LLMParser: An Exploratory Study on Using Large Language Models for Log Parsing

Zeyang Ma
Software PEformance, Analysis and Reliability (SPEAR) Lab
Concordia University
Montreal, Quebec, Canada
m_zeyang@encs.concordia.ca

An Ran Chen
Electrical and Computer Engineering Department
University of Alberta
Edmonton, Alberta, Canada
anran6@ualberta.ca

Dong Jae Kim
Software PEformance, Analysis and Reliability (SPEAR) Lab
Concordia University
Montreal, Quebec, Canada
k_dongja@encs.concordia.ca

Tse-Hsun (Peter) Chen
Software PEformance, Analysis and Reliability (SPEAR) Lab
Concordia University
Montreal, Quebec, Canada
peterc@encs.concordia.ca

Shaowei Wang
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada
shaowei@cs.umanitoba.ca

ABSTRACT

Logs are important in modern software development with runtime information. Log parsing is the first step in many log-based analyses, that involve extracting structured information from unstructured log data. Traditional log parsers face challenges in accurately parsing logs due to the diversity of log formats, which directly impacts the performance of downstream log-analysis tasks. In this paper, we explore the potential of using Large Language Models (LLMs) for log parsing and propose LLMParser, an LLM-based log parser based on generative LLMs and few-shot tuning. We leverage four LLMs, Flan-T5-small, Flan-T5-base, LLaMA-7B, and ChatGLM-6B in LLMParsers. Our evaluation of 16 open-source systems shows that LLMParser achieves statistically significantly higher parsing accuracy than state-of-the-art parsers (a 96% average parsing accuracy). We further conduct a comprehensive empirical analysis on the effect of training size, model size, and pre-training LLM on log parsing accuracy. We find that smaller LLMs may be more effective than more complex LLMs; for instance where Flan-T5-base achieves comparable results as LLaMA-7B with a shorter inference time. We also find that using LLMs pre-trained using logs from other systems does not always improve parsing accuracy. While using pre-trained Flan-T5-base shows an improvement in accuracy, pre-trained LLaMA results in a decrease (decrease by almost 55% in group accuracy). In short, our study provides empirical evidence for using LLMs for log parsing and highlights the limitations and future research direction of LLM-based log parsers.

KEYWORDS

Log parsing, log analysis, large language model

ACM Reference Format:
Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Hsun (Peter) Chen, and Shaowei Wang. 2024. LLMParser: An Exploratory Study on Using Large Language Models for Log Parsing. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639150

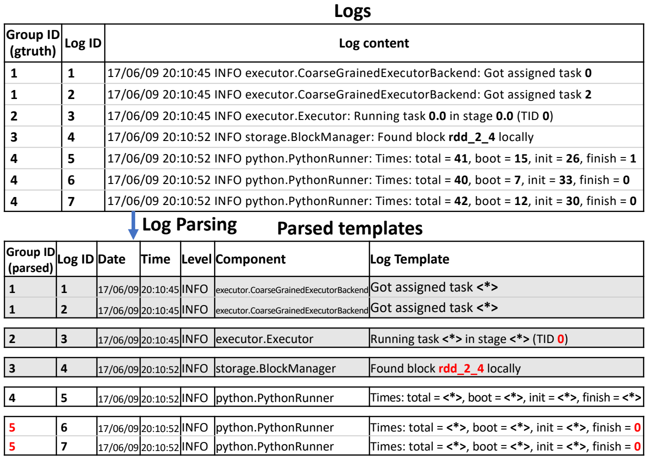
1 INTRODUCTION

Software logs provide developers with valuable system runtime information to understand system execution and debugging. However, due to the sheer volume and complexity of logs, analyzing logs manually becomes infeasible. To assist with log analysis, researchers have proposed many automated approaches for various tasks such as anomaly detection [26, 72], monitoring [5, 63], and root cause analysis [25, 43, 67]. Among all log analysis tasks, log parsing often serves as the first step of log analysis.

The goal of log parsing is to convert raw log data into log templates by identifying and separating static text and variable values in the log messages. As shown in Figure 1, logs contain dynamic information such as the timestamp, log level, and log message (which contains static message and dynamic variable values). Log parsing first extracts consistent information among all the logs using regular expression (e.g., timestamps and log level), and then transforms logs into a more structured format (i.e., log template) by identifying variables in the log message [23, 37]. For instance, the log message in Log 1 from Figure 1 has the log message `Got assigned task 0` and the log can be parsed to the log template `Got assigned task <*>` with an identified variable `0`. The variables record system runtime information that can be in various forms (e.g., string, digits, or symbols). Including such dynamic variable values in the logs makes automated log analysis difficult. Hence, log parsing is an essential first step in log analysis, and having low accuracy in log parsing results directly impacts the performance of downstream tasks [23, 24, 37].

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3639150



Logs

Group ID (gtruth)	Log ID	Log content
1	1	17/06/09 20:10:45 INFO executor.CoarseGrainedExecutorBackend: Got assigned task **0**
1	2	17/06/09 20:10:45 INFO executor.CoarseGrainedExecutorBackend: Got assigned task **2**
2	3	17/06/09 20:10:45 INFO executor.Executor: Running task **0.0** in stage **0.0** (TID **0**)
3	4	17/06/09 20:10:52 INFO storage.BlockManager: Found block **rdd_2_4** locally
4	5	17/06/09 20:10:52 INFO python.PythonRunner: Times: total = **41**, boot = **15**, init = **26**, finish = **1**
4	6	17/06/09 20:10:52 INFO python.PythonRunner: Times: total = **40**, boot = **7**, init = **33**, finish = **0**
4	7	17/06/09 20:10:52 INFO python.PythonRunner: Times: total = **42**, boot = **12**, init = **30**, finish = **0**

Log Parsing

Parsed templates

Group ID (parsed)	Log ID	Date	Time	Level	Component	Log Template
1	1	17/06/09	20:10:45	INFO	executor.CoarseGrainedExecutorBackend	Got assigned task <*>
1	2	17/06/09	20:10:45	INFO	executor.CoarseGrainedExecutorBackend	Got assigned task <*>
2	3	17/06/09	20:10:45	INFO	executor.Executor	Running task <*> in stage <*> (TID 0)
3	4	17/06/09	20:10:52	INFO	storage.BlockManager	Found block **rdd_2_4** locally
4	5	17/06/09	20:10:52	INFO	python.PythonRunner	Times: total = <*>, boot = <*>, init = <*>, finish = <*>
5	6	17/06/09	20:10:52	INFO	python.PythonRunner	Times: total = <*>, boot = <*>, init = <*>, finish = 0
5	7	17/06/09	20:10:52	INFO	python.PythonRunner	Times: total = <*>, boot = <*>, init = <*>, finish = 0

Figure 1: An example of log parsing and validating the result from Spark. The incorrectly parsed results are highlighted in red.

Despite the importance of log parsing, effectively parsing logs remains a challenging task. There are many prior research that proposed various log parsers [12, 19, 24, 59, 76]. Yet, recent studies [30, 75] show that these approaches often fail to identify parameters in logs, which may affect the downstream log analysis tasks. Recently, Large Language Models (LLMs) have demonstrated promising results in text-related and code-related tasks, such as code understanding and generation [35, 66]. Intuitively, log is composed of both natural language and code-like variables. LLMs’ strong ability for language translation can be potentially leveraged for log parsing, which can also be viewed as translating from logs to log templates.

In this paper, we investigate the potential of using LLMs for log parsing, with a focus on studying the effect of varying LLMs, shot sizes, and pre-training particularly when working with limited training data. We proposed LLMParser, an innovative log parser. LLMParsers learn from few-shot examples on how to “translate” a log into a log template and evaluate LLMParser using four text-to-text or text generation LLMs: Flan-T5-small [10], Flan-T5-base [10], LLaMA-7B [58], and ChatGLM-6B [68]. We train and evaluate LLMParsers using a widely-used log benchmark that contains logs data from 16 open-source systems [27, 30]. Our evaluation shows that 1) LLMParsers can achieve an average parsing accuracy (PA) of 0.96, which is higher state-of-the-arts parsers Drain [24], Logram [12], and LogPPT [33] (among them, the highest PA is 0.92). 2) Few-shot tuning is more effective than in-context learning, where in-context learning only results in an average PA of 0.46. 3) Increasing the number of training examples may not always give better parsing results; data diversity and balance may be more important. 4) More complex LLMs may not always give better results. We find that Flan-T5-base, which only has 240M parameters, can achieve similar results compared to LLaMA which has 7B parameters. 5) LLM pre-trained using logs from other systems may not always help improve PA. We find opposite findings between Flan-T5-base and LLaMA, where LLaMA experiences a decrease in parsing accuracy while Flan-T5-base has an improved parsing result.

We summarize the main contributions of this paper as follows:

- We explore the use of LLMs for log parsing, and propose LLMParser, a generative LLM-based approach for log parsing. LLMParser achieves a higher parsing accuracy (PA) compared to state-of-the-arts.
- We compare in-context learning and few-shot tuning and found that few-shot tuning achieves a much higher PA (up to 0.96 v.s. 0.46). We also found that few-shot tuning is efficient, which only takes from one to five minutes on an NVIDIA A100 GPU.
- We found that increasing training shot sizes may not always improve PA. Future studies should explore better sampling approaches to improve LLM-based log parsers.
- LLMs with more parameters may not always give better PA. We find that a medium-size LLM (Flan-T5-base) achieves comparable performance compared to LLaMA-7B. Future studies should consider the trade-off between model complexity and accuracy.
- We find that using LLMs pre-trained using logs from other systems may not necessarily improve PA. We saw contradictory results in LLaMA and Flan-T5-base, where the parsing accuracy using LLaMA decreases. Future studies are needed to explore the impact and effectiveness of pre-trained log models on log parsing.

Paper Organization. Section 2 discusses background and related work. Section 3 provides the details of LLMParser. Section 4 shows experiment setup and our implementation. Section 5 evaluates LLMParser. Section 6 discusses the implications of our findings. Section 7 discusses threats to validity. Section 8 concludes the paper.

Data Availability: We made our source code and experimental results publicly available at: https://github.com/zeyang919/LLMParser

2 BACKGROUND AND RELATED WORK

In this section, we discuss the background of Large Language Models (LLMs) and how to optimize LLMs on specific tasks. We also discuss related work, existing log parsing approaches, and applications that use LLMs to solve log-related tasks.

2.1 Background

Large Language Models. The Large Language Models (LLMs) are mostly developed based on the transformer architecture [50, 58, 68]. LLMs have made important advancements in natural language processing (NLP) by providing models that have an extraordinary capacity for understanding language and producing contextually relevant and semantically consistent text. LLMs are generally pre-trained on a large corpus of text data from diverse sources such as books, articles, websites, and even source code. Recent studies [35, 66] have highlighted the capability of LLMs in code recognition, understanding, and code generation.

As logs consist of both natural language sequences and code-like variables, prior work [6, 12, 22, 36] has leveraged language models to analyze logs. However, it remains unclear whether LLMs can be effectively used for log parsing due to the varying pre-training data and model characteristics. Adopting LLMs for log parsing brings potential advances in the research area. First, LLMs are shown to be very powerful on text-related tasks [35, 66], which may be able to achieve more accurate log parsing results. Second, LLMs are generalizable on unseen data [29, 61, 73], which may be leveraged to parse new logs without continuous retraining. Nevertheless, there is a need for a comprehensive study on using LLMs for log parsing and what kinds of adaptions are needed for logs.

In-context Learning and Fine-Tuning of Large Language Models. To adapt LLMs to specific tasks, there are two main strategies: in-context learning and few-shot tuning. In-context learning [4, 62] is a method that incorporates task-specific demonstrations directly into the input (i.e., prompt) during LLMs' inference, guiding the model to generate responses in a desired manner without the need for retraining/changing the model's parameters. In-context learning relies on the model's ability to generalize from the provided demonstrations to understand and execute the task at hand. On the other hand, fine-tuning [15, 20, 49] involves re-training the pre-trained LLM on a dataset tailored to the specific task, allowing the model to adjust its internal parameters and better align its outputs with the desired outcomes. In particular, few-shot tuning [38, 46] is a fine-tuning method that enables LLMs to generalize from limited examples, which may facilitate the extraction of relevant information across diverse log formats, including log variables, log structure, and semantic patterns.

While in-context learning provides quick adaptability, it has several drawbacks. First, processing prompts with several demonstrations every time the model parses logs can contribute to further computational costs. In-context learning prompted with few-shot demonstrations requires the model to process both the target instance and all the demonstrations during each inference, leading to an increased inference time. Second, the context size of the model's inputs limits the number of demonstrations that can be used. For example, performing in-context learning with four prompts on Flan-T5-Base [10] exceeds its context size of 512 tokens. This limitation poses a challenge for LLMs to learn from a larger number of demonstrations and improve their performance. Finally, selecting effective demonstrations is also crucial for improving the performance of in-context learning, as it is sensitive to the format and order of the prompts [16, 44, 45].

In contrast, few-shot tuning does not demand continuous in-context demonstrations for every inference, which can speed up inference time. Moreover, using fine-tuning, we can provide more diverse log examples to train the model as the tuning is already performed during training. Prior studies [4, 20, 38] have also demonstrated that few-shot tuning offers better accuracy at lower computational costs. Furthermore, since few-shot tuning only involves a small number of data samples, the entire fine-tuning process can be fast (e.g., only a few minutes for our approach). As a result, there is no significant time overhead incurred due to fine-tuning. Therefore, in this paper, we utilize few-shot tuning to integrate domain-specific knowledge into LLMs.

2.2 Related Work

We discuss related work along two directions: log parsing and using LLM for other log-related tasks.

Log Parsing. To support log parsing for large volumes of logs, researchers have proposed many automated log parsing techniques. Existing log parsers primarily use three approaches: frequent pattern mining, log clustering, and parsing trees. (1) Frequent pattern mining identifies static text and variables by counting the number of times a pattern or sequence recurs in the logs (e.g., SLCT [59], LogCluster [76], Logram [12]). (2) Log clustering groups logs using clustering algorithms, thereby categorizing logs into different groups (e.g., LKE [19], LogSig [56], and LenMa [53]). (3) Parsing tree builds a parse tree with fixed depth to guide the log group search process (e.g., Drain [24]). These studies aim to strike a balance between optimizing accuracy and the size of pre-learned data for log parsing tasks. While the accuracy of log parsing has shown improvement over time, recent studies [30, 75] reveal that traditional algorithm-based log parsing tools primarily emphasize log clustering. Although these approaches achieve high grouping accuracy, they often fail to accurately identify all the variables in logs [30]. Therefore, this limitation may hinder downstream log analysis tasks, such as missing some anomalies recorded by unrecognized variables during log anomaly detection [37].

The recent rise of LLMs has brought new possibilities for improving log parsing. Le and Zhang [33] proposed LogPPT, which is a log parser based on a masked language model (RoBERTa [40]). LogPPT improved the accuracy of log parsing compared to traditional algorithm-based log parsers. LogPPT converts the log parsing task into a token classification task to classify whether a token in the log is variable or static. However, this process requires more manual effort in labelling every token in a log on whether or not a token is a static text. Le and Zhang [32] further evaluated using ChatGPT [4] to parse logs. Their study shows that ChatGPT can parse the logs but the accuracy is worse than LogPPT. However, due to the closed-source nature of ChatGPT, the monetary cost can be high, the LLM is not fine-tunable, and the stability of the LLM is out of the control of the developers. More importantly, logs often contain sensitive data that cannot be sent to third parties.

In this paper, we investigate the application of text generation and text2text generation LLMs to tackle the log parsing task. The recent advancements and ongoing research in LLMs, particularly in text2text and text generation, have significantly improved their text understanding and processing capabilities [4, 9, 58]. Intuitively, log parsing is similar to language translation, where a log is translated into a log template. This allows us to eliminate the process of splitting logs and manually labeling individual tokens, and the parsed log template is directly obtained. We leverage four open-source LLMs (Flan-T5-Base [10], Flan-T5-Small [10], LLaMA-7B [58], and ChatGLM-6B [68]) to generate the log template by inputting the prompt and explore the performance of the LLMs compared with the state-of-the-art approaches. Furthermore, we study the limitations of LLM-based log parsers and explore the potential of using pre-trained LLM-based log parsers.

Using LLMs for Other Log-related Tasks. The recent advances in LLMs have shown success in both natural language processing and code generation [35, 51, 66]. Since logs are semi-structured text composed of natural language with some code elements, researchers have adopted LLMs for log-related tasks [7, 34, 39]. Some studies [34, 39] used LLMs for log anomaly detection. Lee et al. [34] proposed LAnoBERT which is an anomaly log detector. LAnoBERT masked the specific word in the log and then used BERT [13] to predict the masked word and calculate the predictive probability of the origin masked word. When there are large differences between the actual and the predicted words, the respective log is identified as an

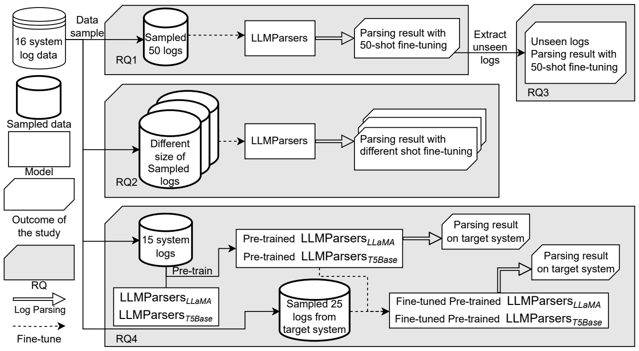

16 system log data
Data sample
Sampled 50 logs
LLMParsers
Parsing result with 50-shot fine-tuning
RQ1
Extract unseen logs
Unseen logs Parsing result with 50-shot fine-tuning
RQ3
Sampled data
Different size of Sampled logs
LLMParsers
Parsing result with different shot fine-tuning
RQ2
Model
Outcome of the study
RQ
Log Parsing
Fine-tune
15 system logs
Pre-train
Pre-trained LLMParsers$_{LLaMA}$
Pre-trained LLMParsers$_{T5Base}$
Parsing result on target system
LLMParsers$_{LLaMA}$
LLMParsers$_{T5Base}$
Sampled 25 logs from target system
Parsing result on target system
Fine-tuned Pre-trained LLMParsers$_{LLaMA}$
Fine-tuned Pre-trained LLMParsers$_{T5Base}$
RQ4

Figure 2: An overview of the evaluation of LLMParsers.

anomaly. Liu et al. [39] conducted a case study on logs from Huawei Cloud and found that the effectiveness of the anomaly detected by ChatGPT was partially consistent with that of the on-call engineers, which suggests that LLMs could potentially reduce manual verification efforts. Chen et al. [7] introduced RCACopilot, an on-call system integrated with OpenAI’s GPT models for automating root cause analysis of cloud incidents.

3 APPROACH

In this section, we introduce LLMParsers, which adopts LLMs for log parsing. LLMParsers tackle log parsing as a text2text and text generation task using few-shot fine-tuning. Figure 2 illustrates our overall evaluation architecture of LLMParser. In RQ1, we sample 50 logs from each system using our sampling algorithm for fine-tuning LLMParsers, and evaluate the log parsing result. In RQ2, we conduct a sensitivity analysis on the number of shots used for fine-tuning. In RQ3, we evaluate the effectiveness of LLMParsers on unseen log templates. Finally, in RQ4, evaluate the log parsing accuracy of using a pre-trained LLMParser using 15 other systems. Below, we present our few-shot sampling algorithm, LLM selection, prompt selection, and fine-tuning process.

3.1 Sampling Few-Shot Data

LLMs require data samples to learn how to process different requests, a task often accomplished by providing a few training examples [32, 33]. Similar to most other log parsers (e.g., Logram, Drain, and LogPPT), LLMParser is an offline parser. Log parsers typically need to scan all available logs to identify patterns and abstract dynamic values, a process that is inherently offline. Access to all necessary logs is a prerequisite for this procedure, enabling the application of various techniques like clustering and log sampling. Hence, we can apply our clustering and log sampling techniques to sample a small number of logs and their associated log templates to train the LLMs. We prioritize the sampling of more commonly-seen logs while ensuring data diversity. Prior studies [21, 55] also suggest that increasing the diversity of training data is effective in improving the understanding and generalization ability of deep learning models. Therefore, we proposed a data sampling algorithm to sample logs with high frequency and variety to increase the coverage and diversity of the training dataset.

Algorithm 1: Few-shot Log Sampling

```
Input  : $D$: Dataset containing raw logs
Input  : $N$: Number of logs to sample
Output : $D_{train}$: Dataset with $N$ sampled logs and log templates
/* Extract timestamps and replace numbers */
processed_logs ← process_raw_logs($D$);
log_clusters ←mean_shift_clustering (processed_logs);
$D_{train} \leftarrow \varnothing$;
foreach cluster in sorted (log_clusters, “descend”) do
    log ← sample_one_log(cluster);
    $D_{train}$.add (log, find_template(log));
    if length of $D_{train} = N$ then
        break;
    end if
end foreach
return $D_{train}$
```

Algorithm 1 demonstrates our log sampling process. The sampling process does not require any pre-labelled data and is unsupervised. Firstly, similar to other log parsers [33, 70], we process the raw logs to separate and remove the information generated by logging frameworks, such as dates and timestamps, and extract the log messages. We further process the logs by using regular expressions to replace all the numbers in the log with a unified character to minimize the effect of the dynamically generated values on the next step. Secondly, we apply the Mean Shift [8] clustering algorithm to cluster the processed logs. We choose Mean Shift because, unlike K-means, it does not need the user to specify the number of clusters in advance. However, other clustering algorithms can also be used. Thirdly, we sort the generated clusters in descending order based on the cluster size. Finally, we sample one log from every cluster and repeat the process until we reach the desired number of samples. By sampling and iterating from the largest cluster, we consider both diversity and coverage (i.e., possibly covering more logs) in the sampling process. We label the log templates for all the sampled logs to guide the LLM on how to parse logs.

3.2 LLMParser: Using LLMs for Parsing Logs

Parsing Logs Using Text2text and Text Generation LLMs. When using text2text or text generation LLMs, we can parse a log by simply giving them a log message. The LLMs “translate” the log into a log template. Compared to LogPPT [33], which uses LLMs for token classification, log parsing using text2text and text generation LLMs eliminates the log splitting and output conversion process. The LLMs directly use the logs for input and output, which fully leverages LLMs’ abilities and makes the parsing result more intuitive and easier to diagnose parsing issues.

We evaluate and compare four LLMs on their log parsing accuracy: Flan-T5-small [10], Flan-T5-base [10], LLaMA-7B [58] and ChatGLM-6B [68]. Table 1 shows the architecture of the LLMs and the parameter size. The LLMs cover both text2text (Flan-T5) and text generation (LLaMa and ChatGLM), vary in size (range from 80M to 7B parameters), and are pre-trained using different architectures. Our goal is to explore the difference in log parsing accuracy among LLMs with different architectures and parameter sizes. Flan-T5-small and Flan-T5-base are the instruction fine-tuned versions of T5 [50] with different parameter sizes. Prior research [10, 52]

Table 1: Information on the Large Language Models that we used for log parsing.

LLM Name	Architecture	Pre-training Objective	Parameter Size
Flan-T5-small	Encoder-decoder	Text2text Generation	80M
Flan-T5-base	Encoder-decoder	Text2text Generation	240M
LLaMA-7B	Causal Decoder	Text Generation	7B
ChatGLM-6B	Prefix Decoder	Text Generation	6B

showed that Flan-T5 converges faster than T5 and achieves outstanding results on fine-tuned tasks. LLaMA-7B is a publicly released state-of-the-art foundational large language model by Meta, which offers smaller and more efficient models for researchers on multiple NLP tasks. ChatGLM-6B was jointly built by Tsinghua University and Zhipu AI Company [68] that enable widespread access to researchers for question answering and information processing. Additionally, they are open-source models, which allows for easy in-depth analysis and fine-tuning processes, as well as the replication of our study in future research.

Prompt Templates for Log Parsing. Prompts are user-provided inputs, such as queries, instructions, or questions, that guide LLMs and instruct their behaviour for specific tasks. Specifically, prompts are incorporated into the model's embedding layer to guide its decision-making process. Prompting involves priming an LLM by using prompts to demonstrate examples of the downstream task. Previous studies [20, 65, 74] have demonstrated that the quality of input prompts plays a crucial role in the performance of LLMs, influencing the generated output quality.

A prompt template is often used to generate prompts in a consistent and reproducible way. As an example, in the case of log parsing, one possible prompt template can be formulated as "The log [X] belongs to the log template [Y].", where the objective is to generate the corresponding log template for the input log [X], with [Y] representing the answer. In this paper, we investigate the effectiveness of LLMs in log parsing using hard prompts, which are fixed natural language instructions. We use hard prompts [64] to minimize the variability caused by the prompts, and focus our study on examining the impact of different LLMs and varying training sizes on parsing performance. Below, we discuss the prompts that we used in LLMParsers.

In the design of its prompt, T5 leverages the colon symbol ":" to separate instructions and input data. Since our task shares similarities with the machine translation task, which involves transforming input data to output data, we leverage T5's default prompt structure (i.e., "instruction + input type + output type:") to build our LLMParser$_{T5Small}$ and LLMParser$_{T5Base}$ prompt template as:

> *"Parse the raw log to log template: '{Raw log}'."*
> *"{Log template}"*

We feed the log and log template pairs as examples during training. When parsing logs, we only give instructions about the raw log.

LLaMA-7B and ChatGLM-6B are two large text generation models that have been extensively studied and fine-tuned for various tasks [1, 69, 71]. One notable optimized version of LLaMA is Alpaca [57], which is highly regarded for its performance. The prompt template used by Alpaca has been widely adopted, consistently delivering excellent performance. In this paper, we use Alpaca's prompt template (defined below) to train and generate output for the log parsing task using LLaMA-7B and ChatGLM-6B models.

> *"Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.*
>
> *### Instruction: Parse the input log to the log template.*
> *### Input: '{Raw log}'*
> *### Response: '{Log Template}'*

We give the task description, instruction, input, and response to the LLMs during training. When parsing, we only give the description, instruction, and input, and ask the LLMs to generate the response.

Applying Few-shot Tuning on the LLMs. When the training dataset is the same, the fine-tuning efficiency of a model is directly related to its parameter size, as larger models with more parameters require more computational resources and a longer time to converge [11]. Due to the small parameter size of the Flan-T5-small and Flan-T5-base, fine-tuning can be efficiently done without additional optimization mechanisms. For the larger models, LLaMA-7B and ChatGLM-6B, we used LoRA [28] to accelerate the fine-tuning process. LoRA is an efficient parameter fine-tuning technique, which only trains a very small portion of *ADDITIONAL* parameters while freezing the original parameters of the model. LoRA uses low-rank parameterization and focuses on the most important layers of models, reducing both computational and memory requirements. Consequently, the model converges faster with minimal impact on performance, enabling faster fine-tuning for various tasks, which achieves comparable speed to fine-tuning Flan-t5-base (even with more than 25 times more parameters). Our fine-tuning process takes from several seconds to less than five minutes.

4 EXPERIMENT SETUP AND IMPLEMENTATION

In this section, we discuss our experiment setup to answer our research questions and the implementation details.

Studied Dataset. We conduct our experiment on the log parsing benchmark provided by He et al. [27]. This benchmark contains logs from 16 open-source systems and is widely used to evaluate and compare the accuracy of log parsers [12, 17, 24]. Each system includes 2,000 log messages along with their respective log templates and parameters (the ground truth for evaluating log parsers). The studied systems included in the dataset range from various domains, such as distributed systems, operating systems, mobile systems, supercomputers, server applications, and standalone software. However, recent studies [12, 30] have identified instances of incorrectly labelled log templates in the dataset. As a result, we adopted the corrected benchmark dataset released by Khan et al. [30], following recent research [31, 33].

Environment and Implementation. Our experiments were conducted on a server with an NVIDIA Tesla V100 GPU using CUDA 11.0. For the fine-tuning process of the model, we used a maximum `learning rate` of 5e-4 and use the AdamW [42] optimizer with a linear learning rate decay schedule for optimization. For single system fine-tuning, we uniformly set the `batch size` to 5 and trained 30 `epochs` for LLMParsers. For the cross-system scenario (RQ4), we trained 20 `epochs` and boost the `batch size` to 20 in order to shorten the training time. We used OpenPrompt [14] to fine-tune

LLMParser$_{T5Small}$ and LLMParser$_{T5Base}$. We used LoRA [28] with PEFT v0.3.0 to fine-tune LLMParser$_{LLaMA}$ and LLMParser$_{ChatGLM}$ Note that, fine-tuning with 50 data samples took from only a few seconds to less than five minutes for all the studied LLMs, so the cost of few-shot fine-tuning is small. For log parsing, we set the `temperature` to 0 and `num_beams` to 2 in the generation configuration in order to generate consistent and stable parsed results. Due to the difference in the length of the prompt template, we set the `max_length` to 256 (LLMParser$_{T5Small}$ and LLMParser$_{T5Base}$) and 512 (LLMParser$_{LLaMA}$ and LLMParser$_{ChatGLM}$) as the generation parameter, respectively.

Evaluation Metrics for Log Parsing. Following prior studies [12, 30, 33, 41, 48, 75], we use two metrics to evaluate the effectiveness of LLMs in log parsing: Group Accuracy and Parsing Accuracy.

Group Accuracy (GA): Group Accuracy [75] does not directly evaluate the correctness of the parsed logs. Instead, GA assesses the accuracy of the automatically grouped logs (e.g., GroupID$_{parsed}$ shown in Figure 1). GA is calculated as the ratio between the number of correctly grouped logs and the total number of logs. Specifically, GA first groups logs based on the parsed logs (i.e., generated by a log parser), and compares the resulting groups with grouping results from the ground truth (e.g., GroupID$_{gtruth}$). For instance, the log parsing result in Figure 1 has a GA of 4/7. Once the raw logs are parsed, they are grouped into different groups, as illustrated in Figure 1. We can see that Log 1 and 2 are grouped together, Log 6 and 7 are grouped together, and Log 3, 4, and 5 form a separate group by themselves. The grouping results (GroupID$_{parsed}$) for group 1 to group 3 match the grouping results obtained by using the ground truth log template (GroupID$_{gtruth}$). However, Log 5 to 7 form two groups (GroupID$_{parsed}$ 4 and 5) when using the parsed log template, whereas there is only one group (GroupID$_{gtruth}$ 4) if using the log template from the ground truth. As a result, the GA for this example is 4/7.

Although GA is commonly used, prior studies [12, 30, 41] highlighted its limitations. For instance, even if the logs are perfectly grouped with a GA of 100%, the parsed log template may not fully match the ground truth template due to the misidentified parameters in the parsed templates. As a result, GA cannot show whether or not the logs are parsed correctly. Furthermore, if the logs are not parsed correctly but are still grouped in a cluster (e.g., Log 3 and 4 in Figure 1), GA will still be 100%.

Parsing Accuracy (PA): Parsing Accuracy [41] within the log template must match with the ground truth template. For example, the log parsing result in Figure 1 has a PA of 3/7 because there are missed variables in Log 3, 4, 6, and 7 (highlighted in *red*). Hence, PA is a stricter metric compared to GA and aligns more closely with practical requirements [30, 33, 41]. Prior studies also found that parsing the variables can help downstream log analysis tasks [31, 43, 54], which further shows the importance of PA over GA.

5 EVALUATION

RQ1: What is the accuracy of LLMParsers?

Motivation. A recent work by Le and Zhang [33] proposed using Large Language Models (LLMs) to learn from labelled logs and accurately identify parameters in logs. However, their study only provided initial evidence on the feasibility of using LLMs for log parsing, as they solely utilized one masked-language LLM (RoBERTa-base). Yet, there is a lack of research exploring the impact of different types (e.g., text2text or text generation LLMs) and sizes of LLMs on log parsing accuracy. Hence, in this RQ, we aim to investigate the differences among various types of LLMs and the impact of distinct LLM parameter sizes on log parsing. Such comparisons can assist practitioners in identifying the most effective LLM for log parsing, while also enabling researchers to identify potential future directions on LLM-based log parsing.

Approach. For each system, we fine-tune the four LLMParsers using 50 logs sampled using the sampling algorithm (Algorithm 1). Then, similar to prior studies [33, 37], we use each fine-tuned model to generate the log templates for all 2,000 logs for each of the 16 systems. We compare the grouping and parsing accuracy against state-of-the-art approaches: Drain [24], Logram [12], and LogPPT [33]. We selected the state-of-the-art approaches based on their high accuracy and efficiency [12, 75].

Results. ***LLMParsers have a higher (4.25% to 78.69% higher for LLMParser$_{LLaMA}$) parsing accuracy compared to state-of-the-arts log parsers.*** Table 2 shows both the grouping accuracy (GA) and parsing accuracy (PA) of the state-of-the-art and LLMParsers. We find that, in general, LLMParsers have a higher GA (0.7546~0.8873) and PA (0.9076~0.9587) compared to the traditional log parsers (GA of 0.5513 and 0.8605, and PA of 0.3353 and 0.1718, for Drain and Logram, respectively). LLMs such as LLMParser$_{LLaMA}$ and LLMParser$_{T5Base}$ achieve a GA of 0.88 and a PA of almost 0.96. LogPPT, on the other hand, achieves a high GA (0.9229) and PA (0.9162). Compared to LogPPT which uses a masked language model, we find that our parsers that are based on text2text and text generation models achieve a better PA (except for LLMParser$_{ChatGLM}$). For example, LLMParser$_{LLaMA}$ achieves a PA of 0.9587, which is 4.6% higher than that of LogPPT.

The probabilistic nature of text generation and text2text LLMs may have an impact on the grouping accuracy (GA) of LLMParsers. Nevertheless, the differences in GA between LogPPT and two LLMParsers are not statistically significant. Unlike traditional algorithms, text generation and text2text models have a small probability of generating erratic output that leads to parsing errors [3], which has a larger impact on GA. For example, we observed that when we parsed 2,000 logs from Spark using LLMParser$_{T5Small}$, 1,999 logs were parsed correctly, resulting in a PA of 0.9995. Only one log was parsed incorrectly (one of the dynamic variables was not parsed correctly). However, this log shares the same template with 299 other logs in the system. Although 299/300 logs were correctly parsed, these 300 logs were not grouped together because that one incorrectly parsed log forms a group by itself. As a result, the GA for Spark becomes 0.85 (1,700/2,000). Note that, we set the temperature of the LLMParsers to zero to ensure the consistency in the parsed logs (i.e., given the same input prompt, the output will be the same) [60]. However, logs contain dynamically generated values, so even if two logs have the same template, they are considered two distinct inputs and may have a small probability of resulting in parsed logs with small differences.

Nevertheless, compared to state-of-the-art parsers, the PA of both LLMParser$_{T5Base}$ and LLMParser$_{LLaMA}$ showed a statistically significant increase using paired t-test (p-value<0.05), while GA did not

Table 2: A comparison of the grouping accuracy (GA) and parsing accuracy (PA) for the state-of-the-art (first three columns) and the LLMParser (the last four columns) parsers.

	Drain		**Logram**		**LogPPT**		**LLMParser**$_{T5Small}$		**LLMParser**$_{T5Base}$		**LLMParser**$_{LLaMA}$		**LLMParser**$_{ChatGLM}$	
	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA
Android	0.8305	0.5475	0.7420	0.2780	**0.8845**	0.7665	0.8015	0.9005	0.8680	0.9375	0.8485	**0.9455**	0.8315	0.8395
Apache	**1**	0.6935	0.3125	0.0065	**1**	0.9940	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
BGL	**0.9625**	0.3420	0.5870	0.1245	0.9535	0.9695	0.5040	0.9745	0.4985	0.9770	0.9415	**0.9805**	0.9440	0.9640
Hadoop	0.9475	0.2690	0.4510	0.1125	0.9935	0.8950	**1**	0.9140	0.8055	0.9125	0.9805	**0.9825**	0.6440	0.8375
HDFS	0.9975	0.3545	0.9300	0.0045	**1**	0.9025	**1**	**1**	**1**	**1**	0.9575	0.9880	0.9575	0.9965
HealthApp	0.7800	0.2305	0.2665	0.1120	**1**	0.7885	0.8025	0.9560	0.8085	0.9010	0.8550	**0.9955**	0.5540	0.8190
HPC	0.8870	0.6355	0.9105	0.6430	0.9430	0.9470	0.9685	0.9835	**0.9740**	0.9895	0.9700	**0.9935**	0.9700	0.9855
Linux	0.6900	0.1835	0.3610	0.1240	**0.9335**	0.9485	0.1785	0.8515	0.8190	0.9385	0.5455	0.8385	0.8785	**0.9495**
Mac	**0.7865**	0.2175	0.5680	0.1685	0.7800	0.6725	0.7325	0.6725	0.7750	**0.7090**	0.7390	0.6765	0.6505	0.5205
OpenSSH	0.7890	0.5080	0.6105	0.2980	0.6275	0.9795	0.8870	0.9860	**1**	**1**	0.7095	0.9935	0.5840	0.9450
OpenStack	0.7325	0.0185	0.3255	0	0.9890	0.9065	0.9890	0.9895	**1**	0.9885	0.9785	**0.9960**	0.3125	0.8725
Proxifier	0.5265	0	0.5035	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0.0310	0.9150
Spark	0.9200	0.3595	0.2820	0.2585	**0.9990**	0.9910	0.8500	**0.9995**	0.8500	0.9905	0.9850	0.9850	0.7750	0.9585
Thunderbird	0.9550	0.0465	0.5540	0.0040	0.6790	0.9255	0.9630	0.9595	**0.9705**	**0.9730**	0.6925	0.9675	0.9560	0.9375
Windows	0.9970	0.4620	0.6940	0.1405	0.9910	0.9830	0.7155	0.9885	0.7155	0.9950	**0.9985**	**0.9965**	0.9920	0.9880
Zookeeper	0.9665	0.4970	0.7235	0.4735	0.9935	0.9895	0.9945	**0.9995**	**1**	**0.9995**	0.9945	**0.9995**	0.9935	0.9930
Average	0.8605	0.3353	0.5513	0.1718	**0.9229**	0.9162	0.8367	0.9484	0.8803	0.9570	0.8873	**0.9587**	0.7546	0.9076

Note: The highest values of GA and PA for each system are highlighted in **bold**. The results of Drain and Logram are based on the evaluation conducted by Khan et al. [30] on the corrected log dataset.

Table 3: Grouping accuracy (GA) and parsing accuracy (PA) for logs outside the training dataset.

	LLMParser$_{T5Small}$		**LLMParser**$_{T5Base}$		**LLMParser**$_{LLaMA}$		**LLMParser**$_{ChatGLM}$	
	GA	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**
Android	0.7716	0.8452	0.9058	0.8647	0.8789	0.8680	0.8148	0.8329
Apache	1	1	1	1	1	1	1	1
BGL	0.3798	0.9720	0.3709	0.9752	0.9349	0.9758	0.9375	0.9585
Hadoop	1	0.8517	0.6914	0.8526	0.9768	0.9662	0.3443	0.8501
HDFS	1	1	1	1	0.9605	0.9892	0.9605	0.9979
HealthApp	0.7547	0.9457	0.7622	0.8770	0.8192	0.9944	0.4426	0.7765
HPC	0.9040	0.9584	0.9200	0.9744	0.9154	0.9789	0.9106	0.9610
Linux	0.0474	0.8456	0.9465	0.9569	0.4392	0.7996	0.8834	0.9504
Mac	0.6583	0.5782	0.7106	0.6247	0.6651	0.5857	0.5678	0.4335
OpenSSH	0.8500	0.9848	1	1	0.5129	0.9889	0.3036	0.9082
OpenStack	0.9865	0.9878	1	0.9865	0.9737	0.9955	0.1432	0.8504
Proxifier	1	1	1	1	1	1	0.0137	0.8901
Spark	0.8443	0.9995	0.8443	0.9901	0.9854	0.9854	0.7695	0.9629
Thunderbird	0.9550	0.9498	0.9648	0.9667	0.6638	0.9579	0.9487	0.9303
Windows	0.5271	0.9825	0.5271	0.9925	0.9983	0.9975	0.9882	0.9891
Zookeeper	0.9886	0.9989	1	1	0.9886	0.9989	0.9886	0.9874
Average	0.7917	0.9313	0.8527	0.9413	0.8570	0.9426	0.6886	0.8925

exhibit a statistically significant difference compared to LogPPT (p-value>0.05). Prior studies [12, 30, 33] stated that PA evaluates the practical goal (i.e., correctly parsing logs) of log parsing, which makes PA a better evaluation metric than GA. In short, LLMParsers can better identify the variables of logs and generate correct log templates matching the ground truth.

After we remove the logs used for few-shot fine-tuning during evaluation, the average GA and PA of LLMParser on the remaining logs decrease slightly but are still higher than other baselines. Previous studies on log parsers usually evaluate and compare the effectiveness of log parsers on the entire data set (i.e., all 2,000 logs) [24, 33, 75]. However, such evaluation approaches may include the training data in the evaluation process, causing potential data leakage issues. Hence, we re-evaluated the GA and PA of all four LLMParsers only on the logs by **removing all the logs from the test set that were exactly the same as those in the training set** and showing the results in Table 3. Compared with the baseline result in Table 2 (evaluated using all the logs), all four LLMParsers ' average GA and average PA experienced a slight decrease, with the average PA decreasing by less than 1%. Nevertheless, LLMParser$_{T5Base}$ and LLMParser$_{LLaMA}$ still have higher average PAs (0.9413 for LLMParser$_{T5Base}$ and 0.9426 for LLMParser$_{LLaMA}$) compared to LogPPT (PA is 0.9162, but LogPPT included the training data in the evaluation, having a potential data leakage issue). Our findings show that, after removing the log samples used for training in the evaluation process, LLMParsers can still achieve higher PA than all the baselines.

While more complex LLMParsers (more parameters) generally give better parsing results, simpler models already give promising results. Future studies should consider the trade-off between parsing accuracy and the complexity of LLMs. In general, more complex models give better parsing results, with the exception of LLMParser$_{ChatGLM}$. One reason that LLMParser$_{ChatGLM}$ has a worse accuracy may be that it is a bilingual model and the logs are written in English. Another possible reason is that ChatGLM is engineered as a chatbot, which is fine-tuned to give human-like responses. Between LLMParser$_{T5Small}$ and LLMParser$_{T5Base}$, the GA and PA increased by 5.21% and 0.9%, respectively. We see a further improvement when using LLMParser$_{LLaMA}$ compared to LLMParser$_{T5Base}$, although the improvement is small (0.8% and 0.17% in GA and PA).

However, more complex models may require more resources and time to parse the logs. We randomly selected 100 logs from Spark's log dataset, and measured the average parsing time by repeating the process 20 times. Under the same hardware environment, complex LLMParsers require a longer parsing time. LLMParser$_{T5Small}$ and LLMParser$_{T5Base}$ could parse 100 logs in an average of 1.27 seconds and 4 seconds, respectively, while LLMParser$_{ChatGLM}$ and LLMParser$_{LLaMA}$ require 19.87 and 28.93 seconds, respectively. Therefore, future studies should encompass a trade-off between the accuracy and efficiency of using LLMs on log parsing.

> LLMParsers achieve better PA than state-of-the-art and similar GA compared to LogPPT. Although LLMParser$_{LLaMA}$, which has a larger number of model parameters, achieves the best GA and PA, the difference is small compared to smaller LLMs like LLMParser$_{T5Base}$.

Table 4: Grouping accuracy (GA) and parsing accuracy (PA) for different numbers of training shots.

	LLMParser$_{T5Small}$								**LLMParser**$_{T5Base}$								**LLMParser**$_{LLaMA}$								**LLMParser**$_{ChatGLM}$							
	25 shots		50 shots		75 shots		100 shots		25 shots		50 shots		75 shots		100 shots		25 shots		50 shots		75 shots		100 shots		25 shots		50 shots		75 shots		100 shots	
	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA
Android	0.90	0.88	0.80	0.90	**0.97**	**0.93**	0.86	**0.93**	0.95	0.91	0.87	0.94	**0.98**	0.96	0.96	**0.97**	0.95	0.91	0.85	0.95	**0.98**	0.98	0.97	**0.99**	0.78	0.68	0.83	0.84	**0.93**	**0.96**	0.83	0.93
Apache	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0.99	**1**	**1**	**1**	**1**	**1**	**1**
BGL	**0.56**	0.93	0.50	0.97	0.51	**0.98**	**0.56**	**0.98**	**0.60**	0.94	0.50	0.98	**0.60**	0.98	**0.60**	**0.99**	0.74	0.84	0.94	0.98	0.85	**0.99**	**0.96**	**0.99**	0.84	0.90	0.94	0.96	0.35	0.92	**0.96**	**0.98**
Hadoop	0.80	0.87	**1**	0.91	0.98	0.90	0.97	**0.92**	0.99	0.89	0.81	0.91	**1**	**0.99**	0.97	**0.99**	**0.99**	0.96	0.98	0.98	0.98	**0.99**	0.96	0.98	0.95	0.61	0.64	0.84	**0.96**	**0.97**	0.79	0.83
HDFS	0.70	0.98	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0.92	0.92	0.96	0.99	0.96	**1**	**1**	**1**	0.84	0.94	**0.96**	**1**	0.84	0.99	0.80	0.99
HealthApp	0.67	0.77	**0.80**	**0.96**	**0.80**	0.93	0.67	0.94	0.66	0.85	**0.81**	0.90	**0.81**	**0.99**	**0.81**	**0.99**	0.86	0.99	0.86	**1**	0.87	**1**	**1**	**1**	0.66	0.73	0.55	0.82	0.68	**0.86**	**0.73**	**0.86**
HPC	0.95	0.98	0.97	0.98	0.97	**0.99**	**0.98**	**0.99**	0.97	0.98	0.97	0.99	0.97	0.99	**1**	**1**	**0.99**	0.99	0.97	0.99	0.98	**1**	**0.99**	**1**	0.77	0.91	**0.97**	**0.99**	0.95	**0.99**	0.93	**0.99**
Linux	**0.36**	**0.89**	0.18	0.85	**0.36**	**0.89**	0.18	0.87	**0.94**	**0.97**	0.82	0.94	0.88	0.91	0.36	0.89	**1**	**0.99**	0.55	0.84	0.76	0.98	0.24	0.95	0.42	0.87	**0.88**	0.95	0.48	**0.96**	0.69	0.92
Mac	0.64	0.52	0.73	0.67	0.77	0.76	**0.82**	**0.80**	0.70	0.59	0.78	0.71	0.82	0.73	**0.84**	**0.83**	0.77	0.58	0.74	0.68	0.79	0.78	**0.82**	**0.80**	0.70	0.44	0.65	0.52	0.70	0.53	**0.83**	**0.79**
OpenSSH	0.63	0.92	0.89	0.99	0.64	0.98	**0.94**	**1**	0.33	0.88	**1**	**1**	0.81	**1**	**1**	**1**	0.44	0.99	0.71	0.99	**0.94**	**1**	0.75	**1**	0.58	0.90	0.58	0.95	0.58	0.98	**0.75**	**0.99**
OpenStack	0.96	0.94	**0.99**	0.99	**0.99**	**1**	**0.99**	0.99	0.97	0.95	**1**	0.99	0.99	**1**	**1**	**1**	0.52	0.84	0.98	**1**	0.99	**1**	**1**	**1**	0.31	0.51	0.31	0.87	0.47	0.83	**0.99**	**1**
Proxifier	0.53	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0.52	**1**	**1**	**1**	0.05	**1**	0.53	**1**	0.50	0.95	0.03	0.92	0.53	**1**	**0.98**	**1**
Spark	0.36	0.95	**0.85**	**1**	**0.85**	**1**	0.66	**1**	0.85	**1**	0.85	0.99	**1**	**1**	**1**	**1**	0.81	**1**	0.99	0.99	**1**	0.99	**1**	**1**	0.78	0.79	0.78	0.96	0.79	**0.97**	**0.87**	0.94
Thunderbird	**0.96**	0.93	**0.96**	0.96	0.68	0.97	0.68	**0.98**	0.96	0.92	**0.97**	0.97	0.70	**0.98**	0.70	**0.98**	0.68	0.96	**0.69**	**0.97**	0.68	**0.97**	0.68	**0.97**	0.64	0.57	**0.96**	0.94	0.67	**0.95**	0.67	**0.95**
Windows	0.71	0.98	0.72	0.99	**1**	**1**	**1**	**1**	0.99	0.99	0.72	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0.99**	0.84	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
Zookeeper	**1**	0.99	0.99	**1**	0.99	**1**	**1**	**1**	0.99	**1**	**1**	**1**	0.99	**1**	0.99	**1**	**0.99**	0.99	**0.99**	**1**	**0.99**	**1**	**0.99**	**1**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.97	0.97
Average	0.73	0.91	**0.84**	0.95	**0.84**	**0.96**	0.83	**0.96**	0.87	0.93	0.88	0.96	**0.91**	0.97	0.89	**0.98**	0.82	0.93	**0.89**	0.96	0.86	**0.98**	0.87	**0.98**	0.73	0.79	0.75	0.91	0.75	0.93	**0.86**	**0.95**

Note: The highest values of GA and PA for each LLM for each system are highlighted in **bold**.

RQ2: How does the accuracy of log parsing vary under different shot sizes?

Motivation. In RQ1, we ascertain that LLMs exhibit superior accuracy in log parsing compared to the state-of-the-art approaches. When using LLMs, one thing that researchers and practitioners need to decide is the number of samples for few-shot tuning. Prior research [2, 20] has demonstrated that the efficacy of a model fine-tuned for an individual task is contingent upon the size and diversity of the training dataset. However, manually labelling data can be time-consuming and manual-intensive. Hence, in this RQ, we examine the ramifications of varying training set sizes on the accuracy of log parsing tasks when employing distinct LLMs.

Approach. For each system, we sample 25, 50, 75, and 100 log lines and their corresponding log template using our log sampling algorithm (Algorithm 1). The same sets of logs are used as the fine-tuning dataset for all LLMParsers. We then evaluate the log parsing performance (i.e., GA and PA) of LLMParsers using different sizes of fine-tuning datasets. We vertically compare the accuracy changes of each LLMParser after increasing the training data size. Simultaneously, we also horizontally compare the accuracy differences of different LLMParsers under the same training data size.

To compare, we also apply in-context learning on LLMParser$_{T5Base}$ and LLMParser$_{LLaMA}$ without fine-tuning to investigate the log parsing accuracy. We chose these two LLMs because of their large size and good parsing results shown in RQ1. We use 3 and 15 shots (in-context log parsing demonstrations), respectively for the two LLMs, due to their limits on the size of the input tokens.

Results. ***Increasing the number of shots increases the accuracy of LLMParsers, but the difference is small (e.g., 1–2%) or fluctuates for most LLMParsers beyond 50 shots, except for LLMParser$_{ChatGLM}$.*** Table 4 shows the accuracy of LLMParsers using different numbers of shots. Although there are some improvements in PA and GA when the number of shots increases, the accuracies stabilize after 50 shots. When the shot size is 25, both GA and PA decrease (1% to 15% and 3% to 15%, respectively) for all LLMParsers compared to using 50 shots. We also find that LLMParsers have different sensitivity on the shot sizes. For example, LLMParser$_{T5Base}$ is relatively stable across all shot sizes, while LLMParser$_{ChatGLM}$ has the largest improvement when the shot size increases. When the shot size is increased to 100, the GA decreases for all LLMParsers except for LLMParser$_{ChatGLM}$, but the improvement for PA is barely noticeable. We also find that LLMParser$_{T5Base}$ has better GA and the same PA compared to LLMParser$_{LLaMA}$, the largest LLM among the four studied LLMs. Our finding shows that more complex LLMs may not achieve better PA and GA. For instance, the smallest model LLMParser$_{T5Small}$ achieves comparable results to the second largest model LLMParser$_{ChatGLM}$, and their difference remains trivial even with the increased training shots.

Compared to in-context learning, few-shot tuning achieves better parsing accuracy. Table 5 shows the accuracy for different shots of in-context learning. However, in-context learning yielded worse results. LLMParser$_{LLaMA}$ achieved only an average GA and PA of 30% and 45%, respectively, across 16 systems. LLMParser$_{T5Base}$ also performed poorly, resulting in an average GA and PA of only 22% and 1%. The finding aligns with the discussion provided by a recent study [46] and Flan-T5 developers [10] where few-shot tuning achieves better results than in-context learning.

Different LLMParsers may require different shot sizes to achieve good accuracy in log parsing. For example, LLMParser$_{T5Base}$ already achieves very high accuracy (87% and 93% for GA and PA) when fine-tuned using only 25 shots. However, as the number of shots increases further, the improvement consistently plateaus among all systems, especially when the shot size is over 75. On the other hand, when the shot size is 100, the accuracy of LLMParser$_{ChatGLM}$ becomes comparable to the other parsers. This trend is particularly noticeable in OpenStack and Proxifier, where both GA and PA show substantial improvements from 25 shots to 100 shots (GA increases from 0.31 to 0.99, and PA increases from 0.51 to 1 for OpenStack, and GA increases from 0.50 to 0.98, and PA increases from 0.95 to 1 for Proxifier).

The variation in the accuracy of log parsing across different LLMParsers may be attributed to the nature of LLMs as statistical models. Each model learns to parse logs by identifying distinct

Table 5: Grouping accuracy (GA) and parsing accuracy (PA) for different shots of in-context learning.

	LLMParser$_{LLaMA}$								**LLMParser$_{T5Base}$**									
	5 shots		**10 shots**		**15 shots**		**20 shots**		**1 shots**		**2 shots**		**3 shots**		**4 shots**		**5 shots**	
	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA	GA	PA
Android	0.200	0.093	0.393	0.126	0.430	0.222	0.642	0.318	0.423	0.043	0.427	0.018	0.424	0.002	0.482	0.033	0.530	0.021
Apache	0.984	0.700	0.430	0.460	0.709	0.711	1	0.725	0.291	0	0.291	0	0.566	0	0.291	0	0.291	0
BGL	0.240	0.118	0.258	0.211	0.368	0.418	0.238	0.665	0.126	0	0.085	0.001	0.130	0.002	0.133	0.009	0.154	0.008
Hadoop	0.335	0.244	0.407	0.395	0.180	0.164	0.255	0.321	0.116	0	0.188	0.003	0.213	0.003	0.456	0.004	0.284	0.006
HDFS	0.001	0.071	0.041	0.178	0.011	0.335	0.001	0.241	0.001	0	0.001	0	0.001	0	0.001	0	0.001	0
HealthApp	0.332	0.561	0.429	0.528	0.626	0.756	0.584	0.824	0.120	0.001	0.127	0	0.128	0	0.147	0.019	0.074	0.012
HPC	0.295	0.490	0.155	0.506	0.354	0.592	0.244	0.478	0.635	0.001	0.384	0.001	0.595	0	0.390	0	0.530	0.081
Linux	0.036	0.106	0.201	0.611	0.264	0.616	0.199	0.675	0.129	0.002	0.167	0.009	0.154	0.012	0.170	0.024	0.165	0.012
Mac	0.247	0.129	0.345	0.155	0.414	0.201	0.384	0.180	0.247	0.012	0.355	0.066	0.344	0.008	0.378	0.024	0.389	0.056
OpenSSH	0.002	0.182	0.068	0.382	0.074	0.361	0.028	0.389	0.095	0	0.025	0	0.255	0	0.076	0	0.094	0
OpenStack	0.041	0	0.049	0.054	0.083	0.025	0.070	0.012	0.194	0	0.147	0	0.102	0	0.112	0	0.097	0
Proxifier	0.050	0.627	0.050	0.981	0.050	0.980	0.050	0.963	0	0	0.001	0	0	0	0	0	0.003	0
Spark	0.003	0.312	0.023	0.440	0.401	0.579	0.215	0.494	0.009	0	0.023	0.002	0.048	0.003	0.064	0.001	0.027	0.003
Thunderbird	0.079	0.062	0.118	0.326	0.165	0.398	0.019	0	0.203	0.010	0.178	0.004	0.168	0.004	0.123	0.004	0.101	0.004
Windows	0.162	0.003	0.398	0.567	0.410	0.427	0.181	0.309	0.033	0	0.132	0	0.259	0.009	0.139	0.006	0.011	0
Zookeeper	0.171	0.142	0.017	0.206	0.294	0.380	0.248	0.469	0.171	0.158	0.152	0.001	0.173	0.133	0.042	0.020	0.037	0.027
Average	0.198	0.240	0.211	0.383	0.302	0.448	0.272	0.441	0.174	0.014	0.167	0.006	0.222	0.011	0.187	0.009	0.174	0.014

patterns from the training shots. To determine the best shot size and reduce manual effort on data creation, future studies should investigate the relationship between the characteristics of the LLMs (e.g., architecture and training data) and the needed data to fine-tune the LLMs for log parsing.

> For all LLMParsers except for LLMParser$_{ChatGLM}$, the accuracy improvement in log parsing becomes small or starts to fluctuate when the shot size is over 50. Different LLMs may also require different shot sizes to achieve good parsing results, and more shots do not always give the best results.

RQ3: How is the generalizability of LLMParsers on unseen log templates?

Motivation. In RQ2, we studied the accuracy of LLMParsers using various numbers of shots. We found that although GA and PA improve noticeably when the shot size increases from 25 to 50, the improvement is small or remains almost the same when the shot size is 50 or larger. One hypothesis is that the effectiveness of few-shot tuning is constrained by the diversity of the sampling algorithm when presented with diverse log templates. Logs with different variable values may still have the same log template. By including a single log in the fine-tuning process, it is possible to enhance the parsing accuracy for all other logs that share the same log template. In other words, 50 shots may not capture all the unique log templates, leading to a saturation point where the LLM fails to generalize well to unseen examples. Therefore, our objective is to investigate the extent to which few-shot tuning can generalize to unseen log templates. The finding of this RQ may help future research further improve the accuracy of LLM-based log parsers.

Approach. We use the same set of LLMParsers that are trained using 50 log samples from prior RQs. Specifically, we want to study if a ***log's log template was not included in the training***, would such a log have lower parsing accuracy. We first identify the log templates and the corresponding logs that were not used for few-shot tuning (we call them log$_{unseen}$). Then, we calculate the PA for log$_{unseen}$ and compare it with the PA for log$_{seen}$.

Results. *The PA on log$_{unseen}$ are much lower (e.g., 0.638 for LLMParser$_{LLaMA}$) compared to the PA on log$_{seen}$ (e.g., 0.9539 for LLMParser$_{LLaMA}$). Although only 4.4% of the logs have unseen log templates, they account for 50% of the incorrectly parsed logs.* Table 6 shows the number of total templates, the number of unseen log templates, the number of log$_{unseen}$, and the PA of log$_{unseen}$ and PA of log$_{seen}$. We find that the PA decreases significantly for the log$_{unseen}$ compared to the PA for log$_{seen}$. For example, as shown in Table 6, the PAs for log$_{seen}$ of LLMParser$_{T5Base}$ and LLMParser$_{LLaMA}$ are 0.9506 and 0.9539, whereas their average PAs for the log$_{unseen}$ are 0.6385 and 0.6507, respectively. After some investigation, we find that around 50% of the incorrectly parsed logs among all the 16 systems belong to one of the unseen log templates, and the finding is consistent across all LLMs. Given that, only 4.4% of the logs are log$_{unseen}$ (1,406 out of all 32,000 logs from all the systems), they are disproportionately more likely to be parsed incorrectly. Hence, our finding suggests that log$_{unseen}$ is one of the bottlenecks to further improving parsing accuracy and shed light on future research in log parsers. Nevertheless, we find that LLMParsers still achieve better PA when parsing log$_{unseen}$ compared to traditional state-of-the-art approaches such as Drain and Logram, which have an average PA of 0.3353 and 0.1718, respectively.

We observe a decrease in the performance of LLMParsers when they encounter unseen log templates, indicating their limited ability to generalize. This behaviour in LLMParsers may be attributed to the limitation of the training data during fine-tuning, which primarily focuses on identifying seen log templates. This limitation becomes more apparent when log templates share high similarities. For example, two logs with similar log templates, such as "`(<*>) CMD (<*> <*>)`" and "`(<*>) CMD (run-parts <*>)`", might be mistakenly parsed as the same log template due to their textual similarity, resulting in reduced accuracy. However, if both logs and their templates were provided as training data, LLMParsers could better

Table 6: LLMParsers' PA on $\log_{unseen}$ and PA on $\log_{seen}$ when using 50 log samples. Note that we excluded the systems where all the unique log templates were included in the shots.

				LLMParser$_{T5Small}$		**LLMParser$_{T5Base}$**		**LLMParser$_{LLaMA}$**		**LLMParser$_{ChatGLM}$**	
	Total Template	Unseen Template	Unseen Log	PA-unseen	PA-seen	PA-unseen	PA-seen	PA-unseen	PA-seen	PA-unseen	PA-seen
Android	158	108	224	0.5536	0.9443	0.5893	0.9814	0.7009	0.9764	0.5446	0.8767
BGL	120	70	105	0.7714	0.9858	0.8476	0.9842	0.7714	0.9921	0.6762	0.9799
Hadoop	114	64	64	0.5469	0.9261	0.6719	0.9205	0.5156	0.9979	0.3906	0.8523
HealthApp	75	25	25	0.7200	0.9590	0.7200	0.9033	0.8000	0.9980	0.8400	0.8187
Linux	116	66	66	0.5152	0.8630	0.5909	0.9504	0.8182	0.8392	0.5909	0.9617
Mac	341	291	819	0.3297	0.9102	0.3993	0.9238	0.3761	0.8848	0.3907	0.6105
Thunderbird	149	99	103	0.5146	0.9837	0.6505	0.9905	0.5728	0.9889	0.5631	0.9578
Average	153 (Sum: 1073)	103 (Sum: 723)	201 (Sum: 1406)	0.5645	0.9389	0.6385	0.9506	0.6507	0.9539	0.5709	0.8654

Table 7: Grouping (GA) and parsing (PA) accuracy of using pre-trained LLMParsers (i.e., *pt*), and LLMParsers that is fine-tuned based on the pre-trained LLMParsers (i.e., ft).

	LLMParser$_{LLaMA}$				**LLMParser$_{T5Base}$**			
	GA$_{pt}$	PA$_{pt}$	GA$_{ft}$	PA$_{ft}$	GA$_{pt}$	PA$_{pt}$	GA$_{ft}$	PA$_{ft}$
Android	0.9325	0.7965	0.7655	0.6475	0.6805	0.6575	0.9455	0.9230
Apache	1	0.9940	0.7245	0.7300	1	1	1	1
BGL	0.8250	0.5565	0.4415	0.8230	0.5840	0.9480	0.9575	0.9715
Hadoop	0.9595	0.5030	0.3270	0.4105	0.9560	0.6780	0.9955	0.9810
HDFS	0.1335	0.2000	0.2710	0.7790	0.7470	0.4590	1	1
HealthApp	0.6885	0.6825	0.6250	0.7560	0.7635	0.7560	0.8005	0.9400
HPC	0.9445	0.8745	0.6490	0.8875	0.9150	0.9210	0.9780	0.9895
Linux	0.5440	0.5185	0.1755	0.7350	0.5440	0.5275	0.4870	0.9415
Mac	0.7620	0.4805	0.2065	0.5005	0.7350	0.4335	0.8910	0.7240
OpenSSH	0.2280	0.8710	0.1660	0.9105	0.3900	0.7875	0.5020	0.9745
OpenStack	0.3740	0.4235	0.0385	0.6680	0.2670	0.7950	0.9890	0.9915
Proxifier	0.0010	0	0	0.1730	0.0010	0.0005	1	1
Spark	0.9030	0.9010	0.1195	0.8755	0.7755	0.8980	1	1
Thunderbird	0.6615	0.8430	0.0825	0.1055	0.9465	0.8135	0.6955	0.9550
Windows	0.4015	0.5795	0.2780	0.8620	0.9890	0.9810	1	0.9970
Zookeeper	0.8045	0.8770	0.7470	0.8955	0.9600	0.6775	0.9935	0.9930
Average	0.6352	0.6313	0.3511	0.6724	0.7034	0.7083	0.8897	0.9613

differentiate between them and achieve higher accuracy. Future research may consider improving the generalization of LLMParsers by proposing sampling algorithms that can select a more diverse sampled set of logs and templates during fine-tuning.

> LLMParsers achieves bad results on $\log_{unseen}$ compared with results on $\log_{seen}$. The unseen logs, which only make up 4.4% of all logs, form 50% of the incorrectly parsed logs. Some types of variables may not be identified even if they appear in the training dataset.

RQ4: Can pre-trained LLMParsers help improve parsing accuracy?

Motivation. In the previous RQs, we investigated the log parsing accuracy when fine-tuning the LLMs using the logs from the same system. However, one major advantage of LLM is its ability to generalize on new datasets [29, 61, 73]. Therefore, in this RQ, we study if using a LLMParser that is pre-trained using logs from other systems can further improve log parsing results.

Approach. We consider LLMParser$_{T5Base}$ and LLMParser$_{LLaMA}$ in this RQ because of their high log parsing accuracy and representative model size. For every system, we pre-train the LLM using 15 *other* systems by following the same fine-tuning process as done before. For the first part of the evaluation, we apply the pre-trained LLMParsers directly to parse the logs of the target system (the system of which the logs are not used for pre-training). Then, we further fine-tune the pre-trained LLMParsers using 25 log samples from the target system and evaluate the accuracy of the parsed logs.

Results. ***LLMParsers pre-trained using logs from other systems achieve considerably lower GA and PA compared to LLMParsers that use few-shot tuning*** Table 7 shows the GA and PA of using the pre-trained LLMParser$_{LLaMA}$ and LLMParser$_{T5Base}$. By pre-training using only the logs from 15 other systems, LLMParser$_{LLaMA}$ achieves a PA and GA of 0.6352 and 0.6313, and LLMParser$_{T5Base}$ achieves a PA and GA of 0.7034 and 0.7083. Compared to few-shot tuning using logs from the same system, the PA and GA for the pre-trained LLMParsers decrease considerably. Interestingly, despite having more parameters, LLMParser$_{LLaMA}$ achieves lower GA and PA compared to LLMParser$_{T5Base}$. The reason may be that, compared to LLMParser$_{LLaMA}$, LLMParser$_{T5Base}$ converges faster and is better at avoiding underfitting during fine-tuning due to its smaller parameter size. Nevertheless, we find that the GA and PA of the pre-trained LLMParsers are still comparable to that of the traditional log parsers (i.e., Drain and Logram), which further shows the potential of using LLMs for log parsing. *Our finding shows that logs from different systems may have different characteristics, so using only logs from other systems gives worse GA and PA.*

Further tuning using logs from the target system shows opposite results in different LLMParsers. The GA and PA improved considerably in LLMParser$_{T5Base}$ but the GA of LLMParser$_{LLaMA}$ decreased by almost 55% compared to using only the pre-trained LLMParser$_{LLaMA}$. We further fine-tune the pre-trained LLMParsers using 25 log samples from the target system. We find that few-shot tuning the pre-trained LLMParsers improved the GA and PA of LLMParser$_{T5Base}$ to 0.8897 and 0.9613, respectively (Table 7). The improvement in GA and PA is large compared to using only the pre-trained LLMParser$_{T5Base}$ (from a GA and PA of around 0.7 to 0.8897 and 0.9613 in GA and PA, respectively). The GA and PA are also comparable to using 100 log samples from the target system to fine-tune LLMParser$_{T5Base}$, which has a GA and PA of 0.89 and 0.98, as shown in Table 2. This enables us to minimize the time cost and manual effort required for labelling training data and also reduces the fine-tuning time. However, we see an opposite result in LLMParser$_{LLaMA}$, where further tuning using 25 log samples from the target system results in worse GA (decreased from 0.6352 to 0.3511, a 55% decrease) and similar PA (increased from 0.6313 to 0.6724, a 6% increase).

Some studies [9, 18, 47] have shown that the amount of data and model parameter size required to achieve optimal performance on certain tasks are not directly correlated across models with different parameter sizes and architectures. Sometimes, having too much or too little amount of data can result in model overfitting and poor model performance. The pre-trained LLMParser$_{LLaMA}$ may also need more data to fine-tune due to its larger size. Future research should explore the use of different quantities or types of log data for pre-training models based on our study in order to build more generalized, high-accuracy log parsing models that require fewer fine-tuning samples and better align with practical needs.

> Although pre-trained LLMParsers achieves worse results compared to few-shot tuning, they achieve similar results compared to the prior state-of-the-art. Further tuning the pre-trained LLMParsers shows opposite results in two LLMs, where the result became worse for LLMParser$_{LLaMA}$ but better for LLMParser$_{T5Base}$.

6 DISCUSSION

LLMParsers achieve promising results with higher parsing accuracy than state-of-the-arts and comparable grouping accuracy with LogPPT [33]. However, we also find some limitations and potential improvements in LLM-based log parsing. In this section, we summarize our observations and highlight future research directions.

LLMParsers face challenges in parsing some specific logs. More advanced or log-tailored LLMs are needed to further improve LLM-based log parsers. LLMParsers takes a few training samples as input and then learns how to parse logs. However, as shown in Table 6, even though LLMParsers achieve high accuracy (above 90% PA), there are still some logs that were not parsed correctly. Through manual investigation, we find that LLMParsers face challenges in recognizing specific data types (e.g., datetime) as variables. For instance, LLMParser$_{LLaMA}$ fails to parse the log template "`connection from <*> at <*>`" correctly. Instead, the logs are parsed as "`connection from <*> at Mon Jul 25 23:24:09 2005`" without recognizing the timestamp value as the second variable. Due to having a limited number of samples, the LLMs are not able to learn how to parse some variables. One potential solution is, similar to RQ4, to use log data from other systems to help pre-train an LLM-based log parser so that the parsers can generalize and identify more variables. The other potential solution is to use a more complex LLM with more parameters. However, future studies should consider the trade-off between more complex LLMs and higher fine-tuning and inference costs.

Since more complex LLMs may not always give better results, future studies are needed to find the balance between accuracy and efficiency. We find that simpler models, such as T5, can already achieve promising log parsing results and larger models may not give better results. There may be significant cost implications when using larger or even commercial LLMs. More importantly, as we found, larger LLMs also need more inference time to parse logs. Since logs are often large in volume, having an efficient parser is important. Future research should explore the right balance in model size and parsing efficiency.

Future research should explore the most effective sampling algorithms for identifying training log samples. In RQ3, we find that LLMParsers have worse parsing results on unseen log templates and there are cases where increasing shot sizes result in worse parsing results. For example, when two very similar yet different log templates (e.g., one template has one more variable) are included in the training, the models may get confused when parsing the corresponding logs. Hence, future research should explore the optimal sampling strategy that can maximize diversity while also considering the characteristics of the logs and the corresponding templates (e.g., should certain types of logs be sampled more to distinguish the similar templates).

Fine-tuning with samples from the target system is demonstrated as the most effective way for log parsing. In RQ2, we find that in-context learning shows bad performance on log parsing. Also, in-context learning is not as effective as fine-tuning due to the token limitation and efficiency concerns. In RQ3 and RQ4, we also find that LLMParsers face challenges in generalizing parsing unseen logs. We found that fine-tuning LLM with samples from the target system gives the best result. Future studies should consider validating the findings on more complex LLMs (e.g., LLaMA 70B) and see if such models have better generalizability on log parsing.

7 THREATS TO VALIDITY

External validity. Similar to prior work [30, 32, 33], we train and validate LLMParser using logs and log templates from public datasets that are commonly used in log-related research, However, recent research [30] has indicated the dataset might contain data errors. To mitigate this potential issue, we leverage the corrected dataset [30] to reduce such a threat. For a fair comparison, we also compare our results with the results from the state-of-the-art based on the corrected data [30, 33]. The log format may also affect our result, but the used datasets cover logs from various systems with different formats. Future studies are needed to evaluate LLM-based parsers on logs from other systems. **Internal validity.** While LLMParser outperforms other state-of-the-art approaches, our primary focus in this research is on the exploration of the performance of LLMs in log parsing tasks. Our research does not cover all LLMs and training data sizes. Future studies may explore the optimal solution for LLM-based log parser, enabling further advancements in this domain. **Construct validity.** Our approach requires pairs of logs and their log templates. Hence, the sampling process may affect the parsing result. To mitigate the issue, we apply an unsupervised data sampling algorithm that does not require any knowledge of the ground truth. Future studies are needed to explore the effect of such sampling algorithms on the parsing results.

8 CONCLUSION

In this study, we explore the potential of leveraging LLMs for log parsing. We propose LLMParser, a generative LLM-based log parser, to overcome the limitations of existing log parsers. LLMParser leverages few-shot tuning to learn from a limited set of training logs , which were sampled using a clustering sampling algorithm. Our evaluation shows that LLMParsers achieve high accuracy, outperforming state-of-the-arts log parsers. We then evaluate LLMParsers under different pre-training settings. Our results show that, compared to in-context learning, few-shot tuning achieves higher parsing accuracy and requires less inference time. Furthermore, our findings suggest that different LLMParser models may require different numbers of training samples to achieve optimal performance. Instead of increasing the training shot sizes, future studies should investigate how training log diversity and coverage affect log parser accuracy. Our exploratory study leverages generative LLMs for log parsing and delivers comprehensive evaluations in various settings (architecture, shot sizes, and pre-training), which provides empirical evidence for future research.

REFERENCES

[1] Yuvanesh Anand, Zach Nussbaum, Brandon Duderstadt, Benjamin Schmidt, and Andriy Mulyar. 2023. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. *GitHub* (2023).

[2] Alexandre Bailly, Corentin Blanc, Élie Francis, Thierry Guillotin, Fadi Jamal, Béchara Wakim, and Pascal Roy. 2022. Effects of dataset size and interactions on the prediction performance of logistic regression and deep learning models. *Computer Methods and Programs in Biomedicine* 213 (2022), 106504. https://doi.org/10.1016/j.cmpb.2021.106504

[3] Mark Belford, Brian Mac Namee, and Derek Greene. 2018. Stability of topic modeling via matrix factorization. *Expert Systems with Applications* 91 (2018), 159–169. https://doi.org/10.1016/j.eswa.2017.08.047

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Jinfu Chen, Weiyi Shang, Ahmed E Hassan, Yong Wang, and Jiangbin Lin. 2019. An experience report of generating load tests using log-recovered workloads at varying granularities of user behaviour. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, IEEE, 669–681.

[6] Song Chen and Hai Liao. 2022. Bert-log: Anomaly detection for system logs based on pre-trained language model. *Applied Artificial Intelligence* 36, 1 (2022), 2145642.

[7] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2023. Empowering Practical Root Cause Analysis by Large Language Models for Cloud Incidents. *arXiv preprint arXiv:2305.15778* (2023).

[8] Yizong Cheng. 1995. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17, 8 (1995), 790–799. https://doi.org/10.1109/34.400568

[9] Yew Ken Chia, Pengfei Hong, Lidong Bing, and Soujanya Poria. 2023. INSTRUCTEVAL: Towards Holistic Evaluation of Instruction-Tuned Large Language Models. *arXiv preprint arXiv:2306.04757* (2023).

[10] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. https://doi.org/10.48550/ARXIV.2210.11416

[11] Kenneth Ward Church, Zeyu Chen, and Yanjun Ma. 2021. Emerging trends: A gentle introduction to fine-tuning. *Natural Language Engineering* 27, 6 (2021), 763–778.

[12] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient Log Parsing Using n n-Gram Dictionaries. *IEEE Transactions on Software Engineering* 48, 3 (2020), 879–892.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Hai-Tao Zheng, and Maosong Sun. 2021. Openprompt: An open-source framework for prompt-learning. *arXiv preprint arXiv:2111.01998* (2021).

[15] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305* (2020).

[16] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey for in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[17] Min Du and Feifei Li. 2019. Spell: Online Streaming Parsing of Large Unstructured System Logs. *IEEE Transactions on Knowledge and Data Engineering* 31, 11 (2019), 2213–2227. https://doi.org/10.1109/TKDE.2018.2875442

[18] Ronen Eldan and Yuanzhi Li. 2023. TinyStories: How Small Can Language Models Be and Still Speak Coherent English? *arXiv preprint arXiv:2305.07759* (2023).

[19] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In *2009 Ninth IEEE International Conference on Data Mining*. 149–158. https://doi.org/10.1109/ICDM.2009.60

[20] Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making Pre-trained Language Models Better Few-shot Learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 3816–3830. https://doi.org/10.18653/v1/2021.acl-long.295

[21] Zhiqiang Gong, Ping Zhong, and Weidong Hu. 2019. Diversity in Machine Learning. *IEEE Access* 7 (2019), 64323–64350. https://doi.org/10.1109/ACCESS.2019.2917620

[22] Pinjia He, Zhuangbin Chen, Shilin He, and Michael R Lyu. 2018. Characterizing the natural language descriptions in software logging statements. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 178–189.

[23] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. 2017. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing* 15, 6 (2017), 931–944.

[24] 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)*. 33–40. https://doi.org/10.1109/ICWS.2017.13

[25] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37.

[26] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 207–218.

[27] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448* (2020).

[28] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[29] Hong Jin Kang, Tegawendé F. Bissyandé, and David Lo. 2019. Assessing the Generalizability of Code2vec Token Embeddings. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1–12. https://doi.org/10.1109/ASE.2019.00011

[30] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2022. Guidelines for Assessing the Accuracy of Log Message Template Identification Techniques. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1095–1106. https://doi.org/10.1145/3510003.3510101

[31] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2023. Impact of Log Parsing on Log-based Anomaly Detection. *arXiv preprint arXiv:2305.15897* (2023).

[32] Van-Hoang Le and Hongyu Zhang. 2023. An Evaluation of Log Parsing with ChatGPT. *arXiv preprint arXiv:2306.01590* (2023).

[33] Van-Hoang Le and Hongyu Zhang. 2023. Log Parsing with Prompt-based Few-shot Learning. In *45th International Conference on Software Engineering: Software Engineering in Practice (ICSE)*.

[34] Yukyung Lee, Jina Kim, and Pilsung Kang. 2021. LAnoBERT: System log anomaly detection based on BERT masked language model. *arXiv preprint arXiv:2111.09564* (2021).

[35] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing code explanations created by students and large language models. *arXiv preprint arXiv:2304.03938* (2023).

[36] Heng Li, Tse-Hsun Chen, Weiyi Shang, and Ahmed E Hassan. 2018. Studying software logging using topic models. *Empirical Software Engineering* 23 (2018), 2655–2694.

[37] Zhenhao Li, Chuan Luo, Tse-Hsun Chen, Weiyi Shang, Shilin He, Qingwei Lin, and Dongmei Zhang. 2023. Did We Miss Something Important? Studying and Exploring Variable-Aware Log Abstraction. *arXiv preprint arXiv:2304.11391* (2023).

[38] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems* 35 (2022), 1950–1965.

[39] Jinyang Liu, Junjie Huang, Yintong Huo, Zhihan Jiang, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Minzhi Yan, and Michael R Lyu. 2023. Scalable and Adaptive Log-based Anomaly Detection with Expert in the Loop. *arXiv preprint arXiv:2306.05032* (2023).

[40] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[41] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, et al. 2022. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022*. 1893–1901.

[42] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bkg6RiCqY7

[43] Siyang Lu, BingBing Rao, Xiang Wei, Byungchul Tak, Long Wang, and Liqiang Wang. 2017. Log-based abnormal task detection and root cause analysis for spark. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 389–396.

[44] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786* (2021).

[45] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837* (2022).

[46] Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation. *arXiv preprint arXiv:2305.16938* (2023).

[47] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2021. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment* 2021, 12 (2021), 124003.

[48] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2021. Self-supervised log parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV*. Springer, 122–138.

[49] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[50] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[51] Md Saidur Rahaman, MM Tahmid Ahsan, Nishath Anjum, Harold Jan R Terano, and Md Mizanur Rahman. 2023. From ChatGPT-3 to GPT-4: a significant advancement in ai-driven NLP tools. *Journal of Engineering and Emerging Technologies* 2, 1 (2023), 1–11.

[52] Google Research. 2023. The Flan Collection: Advancing open source methods for instruction tuning – Google Research Blog. https://ai.googleblog.com/2023/02/the-flan-collection-advancing-open.html. (Accessed on 07/16/2023).

[53] Keiichi Shima. 2016. Length matters: Clustering system log messages using length of words. *arXiv preprint arXiv:1611.03213* (2016).

[54] Donghwan Shin, Zanis Ali Khan, Domenico Bianculli, and Lionel Briand. 2021. A Theoretical Framework for Understanding the Relationship Between Log Parsing and Anomaly Detection. In *Runtime Verification: 21st International Conference, RV 2021, Virtual Event, October 11–14, 2021, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 277–287. https://doi.org/10.1007/978-3-030-88494-9_16

[55] Lili Song, Ying Wang, Yinhe Han, Xin Zhao, Bosheng Liu, and Xiaowei Li. 2016. C-Brain: A Deep Learning Accelerator That Tames the Diversity of CNNs through Adaptive Data-Level Parallelization. In *Proceedings of the 53rd Annual Design Automation Conference* (Austin, Texas) *(DAC '16)*. Association for Computing Machinery, New York, NY, USA, Article 123, 6 pages. https://doi.org/10.1145/2897937.2897995

[56] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating System Events from Raw Textual Logs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* (Glasgow, Scotland, UK) *(CIKM '11)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2063576.2063690

[57] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[58] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[59] R. Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764)*. 119–126. https://doi.org/10.1109/IPOM.2003.1251233

[60] Chi Wang, Susan Xueqing Liu, and Ahmed H Awadallah. 2023. Cost-Effective Hyperparameter Optimization for Large Language Model Generation Inference. *arXiv preprint arXiv:2303.04673* (2023).

[61] Peifeng Wang, Filip Ilievski, Muhao Chen, and Xiang Ren. 2021. Do language models perform generalizable commonsense inference? *arXiv preprint arXiv:2106.11533* (2021).

[62] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[63] Zehao Wang, Haoxiang Zhang, Tse-Hsun Chen, and Shaowei Wang. 2021. Would you like a quick peek? providing logging support to monitor data processing in big data applications. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 516–526.

[64] Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2023. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *arXiv preprint arXiv:2302.03668* (2023).

[65] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).

[66] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *arXiv preprint arXiv:2303.07839* (2023).

[67] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. 2010. Sherlog: error diagnosis by connecting clues from run-time logs. In *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*. 143–154.

[68] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130B: An Open Bilingual Pre-trained Model. In *The Eleventh International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=-Aw0rrrPUF

[69] zero_nlp contributors. 2023. "A large collection of large language model-powered solutions in Chinese". https://github.com/yuanzhoulvpi2017/zero_nlp/tree/main/simple_thu_chatglm6b. (Accessed on 06/25/2023).

[70] Bo Zhang, Hongyu Zhang, Pablo Moscato, and Aozhong Zhang. 2020. Anomaly Detection via Mining Numerical Workflow Relations from Logs. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*. 195–204. https://doi.org/10.1109/SRDS51746.2020.00027

[71] Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199* (2023).

[72] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[73] Xin Zhou, DongGyun Han, and David Lo. 2021. Assessing Generalizability of CodeBERT. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 425–436. https://doi.org/10.1109/ICSME52107.2021.00044

[74] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).

[75] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.

[76] Chen Zhuge and Risto Vaarandi. 2017. Efficient Event Log Mining with LogClusterC. In *2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids)*. 261–266. https://doi.org/10.1109/BigDataSecurity.2017.26