanwani, A., Cole-Lewis, H., Pfohl, S., *et al.*: Large language models encode clinical knowledge. Nature **620**(7972), 172–180 (2023)

[50] Li, J., Wang, X., Wu, X., Zhang, Z., Xu, X., Fu, J., Tiwari, P., Wan, X., Wang., B.: Huatuo-26m,a large-scale chinese medical qa dataset. arXiv e-prints (2023)

[51] Miao, Y., Bai, Y., Li Chen, H.S. Dan Li, Wang, X., Luo, Z., Sun, D., Xu, X., Zhang, Q., Xiang, C., Li., X.: An empirical study of netops capability of pre-trained large language models. arXiv e-prints (2023)

[52] Liu, Y., Pei, C., Xu, L., Chen, B., Sun, M., Zhang, Z., Sun, Y., Zhang, S., Wang, K., Zhang, H., Li, J., Xie, G., Wen, X., Nie, X., Ma, M., Pei., D.: Opseval: A comprehensive it operations benchmark suite for large language models. arXiv e-prints (2023)

[53] Silva, A., Monperrus, M.: Repairbench: Leaderboard of frontier models for program repair. arXiv preprint arXiv:2409.18952 (2024)

[54] Jiang, Z., Liu, J., Chen, Z., Li, Y., Huang, J., Huo, Y., He, P., Gu, J., Lyu, M.R.: Lilac: Log parsing using llms with adaptive parsing cache. Proceedings of the ACM on Software Engineering **1**(FSE), 137–160 (2024)

[55] Xu, J., Yang, R., Huo, Y., Zhang, C., He, P.: Prompting for automatic log template extraction. arXiv preprint arXiv:2307.09950 (2023)

[56] Zhang, W., Cheng, X., Zhang, Y., Yang, J., Guo, H., Li, Z., Yin, X., Guan, X., Shi, X., Zheng, L., et al.: Eclipse: Semantic entropy-lcs for cross-lingual industrial log parsing. arXiv preprint arXiv:2405.13548 (2024)

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

[58] Qi, J., Huang, S., Luan, Z., Yang, S., Fung, C., Yang, H., Qian, D., Shang, J., Xiao, Z., Wu, Z.: Loggpt: Exploring chatgpt for log-based anomaly detection. In: 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), pp. 273–280 (2023). IEEE

[59] Shan, S., Huo, Y., Su, Y., Li, Y., Li, D., Zheng, Z.: Face it yourselves: An llm-based two-stage strategy to localize configuration errors via logs. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 13–25 (2024)

[60] Xu, J., Cui, Z., Zhao, Y., Zhang, X., He, S., He, P., Li, L., Kang, Y., Lin, Q., Dang, Y., *et al.*: Unilog: Automatic logging via llm and in-context learning. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, pp. 1–12 (2024)

[61] Karlsen, E., Luo, X., Zincir-Heywood, N., Heywood, M.: Benchmarking large language models for log analysis, security, and interpretation. Journal of Network and Systems Management **32**(3), 59 (2024)

[62] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. (2019). https://api.semanticscholar.org/CorpusID:160025533

[63] Block, J., Chen, Y.-P., Budharapu, A., Anthony, L., Dorr, B.: Summary cycles: Exploring the impact of prompt engineering on large language models' interaction with interaction log information. In: Proceedings of the 4th Workshop on Evaluation and Comparison of NLP Systems, pp. 85–99 (2023)

[64] Rosado, T., Bernardino, J.: An overview of openstack architecture. In: Proceedings of the 18th International Database Engineering & Applications Symposium. IDEAS '14, pp. 366–367. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2628194.2628195

[65] Jiang, Z., Liu, J., Huang, J., Li, Y., Huo, Y., Gu, J., Chen, Z., Zhu, J., Lyu, M.R.: A large-scale evaluation for log parsing techniques: How far are we? In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 223–234 (2024)

[66] OpenAI: Introducing ChatGPT (2022). https://openai.com/blog/chatgpt

[67] Anthropic: (2023). https://claude.ai/

[68] Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., et al.: Gemini: A family of highly capable multimodal models. arXiv preprint arXiv:2312.11805 (2023)

[69] Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L.R., Lachaux, M.-A., Stock, P., Scao, T.L., Lavril, T., Wang, T., Lacroix, T., Sayed, W.E.: Mistral 7B (2023)

[70] InternLM: InternLM: A Multilingual Language Model with Progressively Enhanced Capabilities (2023). https://github.com/InternLM/InternLM

[71] CodeFuse: (2023). https://github.com/codefuse-ai/CodeFuse-DevOps-Model/

[72] BAAI: (2023). https://github.com/FlagAI-Open/Aquila2

[73] Yang, A., Xiao, B., Wang, B., Zhang, B., Bian, C., Yin, C., Lv, C., Pan, D., Wang, D., Yan, D., Yang, F., Deng, F., Wang, F., al., F.L.: Baichuan 2: Open Large-scale Language Models (2023). https://arxiv.org/abs/2309.10305

[74] Liu, Y., Tao, S., Meng, W., Yao, F., Zhao, X., Yang, H.: Logprompt: Prompt engineering towards zero-shot and interpretable log analysis. In: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, pp. 364–365 (2024)