LogRules: Enhancing Log Analysis Capability of Large Language Models through Rules

Xin Huang[*] Beijing Institute of Technology huangxin@bit.edu.cn

Ting Zhang[*] Peking University zhangting2019@pku.edu.cn

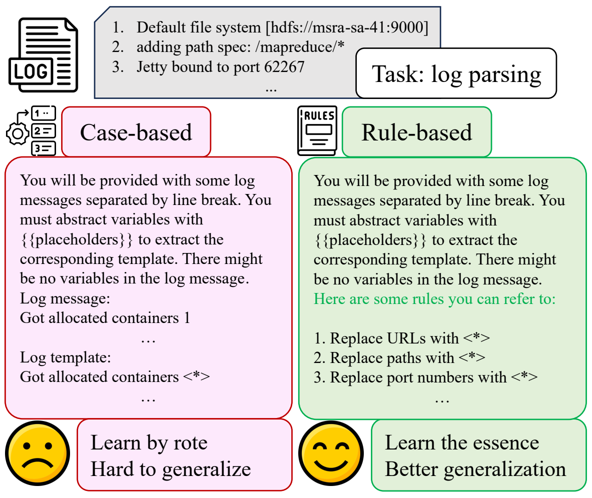
Wen Zhao Peking University zhaowen@pku.edu.cn

Abstract

Currently, large language models (LLMs) have achieved impressive performance in natural language processing tasks. However, LLMs still exhibit many hallucinations when analyzing system logs, which is due to the implicit knowledge and rules in logs that LLMs cannot capture. Based on this, we propose LogRules, a lightweight log analysis framework that generates and utilizes rules through LLMs. LogRules consists of three stages: an induction stage, an alignment stage, and a reasoning stage. Firstly, in the induction stage, an strong LLM (e.g., GPT-4o-mini) is tasked with generating a series of rules related to logs, which are then validated on the training set. When the rules are confirmed to produce correct reasoning results, they are added to a rule repository. Secondly, considering that the LLMs with small size (≈8B parameters) still face challenges in utilizing rules, we design an alignment method based on rule-case contrastive preference optimization (CPO) to effectively enhance the rule reasoning capabilities of these LLMs. Finally, in the reasoning stage, the LLM constructs prompt using the rule repository and performs log analysis on the test set. Experiments show that LogRules outperforms LLM-based methods in log parsing and anomaly detection tasks, and achieves better performance compared to case-based methods.

1 Introduction

System logs provide critical runtime information, helping developers, operators and maintainers understand system behavior and debug issues (Satpathi et al., 2019; Le and Zhang, 2021). However, as systems grow more complex, analyzing vast amounts of log data has become a significant challenge (Mi et al., 2013; Oliner et al., 2012). Recent advances in large language models (LLMs) for natural language processing have inspired efforts to apply LLMs to system log analysis, addressing tasks such as log parsing (Le and Zhang, 2023a; Xu et al., 2024b; Zhong et al., 2024) and log-based anomaly detection (Liu et al., 2024; Qi et al., 2023; Guo et al., 2024).



1. Default file system [hdfs://msra-sa-41:9000]
2. adding path spec: /mapreduce/*
3. Jetty bound to port 62267
...

You will be provided with some log messages separated by line break. You must abstract variables with {{placeholders}} to extract the corresponding template. There might be no variables in the log message.
Log message:
Got allocated containers 1
...
Log template:
Got allocated containers <*>
...

You will be provided with some log messages separated by line break. You must abstract variables with {{placeholders}} to extract the corresponding template. There might be no variables in the log message.
Here are some rules you can refer to:

1. Replace URLs with <*>
2. Replace paths with <*>
3. Replace port numbers with <*>
...

Learn by rote
Hard to generalize

Learn the essence
Better generalization

Figure 1: Comparison between case-based and rule-based reasoning.

However, the cost of LLMs is also a significant issue. In real-world scenarios, systems generate large amounts of log data. For example, Mi et al. (Mi et al., 2013) reported that the Alibaba cloud system generates approximately 30-50 GB (about 100-200 million lines) of trace logs per hour. On the one hand, due to the high inference cost of LLMs, invoking an LLM for each log message is impractical. On the other hand, using smaller LLMs is more cost-effective than using large-scale LLMs. For instance, if a system prompt contains 200 tokens and a log message has average 50 tokens, a query to GPT-3.5-turbo-0613 that generates a 200-token response would cost approximately \$0.000775 per log. This calculation is based on the GPT-3.5-turbo-0613 API pricing of \$1.50 per million input tokens and \$2.00 per million output

[*]Equal contribution.

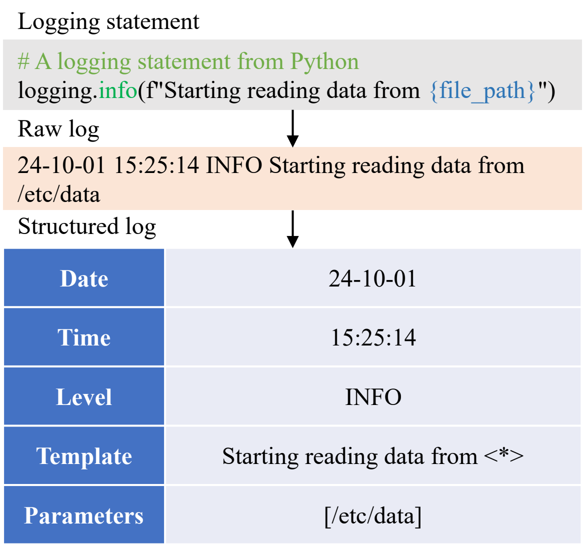
tokens [1]. In contrast, using a smaller LLM with around 8B parameters would cost only $0.00009 per log, given a rate of $0.20 per million tokens for both input and output [2]. This represents an approximate cost saving of 88.4% compared to using a larger LLM. Unfortunately, most current LLM-based log analysis methods rely on large LLMs like ChatGPT (OpenAI, 2023), and their performance on smaller LLMs remains suboptimal.

Besides, LLMs still face hallucination issues when analyzing logs, likely due to the implicit knowledge and rules in logs that LLMs struggle to capture. Hallucination occurs when LLMs produce outputs that seem reasonable but contradict real-world knowledge (Ji et al., 2023; Zhang et al., 2024; McKenna et al., 2023; Zhou et al., 2023). One method to enhance LLM reasoning and reduce hallucinations is prompt engineering, where examples are provided to clarify the objectives and approaches (Wei et al., 2022; Kojima et al., 2022; Khot et al., 2023). This is known as case-based prompting. While it helps reduce hallucinations, its ability to generalize is limited, especially as system logs evolve. As systems are updated, developers may modify source code and log statements, changing the structure of log data (Xu et al., 2009; Kabinna et al., 2018; Zhang et al., 2023). Case-based prompting fails to capture the underlying rules within logs, often leading LLMs to rely on rote memorization, which hampers their ability to generalize effectively (Zhu et al., 2023b).

To address aforementioned issues, we propose a lightweight log analysis framework called LogRules, which leverages rule-based prompting to improve the reasoning capabilities of small-scale LLMs. Compared to case-based prompting, rule-based prompting helps LLMs better understand the essence of tasks and knowledge, leading to improved generalization, as shown in Figure 1.

LogRules enhances the reasoning and generalization abilities of LLMs by building a rule repository specifically for logs, improving performance in log analysis tasks. LogRules operates in three phases: induction, alignment, and deduction. In the induction phase, the LLM generates a set of log-related rules, which are then validated on the training set. Once verified to produce accurate reasoning outcomes, these rules are added to the rule repository. In the alignment phase, the rule utilization ability

[1] `https://openai.com/api/pricing`
[2] `https://www.together.ai/pricing`



```
# A logging statement from Python
logging.info(f"Starting reading data from {file_path}")
```

```
24-10-01 15:25:14 INFO Starting reading data from
/etc/data
```

Date	24-10-01
Time	15:25:14
Level	INFO
Template	Starting reading data from <*>
Parameters	[/etc/data]

Figure 2: An example of log parsing.

of the LLM is improved through rule-case contrastive preference optimization (CPO) (Xu et al., 2024a). In the deduction phase, the LLM uses the rule repository to construct prompts for log analysis on the test set.

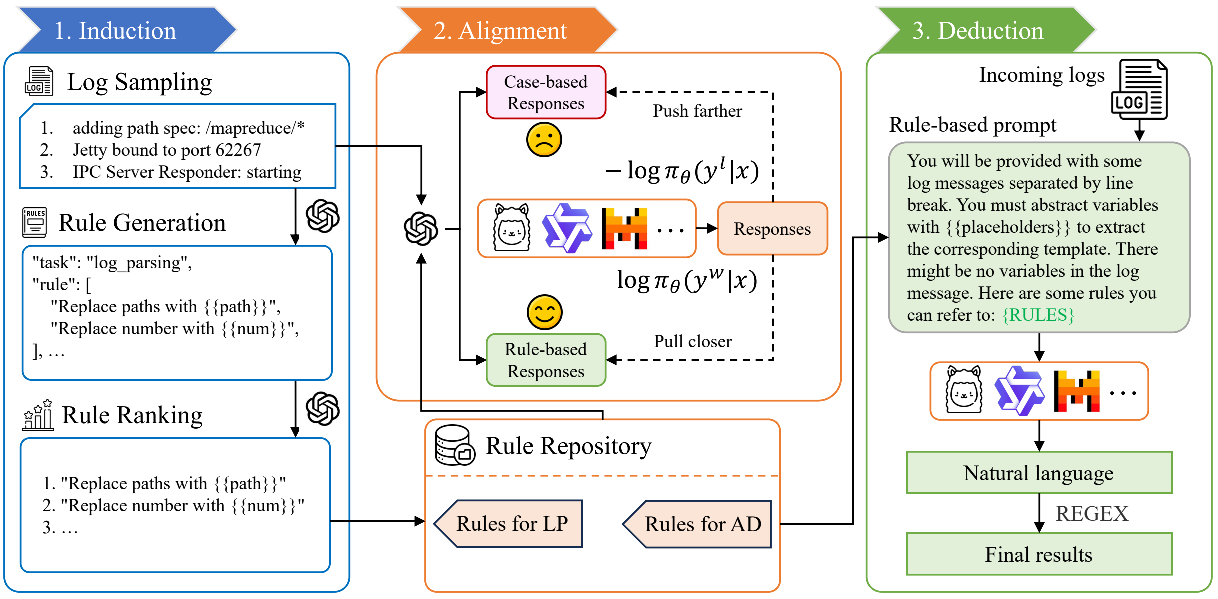
We conduct experiments on several open-source log datasets for log parsing and anomaly detection, demonstrating that LogRules improves reasoning ability and performance in these tasks by leveraging rules. Additionally, compared to case-based methods, rule-based prompting shows stronger robustness.

2 Related Work

2.1 Log Parsing

The goal of log parsing is to transform raw logs into log templates (the constant parts written in log statements) and extract log parameters (the dynamic parts). This structured format facilitates downstream tasks, such as log-based anomaly detection.

Log parsing enables the identification and separation of static text from variables in raw logs (Zhu et al., 2019). Typically, logs contain many variables. First, common variables such as timestamps, log levels, and log content are separated using regular expressions. Then, the log content is further structured into log templates by replacing the variables (e.g., file paths, URLs, and IP addresses) with placeholders like {{variable}} or <*>. For example, the log statement *logging.info(f"Starting reading data from {file_path}")* can generate a series of log messages with different file paths, such



```
1.   adding path spec: /mapreduce/*
2.   Jetty bound to port 62267
3.   IPC Server Responder: starting
```

```
"task": "log_parsing",
"rule": [
    "Replace paths with {{path}}",
    "Replace number with {{num}}",
], …
```

```
1. "Replace paths with {{path}}"
2. "Replace number with {{num}}"
3. …
```

You will be provided with some log messages separated by line break. You must abstract variables with {{placeholders}} to extract the corresponding template. There might be no variables in the log message. Here are some rules you can refer to: {RULES}

Figure 3: The framework of LogRules. The figure shows the process of log parsing, while log-based anomaly detection also follows the same process, with only some differences in the rule-based prompt.

as *Starting reading data from /etc/data/*. In this case, the log template is *Starting reading data from* <*>, and the variable is the data path */etc/data/*. An example of log parsing is shown in Figure 2.

Since source code accessibility is often restricted, many data-driven log parsers have been proposed, including unsupervised (Du and Li, 2016; He et al., 2017; Dai et al., 2022) and supervised parsers (Huo et al., 2023; Le and Zhang, 2023b; Liu et al., 2022). Unsupervised parsers extract patterns using specially designed features or domain-specific heuristic methods, relying on extensive pre-defined domain knowledge and rules (e.g., delimiters and regular expressions). However, manually constructing a large number of rules is clearly impractical. Furthermore, as log data evolves, these pre-defined rules may become outdated, leading to decreased performance. On the other hand, supervised parsers typically require model training on target log samples to learn features, which can result in inadequate generalization and suboptimal performance on diverse log datasets (Xu et al., 2024b).

We aim to address the limitations of both types of parsers using LLMs. First, LLMs can continuously generate and update rules based on historical logs, adapting to log evolution. Second, instead of requiring extensive data for training, LLMs use rule-based alignment, which enhances the model's generalization capability.

2.2 Log-based Anomaly Detection

Log-based anomaly detection aims to identify whether a system failure or anomaly has occurred, which may result from hardware or software issues, such as a kernel crash. In recent years, several deep learning-based methods have been proposed, including supervised approaches like LogRobust (Zhang et al., 2019) and NeuralLog (Le and Zhang, 2021) , and self-supervised methods such as DeepLog (Du et al., 2017) and LogAnomaly (Meng et al., 2019).

To improve the interpretability of anomaly detection, recent work has explored using LLMs for this task. Qi et al. proposed LogGPT (Qi et al., 2023), and Liu et al. introduced LogPrompt (Liu et al., 2024), both utilizing ChatGPT for log-based anomaly detection. Guo et al. developed OWL (Guo et al., 2024), which constructed an instruction dataset for IT-related tasks and fine-tuned LLMs for various IT-related tasks, including log-based anomaly detection.

However, despite improving interpretability, many LLM-based anomaly detection models struggle with precision, even when provided with numerous detection examples in the prompts. This is largely due to hallucinations generated by LLMs during inference, which increase false positives. Therefore, LLMs must understand the rules of anomaly detection beyond just examples to mitigate these hallucinations.

2.3 Enhancing LLM's Reasoning Abilities

Many real-world problems require inductive reasoning to derive correct answers. Typically, the problem statement provides the necessary facts within the context but does not explicitly state the rules. Humans often make hypotheses, test them based on observations, and refine them accordingly, eventually deriving implicit rules (Fränken et al., 2022; Qiu et al., 2024). Furthermore, humans can obtain knowledge by learning from rules in a different context. In other words, given detailed rules and a few examples, humans can generalize effectively.

However, LLMs still lag behind humans in reasoning ability. Currently, LLMs primarily learn through examples, implicitly acquiring internal rules from supervised datasets (Hu et al., 2024). While pre-trained LLMs can retrieve some common-sense knowledge from their parameters (Petroni et al., 2019), this implicit process often leads to unexpected hallucinations (Hu et al., 2024). Such hallucinations are particularly common with problems that are either infrequently addressed in the pre-training data (Roberts et al., 2020) or conflict with typical real-world problem formulations (Longpre et al., 2021; Wu et al., 2024).

Some works address the hallucinations within LLMs by having them propose hypotheses and apply inductive reasoning. For instance, iterative hypothesis refinement (Qiu et al., 2024) explores the potential of LLMs for inductive reasoning. Hypothesis search (Wang et al., 2024) tackles inductive reasoning tasks by generating and testing hypotheses in both natural language and code. When a hypothesis is validated for a specific task, it is treated as a rule. We aim for LLMs to use the correct rules they generate to perform stronger reasoning, leading to better performance in log analysis tasks.

3 LogRules

To reduce hallucinations in LLMs during log analysis tasks, we propose LogRules, a log analysis framework that generates and applies rules using LLMs. LogRules consists of three stages: induction, alignment, and deduction. In the induction stage, we use a powerful LLM (such as OpenAI GPT-4o-mini (OpenAI, 2024)) to generate a series of log-related rules, which are then validated on the training set. Once a rule is verified for accurate reasoning, it is added to the rule repository. In the alignment stage, we also use the powerful LLM to perform reasoning on the training logs using both rule-based and case-based prompting. The results from these reasoning processes are used as chosen and rejected alignment samples, and we fine-tune a smaller LLM (such as LLaMA-3-8B-Instruct (Dubey et al., 2024) and Qwen2.5-7B-Instruct (Qwen, 2024)) through contrastive preference optimization (CPO) (Xu et al., 2024a). In the reasoning stage, the smaller LLM leverages the rule repository to construct prompts for log analysis on the test set. The overall framework of LogRules is shown in Figure 3.

3.1 Induction Stage

To uncover explicit knowledge for reasoning, we use GPT-4o-mini to induce rules. The induction stage consists of three steps: log sampling, rule generation, and rule ranking.

3.1.1 Log Sampling

Since logs are vast and large in volume, generating rules from a massive amount of logs would be more reliable, but using GPT for large-scale reasoning is costly. Therefore, we sample logs to create a dataset $D_T = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i and y_i represent logs and labels respectively, and T denotes the specific log analysis task (e.g., log parsing or anomaly detection). The dataset D_T is then split into a training set D_T^{train} and a test set D_T^{test}, such that $D_T = D_T^{\text{train}} \cup D_T^{\text{test}}$.

3.1.2 Rule Generation

We prompt GPT-4o-mini to generate rules relevant to the log analysis task based on the training set. Rule reasoning involves a function (i.e., GPT and prompts) $f : X \cup Y \rightarrow V$ which maps logs and labels to rules $\{r_i\}_{i=1}^{M}$, where M is the number of rules, $r_i = (v_{i,1}, v_{i,2}, ..., v_{i,l})$, $v_{i,j} \in V (1 \leq j \leq l)$ represents the j-th token of the i-th rule, and l is the number of r_i's tokens. Rule r_i can take various forms, typically expressed in natural language, where V represents the vocabulary space of GPT-4o-mini. Our goal is to find a function f that best describes the rules governing D_T with only the training set D_T^{train}. A good rule should balance accuracy and coverage, meaning it must have enough expressive power to capture the rules of D_T^{train} while generalizing to D_T^{test}.

3.1.3 Rule Ranking

Once the rules are generated, they need to be prioritized so the LLMs can understand their relative importance and apply higher-priority rules during reasoning. We embed all the rules into the prompt and use GPT-4o-mini to perform reasoning on the training set D_T^{train}, asking it to indicate which rules are used. We then tally the usage of each rule and prioritize them based on the frequency of use, from highest to lowest.

3.2 Alignment Stage

We found that powerful LLMs (such as the GPT series) perform well in reasoning for log tasks using rules. However, smaller LLMs ($\approx$ 8B parameters) struggle to effectively utilize these rules. Specifically, smaller LLMs often lack the rule comprehension necessary to solve log tasks correctly or fail to follow the rules properly. To enhance the rule-based reasoning capabilities of smaller LLMs, we propose a rule-based alignment method.

First, we construct case-based prompting and rule-based prompting, with log training samples $x_i \in D_T^{\text{train}}$ as input. This generates outputs y_i^{case} and y_i^{rule}, respectively. Next, we select samples where the outputs generated by case-based prompting do not match the ground true labels as rejected samples, i.e., $y_i^l = \mathbb{1}(y_i^{\text{case}} \neq y_i)$. At the same time, samples where the outputs generated by rule-based prompting match the ground true labels as chosen samples, i.e., $y_i^w = \mathbb{1}(y_i^{\text{rule}} = y_i)$. This expanded the training set to $D_T^{\text{train}} = \{(x_i, y_i, y_i^l, y_i^w)\}_{i=1}^{N_{\text{train}}}$. Finally, we use contrastive preference optimization (CPO) (Xu et al., 2024a) to fine-tune the smaller LLM. Compared to direct preference optimization (DPO) (Rafailov et al., 2023), CPO does not rely on a reference model, optimizing both memory usage and speed. CPO contains two terms: a preferred term and a behavior cloning (BC) regularizer. The computation of the preferred term is as follows:

$$\mathcal{L}_{\text{P}}(\pi_\theta) = -\mathbb{E}_{(x_i, y_i^l, y_i^w)\sim D_T^{\text{train}}}[\log \sigma(\beta \log \pi_\theta(y_i^w|x_i) - \beta \log \pi_\theta(y_i^l|x_i))], \tag{1}$$

where π_θ represents the policy (LLM) with parameters θ, σ is the sigmoid function, and β is is a hyperparameter. Additionally, CPO incorporates a BC regularizer to ensure that π_θ does not deviate from the chosen data distribution, i.e.,

$$\mathcal{L}_{\text{BC}}(\pi_\theta) = -\mathbb{E}_{(x_i, y_i^l, y_i^w)\sim D_T^{\text{train}}}[\log \pi_\theta(y_i^w|x_i)]. \tag{2}$$

Finally, the total loss is computed as

$$\mathcal{L}(\pi_\theta) = \mathcal{L}_{\text{P}}(\pi_\theta) + \alpha \mathcal{L}_{\text{BC}}(\pi_\theta), \tag{3}$$

where α is a hyperparameter that controls the strength of the BC regularizer.

3.3 Deduction Stage

In the rule-based reasoning stage, the aligned LLM uses a rule-based prompt to analyze and reason about incoming logs. These prompts include the specific task (e.g., log parsing or log-based anomaly detection) and a brief task description written by humans based on the task type. Additionally, the set of rules generated during the induction stage is embedded into the rule-based prompt, guiding the LLM to perform reasoning based on these rules. The LLM is instructed to generate output, and the final answers are extracted from the output using regular expressions.

4 Experiments

4.1 Experimental Settings

4.1.1 Datasets

We use several real-world log datasets from the open-source Loghub (Zhu et al., 2023a) repository. These logs were generated by distributed systems, supercomputers, operating systems, standalone software, et al. The statistical information of the datasets can be found in Appendix A.

For log parsing, we evaluate LogRules on 16 datasets, each containing 2,000 logs. To achieve low-cost induction and alignment, we construct the training set using only a single log from each of 10 datasets, while the remaining 6 datasets are used to evaluate the generalization capability of LogRules.

For log-based anomaly detection, we select the labeled datasets BGL (Oliner and Stearley, 2007) and Spirit (Stearley and Oliner, 2008), each containing 10,000 logs. The datasets are split into training and test sets in a 4:1 ratio, ensuring no overlap between the two.

4.1.2 Evaluation Metrics

We select widely used metrics for evaluation. For log parsing, we use grouping accuracy (GA) (Zhu et al., 2019), parsing accuracy (PA), normalized edit distance (ED) (Marzal and Vidal, 1993), F1 score of grouping accuracy (FGA) (Jiang et al., 2023), and F1 score of template accuracy (FTA) (Dai et al., 2022). For log-based anomaly detection, we evaluate performance using precision (P), recall (R), and F1 score (F1). The details of these metrics can be found in Appendix B.

4.1.3 Hyperparameter Settings

The hyperparameter settings for LLM generation are as follows: temperature is set to 0.0, top_p and top_k are both set to 1, repetition_penalty is set to 1, and max_new_tokens is set to 2048. The hyperparameter settings for alignment are shown in Appendix D.

4.2 Comparison with Baselines

4.2.1 Log Parsing

We select 4 open-source log parsers and 1 closed-source LLM for baseline comparison: Brain (Yu et al., 2023), DivLog (Xu et al., 2024b), LILAC (Jiang et al., 2024), LogBatcher (Xiao et al., 2024), and GPT-4-turbo. Among these, Brain is a syntax-based parser, while DivLog, LILAC, and LogBatcher leverage OpenAI GPT for log parsing. To reduce inference costs while maintaining strong parsing performance, we focus on smaller LLMs, and replace the LLM of LogBatcher from GPT to a smaller LLM, LLaMA-3-8B-Instruct. LogRules follows the same process as LogBatcher but uses a smaller LLM and employs rule-based prompting for inference. Detailed descriptions of these baseline parsers are provided in Appendix C.1.

The experimental results of the baseline methods and LogRules across 16 datasets are shown in Table 1, and additional results for GPT-4o and GPT-4o-mini are shown in Appendix E. LILAC and GPT-4-turbo demonstrate excellent performance, while LogBatcher, after replacing GPT with the smaller LLM, shows a notable drop in performance, especially in PA and ED metrics. Our proposed LogRules performs only second to LILAC and GPT-4-turbo, both of which leverage powerful GPT models. In contrast, LogRules utilizes the Qwen2.5-7B-Instruct model with only 7 billion parameters. Notably, LogRules excels on the Windows and Linux datasets, where most parsers struggle.

To evaluate the robustness of the baseline methods and LogRules, we present the boxplot of the results in Figure 4. It shows that for the GA metric, LogRules achieves the third narrowest interquartile range (IQR) after Brain and GPT-4-turbo, and has only one outlier. In terms of PA, LogRules has higher upper and lower quartiles than LogBatcher, though the median is the same, but it is less robust compared with GPT-based parsers. For ED, while LogRules has a wider IQR than LILAC and LogBatcher, it has zero outliers, whereas both LILAC and LogBatcher have three outliers. Overall, LogRules demonstrates strong robustness, making it suitable for a wide range of log datasets.

4.2.2 Log-based Anomaly Detection

We select 8 log-based anomaly detection methods as baselines for comparison, including the unsupervised deep learning methods DeepLog (Du et al., 2017), LogAnomaly (Meng et al., 2019), the LLM-based methods LogPrompt (Liu et al., 2024), LogGPT (Qi et al., 2023), OWL (Guo et al., 2024), and 3 OpenAI GPT series models. Details of these methods can be found in Appendix C.2, with the experimental results shown in Table 2.

As seen in Table 2, LLM-based anomaly detection methods still lag slightly behind unsupervised deep learning approaches. However, our LogRules framework, by incorporating rule-based prompting, achieves the best performance. Compared to the LLM-based method OWL, LogRules improves F1 scores by 36.2% and 29.8% on the BGL and Spirit datasets, respectively.

4.3 Rule-based vs. Case-based Reasoning

4.3.1 Log Parsing

To compare the impact of case-based and rule-based prompting on log parsing, we evaluate LogRules with these tow strategies on 16 datasets, and calculate the average GA, FGA, and FTA metrics. For case-based prompting, we conduct experiments using 1-shot, 2-shot, and 4-shot configurations for LLaMA-3-8B-Instruct, with the results shown in Figure 5.

As seen in Figure 5, rule-based prompting outperforms case-based prompting across all metrics, demonstrating its superiority. Additionally, we observe that the 2-shot case-based prompting performs better than the 1-shot and 4-shot configurations, with FGA performing the worst in the 4-shot setup. This indicates that case-based prompting is highly sensitive to the selection of examples, leading to instability. In contrast, while rule-based prompting also depends on rule selection, the rules are validated and better capture the essence, resulting in greater stability.

4.3.2 Log-based Anomaly Detection

To assess the impact of case-based versus rule-based prompting on log-based anomaly detection, we evaluate LogRules with these tow strategies on the BGL and Spirit datasets, calculating precision, recall, and F1 scores. Similarly, for case-based prompting, we conduct experiments using 1-shot, 2-shot, and 4-shot configurations for LLaMA-3-8B-Instruct, with the results shown in Table 3.

Table 1: Comparison with state-of-the-art log parsers (%). The **bold** and underlined fonts indicate the best and second best results among methods, respectively.

	Brain			DivLog			LILAC			LogBatcher			GPT-4-turbo			LogRules		
	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑
HDFS	99.8	95.9	99.7	93.0	99.6	99.9	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	99.8	99.8	**100**
Hadoop	95.0	15.8	75.1	68.3	74.4	91.5	**99.1**	**95.8**	**98.6**	96.2	78.1	93.1	99.0	88.6	95.2	98.6	55.2	90.1
Spark	99.8	37.6	95.0	73.8	96.0	98.3	**99.9**	**98.3**	99.8	99.7	96.8	98.3	99.8	97.2	**99.9**	81.3	92.9	97.7
Zookeeper	98.9	77.9	98.7	95.5	97.9	99.8	**100**	98.7	**99.9**	98.6	54.3	85.1	99.5	**98.8**	99.5	98.9	98.7	99.6
BGL	**99.6**	42.6	89.1	73.6	95.0	**99.0**	98.3	**97.2**	98.9	86.6	89.1	96.1	96.7	94.1	98.9	98.8	96.2	98.9
HPC	94.5	66.0	97.3	93.5	98.0	99.7	**97.0**	**99.4**	**99.9**	95.1	74.7	94.6	95.3	84.3	99.5	95.0	94.4	99.4
Thunderbird	97.1	6.0	93.2	23.4	87.9	97.8	**98.4**	91.3	98.3	97.8	85.0	96.4	91.4	85.4	95.3	96.2	**92.4**	**98.8**
Windows	**99.7**	46.3	**97.6**	71.0	71.5	90.3	69.6	68.5	89.7	69.6	29.6	73.0	96.6	**85.9**	96.2	99.2	**85.9**	94.7
Linux	35.8	17.6	77.0	48.4	62.0	93.5	29.8	42.2	92.6	75.4	70.7	94.7	**89.8**	**87.6**	**99.2**	87.5	86.2	98.0
Android	96.0	25.3	92.4	73.7	67.7	95.2	95.3	62.7	92.3	94.4	**79.6**	95.8	**97.1**	78.7	**97.3**	92.4	70.7	95.2
HealthApp	**100**	26.1	87.1	87.6	98.4	99.7	99.8	**98.8**	**99.8**	90.0	86.9	94.2	92.0	91.4	98.1	99.6	98.6	99.5
Apache	**100**	98.4	99.6	98.4	98.5	99.7	**100**	**100**	**100**	**100**	73.1	94.1	98.6	97.8	99.6	**100**	99.4	99.9
Proxifier	52.7	70.4	94.5	53.1	99.3	99.9	**100**	99.5	99.9	52.2	52.2	92.5	85.1	90.6	99.7	**100**	**100**	**100**
OpenSSH	**100**	28.7	94.8	74.9	98.7	**99.9**	75.3	80.5	98.3	99.6	49.8	90.3	95.6	95.6	98.9	99.6	**99.6**	**99.9**
OpenStack	49.2	11.2	93.7	22.0	43.7	87.3	**100**	**97.7**	99.1	**100**	43.1	94.6	79.4	48.2	**99.2**	44.1	42.9	89.8
Mac	**94.9**	38.3	90.2	71.2	54.9	89.8	80.5	56.2	89.2	83.9	40.5	84.5	92.5	**62.8**	**94.9**	89.7	50.1	90.1
Average	88.3	44.0	92.2	70.1	83.9	96.3	90.2	**86.7**	97.3	90.4	68.8	92.3	**94.3**	**86.7**	**98.2**	92.5	85.2	97.0

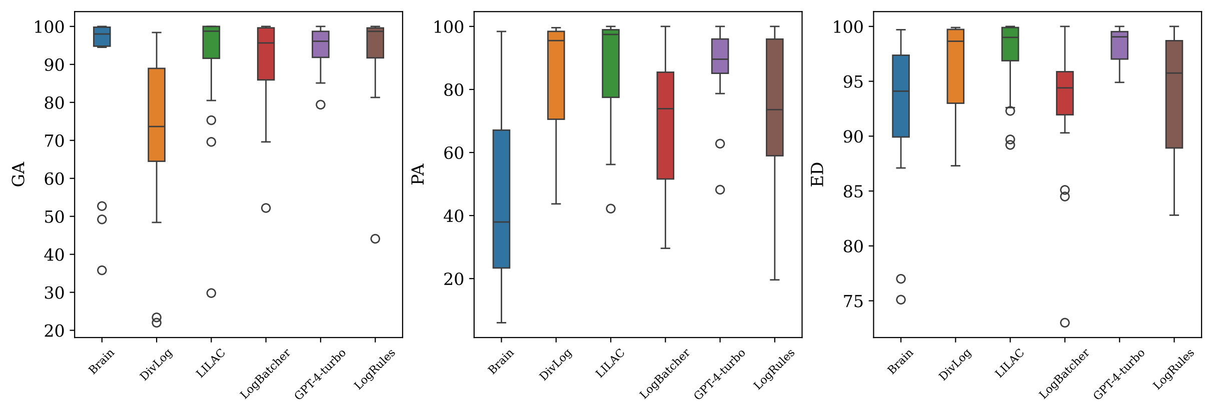


Figure 4: Robustness comparison of baselines and LogRules.

Table 2: Comparison with state-of-the-art log-based anomaly detection methods (%). The **bold** fonts indicate the best results among methods.

Method	BGL			Spirit		
	P↑	R↑	F1↑	P↑	R↑	F1↑
DeepLog	9.2	99.2	16.8	35.2	**100**	52.1
LogAnomaly	11.2	98.5	20.1	60.2	77.9	67.9
LogPrompt	24.9	83.4	38.4	29.0	99.9	45.0
LogGPT	28.0	**100**	35.6	34.0	**100**	50.7
OWL	30.1	86.6	44.6	35.4	97.2	51.8
GPT-4-turbo	25.7	79.2	38.8	36.8	99.4	53.7
GPT-4o	33.1	78.4	46.5	39.2	95.6	55.6
GPT-4o-mini	26.3	80.7	39.7	37.6	96.7	54.1
LogRules	**76.4**	85.7	**80.8**	**69.0**	**100**	**81.6**

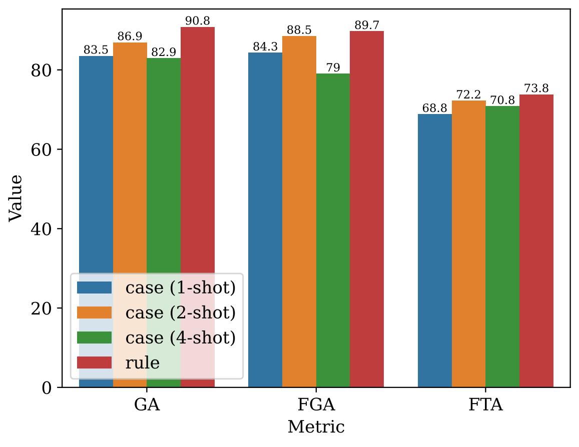


Figure 5: Comparison between case-based and rule-based reasoning for log parsing (%).

As shown in Table 3, case-based prompting performs poorly, whereas rule-based prompting achieve outstanding results. Compared to 4-shot case-based prompting, rule-based prompting improves the F1 score by 29.6% on the BGL dataset and 60.5% on the Spirit dataset.

Table 3: Comparison between case-based and rule-based reasoning for log-based anomaly detection (%). The **bold** fonts indicate the best results among methods.

Method	BGL			Spirit		
	P↑	R↑	F1↑	P↑	R↑	F1↑
Case-based (1-shot)	15.4	**99.3**	26.6	0.5	40.0	1.0
Case-based (2-shot)	17.1	**99.3**	29.1	1.5	40.0	2.9
Case-based (4-shot)	41.3	67.4	51.2	11.9	95.0	21.1
Rule-based	**76.4**	85.7	**80.8**	**69.0**	**100**	**81.6**

4.4 Comparison of Generated and Human-crafted Rules

Some studies have used human-crafted rules for log parsing. For example, Khan et al. (Khan et al., 2022) designed heuristic rules to refine identified templates. Other related works such as LILAC (Jiang et al., 2024) and LogPPT (Le and Zhang, 2023b) have also adopted such rules to fine-tune generated templates, minimizing the impact of inconsistent labels. To compare the rules generated by GPT-4o-mini with those crafted by humans, we embed both into rule-based prompts and conduct comparison experiments on the test set. The rules created by humans and generated by GPT-4o-mini can be found in Appendix G.1 and G.2, respectively.

Table 4: Comparison between human-crafted and generated rules for log parsing (%). The **bold** fonts indicate the best results between them.

	Human-crafted			GPT-4o-mini		
	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑
HDFS	**100**	**100**	**100**	99.8	99.8	**100**
Hadoop	**98.9**	**87.6**	87.3	98.6	55.2	**90.1**
Spark	**99.8**	**95.3**	**98.7**	81.3	92.9	97.7
Zookeeper	**98.9**	83.0	95.9	**98.9**	**98.7**	**99.6**
BGL	**99.2**	95.3	98.5	98.8	**96.2**	**98.9**
HPC	90.7	87.2	96.3	**95.0**	**94.4**	**99.4**
Thunderbird	91.1	83.8	93.6	**96.2**	**92.4**	**98.8**
Windows	69.1	30.4	72.9	**99.2**	**85.9**	**94.7**
Linux	**99.7**	**89.0**	96.2	87.5	86.2	**98.0**
Android	91.0	67.4	93.9	**92.4**	**70.7**	**95.2**
HealthApp	90.0	87.0	93.9	**99.6**	**98.6**	**99.5**
Apache	**100**	**99.4**	**99.9**	**100**	**99.4**	**99.9**
Proxifier	98.5	98.4	99.8	**100**	**100**	**100**
OpenSSH	97.3	90.3	97.5	**99.6**	**99.6**	**99.9**
OpenStack	**49.1**	**45.0**	**90.2**	44.1	42.9	89.8
Mac	87.6	41.0	87.3	**89.7**	**50.1**	**90.1**
Average	91.3	80.0	94.5	**92.5**	**85.2**	**97.0**

The experimental results are shown in Table 4. Overall, the rules generated by GPT-4o-mini outperform those crafted by humans, although the human-crafted rules have some advantages on the HDFS, Hadoop, Spark, Linux, and OpenStack datasets.

4.5 Performance among Different LLMs

We use LLaMA-3-8B-Instruct, LLaMA-3.1-8B-Instruct, and Qwen2.5-7B-Instruct as the LLMs for log parsing. In addition to using the average GA, PA, and ED across 16 datasets as evaluation metrics, we also measure the total inference time for each LLM. As shown in Table 5, Qwen2.5-7B-Instruct not only achieves the best performance in GA, PA, and ED metrics, but its inference speed is also approximately twice as fast as LLaMA-3-8B-Instruct.

Table 5: Comparison among different LLMs for log parsing. The **bold** fonts indicate the best results among LLMs.

LLM	GA↑	PA↑	ED↑	Time (s)↓
LLaMA-3-8B-Instruct	90.8	73.1	93.9	612.8
LLaMA-3.1-8B-Instruct	81.8	77.1	95.0	6018.2
Qwen2.5-7B-Instruct	**92.5**	**85.2**	**97.0**	**317.6**

To explore the differences in log parsing performance between LLaMA-3-8B-Instruct, LLaMA-3.1-8B-Instruct, and Qwen2.5-7B-Instruct, we analyze several log parsing examples. We find that LLaMA-3.1-8B-Instruct tends to generate Python code to implement log parsing based on the given rules. An example is shown in Appendix F.2. This approach, where Python code is first generated and then the parsing results are output, is the main reason its inference time is longer compared to LLaMA-3-8B-Instruct and Qwen2.5-7B-Instruct. In some cases, they only generate Python code without providing the final parsing results. Since LogRules does not include a Python interpreter, this leads to parsing failures.

In contrast, Qwen2.5-7B-Instruct tends to rely more on natural language for log parsing. As the experimental results demonstrate, this approach is more efficient than generating Python code first and then outputting results.

5 Conclusion

LLMs experience hallucinations when analyzing system logs, largely due to the implicit knowledge and rules in logs that LLMs struggle to capture. To address this, we propose LogRules, a lightweight log analysis framework that generates and applies rules through LLMs. Experiments show that LogRules, using LLMs with approximately 8B parameters, outperforms OpenAI GPT-based methods in both log parsing and anomaly detection, and achieves better performance compared to case-based prompting methods.

6 Limitations

Although LogRules has achieved impressive performance, as mentioned in Section 4.5, some LLMs may handle log analysis tasks by generating code. Since LogRules currently lacks a code interpreter, it cannot obtain the final output in such cases. We plan to integrate various tools into LogRules, including a code interpreter, to ensure its effective application in real-world scenarios.

7 Ethical Considerations

We use open-source log datasets for experiments. However, real-world logs may contain personal information or sensitive business data of users. Processing such data with large models could pose a risk of privacy breaches, especially when data needs to be uploaded to the cloud. The security during storage and transmission is crucial. Besides, certain regions or industries (e.g., healthcare, finance) have strict compliance requirements for data storage and processing. Using LLMs to process these logs might violate privacy regulations.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.*

Laming Chen, Guoxin Zhang, and Eric Zhou. 2018. Fast greedy MAP inference for determinantal point process to improve recommendation diversity. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5627–5638.

Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2022. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Trans. Software Eng.*, 48(3):879–892.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A survey for in-context learning. *CoRR*, abs/2301.00234.

Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 859–864. IEEE Computer Society.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 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, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1285–1298. ACM.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press.

Jan-Philipp Fränken, Nikos C Theodoropoulos, and Neil R Bramley. 2022. Algorithms of adaptation in inductive inference. *Cognitive Psychology*, 137:101506.

Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, Xu Shi, Tieqiao Zheng, Liangfan Zheng, Bo Zhang, Ke Xu, and Zhoujun Li. 2024. OWL: A large language model for IT operations. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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 2017, Honolulu, HI, USA, June 25-30, 2017*, pages 33–40. IEEE.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Yi Hu, Xiaojuan Tang, Haotong Yang, and Muhan Zhang. 2024. Case-based or rule-based: How do transformers do the math? In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Yintong Huo, Yuxin Su, Cheryl Lee, and Michael R. Lyu. 2023. Semparser: A semantic parser for log analytics. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 881–893. IEEE.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38.

Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R. Lyu. 2024. LILAC: log parsing using llms with adaptive parsing cache. *Proc. ACM Softw. Eng.*, 1(FSE):137–160.

Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R. Lyu. 2023. A large-scale benchmark for log parsing. *CoRR*, abs/2308.10828.

Suhas Kabinna, Cor-Paul Bezemer, Weiyi Shang, Mark D. Syer, and Ahmed E. Hassan. 2018. Examining the stability of logging statements. *Empir. Softw. Eng.*, 23(1):290–333.

Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel C. Briand. 2022. Guidelines for assessing the accuracy of log message template identification techniques. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 1095–1106. ACM.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*, pages 492–504. IEEE.

Van-Hoang Le and Hongyu Zhang. 2023a. Log parsing: How far can chatgpt go? In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*, pages 1699–1704. IEEE.

Van-Hoang Le and Hongyu Zhang. 2023b. Log parsing with prompt-based few-shot learning. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 2438–2449. IEEE.

Yilun Liu, Shimin Tao, Weibin Meng, Feiyu Yao, Xiaofeng Zhao, and Hao Yang. 2024. 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, ICSE Companion 2024, Lisbon, Portugal, April 14-20, 2024*, pages 364–365. ACM.

Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2022. Uniparser: A unified log parser for heterogeneous log data. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 1893–1901. ACM.

Shayne Longpre, Kartik Perisetla, Anthony Chen, Nikhil Ramesh, Chris DuBois, and Sameer Singh. 2021. Entity-based knowledge conflicts in question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 7052–7063. Association for Computational Linguistics.

Andrés Marzal and Enrique Vidal. 1993. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):926–932.

Nick McKenna, Tianyi Li, Liang Cheng, Mohammad Javad Hosseini, Mark Johnson, and Mark Steedman. 2023. Sources of hallucination by large language models on inference tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 2758–2774. Association for Computational Linguistics.

Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. 2019. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 4739–4745. ijcai.org.

Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, and Hua Cai. 2013. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Trans. Parallel Distributed Syst.*, 24(6):1245–1255.

Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2020. Self-attentive classification-based anomaly detection in unstructured logs. In *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, pages 1196–1201. IEEE.

Adam J. Oliner, Archana Ganapathi, and Wei Xu. 2012. Advances and challenges in log analysis. *Commun. ACM*, 55(2):55–61.

Adam J. Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25-28 June 2007, Edinburgh, UK, Proceedings*, pages 575–584. IEEE Computer Society.

OpenAI. 2023. Chatgpt.

OpenAI. 2024. Gpt-4o mini: advancing cost-efficient intelligence.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2463–2473. Association for Computational Linguistics.

Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol J. Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, and Zhihui Wu. 2023. Loggpt: Exploring chatgpt for log-based anomaly detection. In *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 2023, Melbourne, Australia, December 17-21, 2023*, pages 273–280. IEEE.

Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. 2024. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Team Qwen. 2024. Qwen2.5: A party of foundation models.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 5418–5426. Association for Computational Linguistics.

Siddhartha Satpathi, Supratim Deb, R. Srikant, and He Yan. 2019. Learning latent events from network message logs. *IEEE/ACM Trans. Netw.*, 27(4):1728–1741.

Jon Stearley and Adam J. Oliner. 2008. Bad words: Finding faults in spirit's syslogs. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), 19-22 May 2008, Lyon, France*, pages 765–770. IEEE Computer Society.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D. Goodman. 2024. Hypothesis search: Inductive reasoning with language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 1819–1862. Association for Computational Linguistics.

Yi Xiao, Van-Hoang Le, and Hongyu Zhang. 2024. Stronger, cheaper and demonstration-free log parsing with llms. *CoRR*, abs/2406.06156.

Haoran Xu, Amr Sharaf, Yunmo Chen, Weiting Tan, Lingfeng Shen, Benjamin Van Durme, Kenton Murray, and Young Jin Kim. 2024a. Contrastive preference optimization: Pushing the boundaries of LLM performance in machine translation. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024b. Divlog: Log parsing with prompt enhanced in-context learning. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 199:1–199:12. ACM.

Wei Xu, Ling Huang, Armando Fox, David A. Patterson, and Michael I. Jordan. 2009. Online system problem detection by mining patterns of console logs. In *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, pages 588–597. IEEE Computer Society.

Siyu Yu, Pinjia He, Ningjiang Chen, and Yifan Wu. 2023. Brain: Log parsing with bidirectional parallel tree. *IEEE Trans. Serv. Comput.*, 16(5):3224–3237.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. 2024. How language model hallucinations can snowball. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Ting Zhang, Xin Huang, Wen Zhao, Guozhao Mo, and Shaohuang Bian. 2023. Logcontrast: A weakly supervised anomaly detection method leveraging contrastive learning. In *23rd IEEE International Conference on Software Quality, Reliability, and Security, QRS 2023, Chiang Mai, Thailand, October 22-26, 2023*, pages 48–59. IEEE.

Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 807–817. ACM.

Aoxiao Zhong, Dengyao Mo, Guiyang Liu, Jinbu Liu, Qingda Lu, Qi Zhou, Jiesheng Wu, Quanzheng Li, and Qingsong Wen. 2024. 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 2024, Barcelona, Spain, August 25-29, 2024*, pages 4559–4570. ACM.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. 2023a. Loghub: A large collection of system log datasets for ai-driven log analytics. In *34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023, Florence, Italy, October 9-12, 2023*, pages 355–366. IEEE.

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 *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 121–130. IEEE / ACM.

Zhaocheng Zhu, Yuan Xue, Xinyun Chen, Denny Zhou, Jian Tang, Dale Schuurmans, and Hanjun Dai. 2023b. Large language models can learn rules. *CoRR*, abs/2310.07064.

A Details of Datasets

A.1 Log Parsing

Experiments of log parsing are conducted on the most widely-used benchmark datasets published in Loghub (Zhu et al., 2023a). The details are available in Table 6.

A.2 Log-based Anomaly Detection

We select the labeled datasets BGL (Oliner and Stearley, 2007) and Spirit (Stearley and Oliner, 2008). The datasets are split into training and test sets in a 4:1 ratio, ensuring no overlap between the two. The details are available in Table 7.

B Details of Evaluation Metrics

B.1 Log Parsing

Grouping accuracy (GA). GA (Zhu et al., 2019) assesses the ability to correctly group log messages belonging to the same template. GA is defined as the ratio of correctly grouped log messages to

Table 6: Statistical details of the datasets for log parsing.

Dataset	# Templates	# Train	# Test
HDFS	14	1	2,000
Hadoop	114	0	2,000
Spark	36	0	2,000
Zookeeper	50	1	2,000
BGL	120	1	2,000
HPC	46	1	2,000
Thunderbird	149	0	2,000
Windows	50	0	2,000
Linux	118	1	2,000
Android	166	0	2,000
HealthApp	75	1	2,000
Apache	6	0	2,000
Proxifier	8	1	2,000
OpenSSH	27	1	2,000
OpenStack	43	1	2,000
Mac	341	1	2,000

Table 7: Statistical details of the datasets for log-based anomaly detection.

Dataset	# Train (Anomalies)	# Test (Anomalies)
BGL	8,000 (611)	2,000 (147)
Spirit	8,000 (75)	2,000 (20)

the total number of log messages. A log message is regarded as correctly grouped if and only if its template aligns with the same set of log messages as that of the ground truth.

Parsing accuracy (PA). PA (Dai et al., 2022) assesses the ability to correctly extract the static templates and dynamic variables for each log message, which is essential for downstream tasks, such as anomaly detection (Du et al., 2017). PA is defined as the ratio of correctly parsed log messages to the total number of log messages. A log message is regarded as correctly parsed if and only if all tokens of templates and variables are correctly identified.

Edit distance (ED). ED assesses the performance of template extraction in terms of string comparison (Nedelkoski et al., 2020). It calculates the minimum number of actions needed to convert one template into another. We apply normalized ED (Marzal and Vidal, 1993), which computes the mean ED of all compared template pairs in the dataset (parsed templates versus ground truth templates).

F1 score of grouping accuracy (FGA). FGA (Jiang et al., 2023) is a template-level metric that focuses on the proportion of correctly grouped templates rather than log messages. Specifically, let N_g be the actual correct number of templates in the ground truth, and N_p be the number of templates that are generated by a log parser. If N_c is the number of templates that are correctly parsed by the log parser, we define the Precision of Grouping Accuracy (PGA) as $\frac{N_c}{N_p}$ and the Recall of Grouping Accuracy (RGA) as $\frac{N_c}{N_g}$. Then, we calculate FGA as their harmonic mean, i.e., $\frac{2 \times PGA \times RGA}{PGA + RGA}$.

F1 score of template accuracy (FTA). Similar to FGA, FTA (Khan et al., 2022) is a template-level metric that is calculated based on the proportion of correctly identified templates. It is computed as the harmonic mean of precision and recall of template accuracy. Differently, a template is regarded as correctly identified if and only if log messages of the parsed template share the same ground-truth template and all tokens of the template are the same as those of the ground-truth template.

B.2 Log-based Anomaly Detection

Log-based anomaly detection methods are typically evaluated with three metrics: precision, recall and F1 score, with the following definitions:

Precision (P). The percentage of correctly detected anomalous logs among all predicted anomalous logs, i.e., $\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$.

Recall (R). The percentage of correctly detected anomalous logs among all actual anomalous logs, i.e., $\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$.

F1 score (F1). The harmonic mean of precision and recall, i.e., $\mathrm{F1} = \frac{2\times\mathrm{TP}}{2\times\mathrm{TP+FP+FN}}$.

True positive (TP) denotes the log is actually anomalous and the model detects it as also anomalous, false negative (FN) denotes the log is actually anomalous, but the model incorrectly detects it as normal, and false positive (FP) denotes log is actually normal, but the model incorrectly detects it as anomalous.

C Details of Baseline Methods

C.1 Log Parsing

We select 4 open-source log parsers for baseline comparison: Brain (Yu et al., 2023), DivLog (Xu et al., 2024b), LILAC (Jiang et al., 2024), and LogBatcher (Xiao et al., 2024). Among these, Brain is a syntax-based parser, while DivLog, LILAC, and LogBatcher leverage OpenAI GPT for log parsing.

Brain. It leverages a bidirectional parallel tree to complement the longest common pattern with constant words to form complete log templates. This hierarchical tree structure allows efficient and accurate grouping of log entries. Initially, logs are grouped based on the longest common pattern of words. This pattern is treated as the root of the tree. Subsequently, words are added to the tree in two directions: (1) parent direction, which checks for missing constant words with frequencies higher than the root node, and (2) child direction, which handles words with frequencies lower than the root node to account for variations in log structures. After the tree construction, Brain generates the final log templates by combining constant and variable nodes.

DivLog. It leverages the in-context learning (ICL) (Dong et al., 2023) capabilities of LLMs, particularly GPT-3 (Brown et al., 2020). It uses a few examples within a prompt to guide the model in generating log templates for unseen logs. This method avoids the need for task-specific training or handcrafted features. Before log parsing, DivLog samples a small number of diverse logs from the dataset, maximizing sample diversity to create a labeled candidate set. During parsing, DivLog selects the most relevant examples from the candidate set for each new log. These examples are formatted into a prompt, which is then used by GPT-3 to generate the log template. Moreover, the prompt includes a clear instruction along with relevant examples, helping the LLM to extract common patterns and generate accurate log templates. To ensure output quality, the template format is restricted using special locators (e.g., <START> and <END>). To improve robustness, DivLog uses determinantal point process (DPP) (Chen et al., 2018) for diverse log sampling and K-nearest neighbors (KNN) for selecting the most relevant examples. This helps avoid bias and ensures that examples are diverse and similar to the query log.

LILAC. It leverages LLMs and an adaptive parsing cache to address three challenges: the lack of specialized log parsing capabilities in LLMs, inconsistency in outputs, and the high computational overhead of using LLMs for log parsing tasks. It employs hierarchical candidate sampling algorithm to select diverse and representative log messages from which effective demonstration examples are chosen. To improve efficiency and mitigate the high computational cost of LLM inference, LILAC introduces an adaptive parsing cache. This cache stores previously generated log templates and allows the system to reuse them for similar logs, reducing the need for repeated LLM queries.

LogBatcher. It uses a clustering algorithm DBSCAN (Ester et al., 1996) to partition logs into multiple sections, ensuring that logs within the same partition share common characteristics. It also implements a caching mechanism to store parsed log templates, which are then matched with logs in the current partition to avoid redundant LLM queries, improving parsing efficiency. Additionally, it proposes a batching-query method, where diverse logs are sampled from each partition and sent as a batch to the LLMs for parsing. This approach allows LLMs to better perform log parsing tasks by leveraging the diversity of input logs, even without labeled examples.

C.2 Log-based Anomaly Detection

We evaluate the performance of LogRules and compare it with influential baseline methods, including deep learning-based methods Deeplog (Du et al., 2017) and LogAnomaly (Meng et al., 2019), LLM-based methods LogPrompt (Liu et al., 2024), LogGPT (Qi et al., 2023) and OWL (Guo et al., 2024). The details of these methods are as follows:

DeepLog. It focuses on learning and detecting anomalies from log sequences. In the training stage, an LSTM (Hochreiter and Schmidhuber, 1997) is trained to capture the sequential patterns of normal logs. During the detection stage, the model predicts the next log event based on the current sequence. A sequence is classified as normal if the predicted event matches the actual event, and anomalous otherwise.

LogAnomaly. Similar to DeepLog, it employs LSTM as backbone model for learning and detection from both log sequences and log parameters. It takes into account the semantic information of logs by Template2Vec, which also considers synonyms and antonyms in the log text. Log key sequences and log count vectors are as inputs, and the LSTM is trained to perform anomaly detection.

LogPrompt. It employs ChatGPT (OpenAI, 2023) to perform online log analysis tasks via a suite of advanced prompt strategies tailored for log tasks, which enhances the performance of ChatGPT.

LogGPT. It aims to explore the transferability of knowledge from large-scale corpora to log-based anomaly detection by leveraging the language interpretation capabilities of ChatGPT.

OWL. It is an LLM trained on constructed OWL-Instruct with a wide range of IT-related information. Specifically, limited by the maximum input length, the homogeneous Markov context extension (HMCE) method is proposed. The mixture-of-adapter strategy is leveraged to improve the parameter-efficient tuning across different domains or tasks.

D Hyperparameter Settings for Alignment

We use CPO to enhance the rule-based reasoning capabilities of smaller LLMs. The hyperparameters settings for alignment are shown in Table 8.

Table 8: The hyperparameter settings for alignment.

Hyperparameter	Value
β (Factor for CPO loss)	0.1
α (Factor for BC regularizer)	1.0
learning_rate	2e-4
weight_decay	1e-3
max_grad_norm	0.3
max_length	1024
lr_scheduler_type	linear
num_train_epochs	1
warmup_ratio	0.03
per_device_train_batch_size	1
gradient_accumulation_steps	1

E Additional Results

The experimental results of GPT-4o and GPT-4o-mini for log parsing are presented in Table 9. It is evident that the average performance of both GPT-4o and GPT-4o-mini falls short compared to LogRules, as detailed in Table 1.

F Prompting Details

F.1 Induction Stage

The details of prompts for log parsing and log-based anomaly detection are shown in Figure 6 and 7, respectively.

F.2 Deduction Stage

The details of prompts for log parsing and log-based anomaly detection are shown in Figure 8 and Figure 9, respectively.

Moreover, we observe that LLaMA-3.1-8B-Instruct tends to generated Python code for handling log parsing. An example can be seen in Figure 10.

Table 9: The performance of GPT-4o and GPT-4o-mini for log parsing (%).

	GPT-4o-mini			GPT-4o		
	GA↑	PA↑	ED↑	GA↑	PA↑	ED↑
HDFS	100	100	100	100	100	100
Hadoop	96.2	92.1	93.8	98.8	94	97.4
Spark	99.7	96.8	98.3	94.2	95.6	97.8
Zookeeper	98.6	94.3	95.1	99.1	98.9	99.7
BGL	94	86.6	96.1	99.8	91.8	98.9
HPC	95.1	84.7	94.6	95.3	92.6	99.4
Thunderbird	97.8	93	97.4	94.2	92.1	98.7
Windows	71.6	83.6	94.2	96.2	87.2	95.3
Linux	85.4	80.7	94.7	95.3	52.2	92.7
Android	94.4	79.6	95.8	97.9	81	96.6
HealthApp	92.5	86.9	94.2	88.9	89.3	96.6
Apache	100	89.1	94.1	100	100	100
Proxifier	52.2	83.2	95.5	86.1	82.6	97.1
OpenSSH	99.6	90.8	97.3	92.5	93.5	100
OpenStack	100	43.1	96.6	45.1	45	91.2
Mac	89.9	60.5	88.5	90.3	52	91.2
Average	91.7	84.1	95.4	92.1	84.2	97.0

G Rules

G.1 Human-crafted Rules

The details of human-crafted rules for log parsing are shown in Table 10.

G.2 Generated Rules

The details of rules generated by GPT-4o-mini for log parsing and log-based anomaly detection are shown in Table 11 and 12, respectively.

Log parsing

System
You will be given some examples of log parsing, you need to induce some parsing rules based on these examples. The examples are given with the format "<raw_log> → <parsed_log>". You must output the induced rules with JSON format containing keys "task" and "rules", where "task" is "log_parsing" and "rules" is a list of strings.

User
Given examples:
{EXAMPLES}

Figure 6: The prompt for log parsing in induction stage, with blue fonts indicating the input parts.

Log-based anomaly detection

System
You will be given some examples of log-based anomaly detection, you need to induce some detection rules based on these examples.The examples are given with the format "<raw_log> → <category>". You must output the induced rules with JSON format containing keys "task" and "rules", where "task" is "anomaly_detection" and "rules" is a list of strings.

User
Given examples:
{EXAMPLES}

Figure 7: The prompt for log-based anomaly detection in induction stage, with blue fonts indicating the input part.

Table 10: Human-crafted rules for log parsing.

Rule description	Example (before → after)
Replace double spaces with a single space	Input:__{{variable}} → Input:_{{variable}}
Replace digit tokens with variables	euid=0 → euid={{variable}}
Replace True/False with a variable	cancel=false → cancel={{variable}}
Replace a path-like token with a variable	/lib/tmp started → {{variable}} started
Replace a token containing both fixed and variable parts with a variable	python v{{variable}} → python {{variable}}
Replace consecutive variables as a single variable	value={{variable}}{{variable}} → value={{variable}}
Replace dot separated variables as a single variable	{{variable}}.{{variable}} seconds → {{variable}} seconds

Table 11: The rules generated by GPT-4o-mini for log parsing.

Rank	Rule description
1	Replace sequences of numbers with {{variable}}
2	Replace IP addresses with {{variable}}
3	Replace host names with {{variable}}
4	Replace any occurrence of 'size <number>' with 'size {{variable}}'
5	Replace any occurrence of 'blk_<number>' with '{{variable}}'
6	Replace any sequence indicating a user with 'user={{variable}}'
7	Replace any 'uid=<number>' and 'euid=<number>' with 'uid={{variable}}' and 'euid={{variable}}' respectively
8	Keep the structure and text for logs that have no variable parts
9	Replace time indicators like '<number> s' with '{{variable}} s'
10	For statements resembling an event or action with a number, replace the number with {{variable}}
11	For logs indicating a connection or disconnection, replace specific addresses and ports with {{variable}}

Log parsing

System
You will be provided with some log messages separated by line break. You must abstract variables with {{placeholders}} to extract the corresponding template. There might be no variables in the log message. Here are some rules you can refer to:
{RULES}
Print the input log's template delimited by backticks.

User
Log message:
``` `{LOG MESSAGES}` ```
Log template:

System
You will be provided with some log messages separated by line break. You must abstract variables with {{placeholders}} to extract the corresponding template. There might be no variables in the log message.
Print the input log's template delimited by backticks.

User
Log message:
``` `{CASES OF LOG MESSAGES}` ```
Log template:
``` `{CASES OF LOG TEMPLATES}` ```
Log message:
``` `{LOG MESSAGES}` ```
Log template:

Figure 8: The prompts for log parsing in deduction stage, with blue fonts indicating the input parts. Top: rule-based prompting; Bottom: case-based prompting.

Table 12: The rules generated by GPT-4o-mini for log-based anomaly detection.

Rank	Rule description
1	If log contains 'data storage interrupt', then label as 'Anomalous'
2	If log contains 'data TLB error interrupt', then label as 'Anomalous'
3	If log contains 'rts: kernel terminated for reason', then label as 'Anomalous'
4	If log contains 'ciod: failed to read message prefix', then label as 'Anomalous'
5	If log contains 'ciod: Error reading message prefix', then label as 'Anomalous'
6	If log contains 'status error: status=0x00', then label as 'Anomalous'
7	If log contains 'drive not ready for command', then label as 'Anomalous'
8	If log contains 'system epilog failed w/rc=1', then label as 'Anomalous'

Log-based anomaly detection

System
You will be given some rules of log-based anomaly detection with JSON format, you need to utilize these rules to determine if the incoming logs are anomalous. You must output the results with JSON format containing keys "task" and "category", where "task" is "anomaly_detection" and "category" is "Normal" or "Anomalous".
Rules:
{RULES}

User
Given logs:
{LOG MESSAGES}

System
You will be given some cases of log-based anomaly detection with JSON format, you need to utilize these cases to determine if the incoming logs are anomalous. You must output the results with JSON format containing keys "task" and "category", where "task" is "anomaly_detection" and "category" is "Normal" or "Anomalous".
Cases:
{CASES}

User
Given logs:
{LOG MESSAGES}

Figure 9: The prompts for log-based anomaly detection in deduction stage, with blue fonts indicating the input parts. Top: rule-based prompting; Bottom: case-based prompting.

~~~
```python
import re

def extract_template(log_message):
  # Replace sequences of numbers with {{variable}}
  log_message = re.sub(r'\d+', '{{variable}}', log_message)

  # Replace IP addresses with {{variable}}
  log_message = re.sub(r'00[0-9a-fA-F]{6}', '{{variable}}', log_message)

  # Replace host names with {{variable}}
  log_message = re.sub(r'[0-9a-fA-F]{8}', '{{variable}}', log_message)

  # Replace any occurrence of 'size <number>' with 'size {{variable}}'
  log_message = re.sub(r'size␣\d+', 'size␣{{variable}}', log_message)

  # Replace any occurrence of 'blk_<number>' with 'blk_{{variable}}'
  log_message = re.sub(r'blk_\d+', 'blk_{{variable}}', log_message)

  # Replace any sequence indicating a user with 'user={{variable}}'
  log_message = re.sub(r'dear␣[0-9a-fA-F]{8}', 'user={{variable}}',
      log_message)

  # Replace any 'uid=<number>' and 'euid=<number>' with 'uid={{variable}}' and
       'euid={{variable}}' respectively
  log_message = re.sub(r'uid=\d+', 'uid={{variable}}', log_message)
  log_message = re.sub(r'euid=\d+', 'euid={{variable}}', log_message)

  # Keep the structure and text for logs that have no variable parts
  log_message = re.sub(r'\d+␣s', '{{variable}}␣s', log_message)

  # For statements resembling an event or action with a number, replace the
      number with {{variable}}
  log_message = re.sub(r'\d+', '{{variable}}', log_message)

  # For logs indicating a connection or disconnection, replace specific
      addresses and ports with {{variable}}
  log_message = re.sub(r'iar␣[0-9a-fA-F]{8}', 'iar␣{{variable}}', log_message)

  return log_message

log_message = """
iar 003a90fc dear 00b360e8
iar 003a90dc dear 00b35148
iar 003a9108 dear 00c07348
iar 003a9294 dear 00c06c48
iar 003a926c dear 00f2b078
iar 00149fc0 dear 00739d98
iar 001061fc dear 02f6676c
iar 0014a1e4 dear 0099e6c8
iar 003a9258 dear 0136bd38
iar 00105e94 dear 02f5883c
"""

print(f"`{extract_template(log_message)}`")
```
~~~

Figure 10: An example of code generated by LLaMA-3.1-8B-Instruct during log parsing.