Leveraging Prompt Engineering on Large Language Model for Semantic Log Parsing

Hansae Ju, Scott Uk-Jin Lee
Major in Bio Artificial Intelligence, Dept. of Applied Artificial Intelligence, Hanyang University, Ansan, Korea
sparky@hanyang.ac,kr
Major in Bio Artificial Intelligence, Dept. of Computer Science & Engineering, Hanyang University, Ansan, Korea
scottlee@hanyang.ac.kr

Abstract—**Log parsing serves as a pivotal initial stage in log analysis, identifying event templates from logs, which are unstructured (or semi-structured) texts containing crucial runtime data from software systems. Conventionally, research in this field has focused on discerning static description (i.e., template) and dynamic parameters (i.e., variables) within a log. Nonetheless, recent studies have confirmed that understanding the meaning of dynamic variables provides valuable information for log interpretation and analysis, which has consequently sparked the initiation of 'semantic log parsing'. In this paper, we introduced LoGPT, a novel mechanism that perceives log parsing as a code generation task, subsequently identifying templates with semantic variables. LoGPT utilizes a few-shot prompt engineering approach with the Large Language Model (LLM) to facilitate semantic log parsing, thereby significantly reducing the need for labor-intensive manual labeling and resources. Our findings shed light on a fresh perspective in the domain of semantic log parsing.**

Keywords-component; log parsing, prompt engineering

I. INTRODUCTION

Logs are an essential part of software systems, acting as repositories for critical runtime information, captured in unstructured or semi-structured text formats. The sheer volume of logs generated in real-world systems – amounting to thousands of terabytes per day in some instances – necessitates efficient and intelligent automated log analysis methodologies [1]. A prerequisite for numerous log analysis techniques, such as anomaly detection, failure detection and software bug localization, is the conversion of unstructured logs into a structured format, a process known as log parsing.

Log parsing primarily involves identifying templates from unstructured log messages. A log message typically consists of an event template (static part) and variables (dynamic part), corresponding to natural language descriptions and variable values in logging statements. Consider a Java logging statement in Fig. 1 (a). The log message generated by (a) is a combination of structured text (created by the logging library) and unstructured text (consisting of a developer-written description and dynamic variables). Parsing certain elements such as timestamps, PIDs, levels, and classes is straightforward due to their structured format, facilitated by regular expressions and pattern matching techniques. However, parsing the message portion of the log, composed of templates and variables, poses significant challenges.

Understanding the developers' logging intentions can often be deduced from the code's natural language descriptions and variable selections. However, in most systems, operators do not have access to logging statements, especially in multi-component systems where

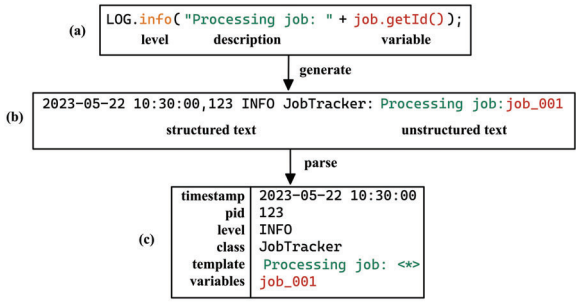

Figure 1. Log example in Java. (a) logging statement (b) log message (c) log parsing.

logs from various components are collected. There have been some studies on matching logging statements with corresponding logs, but in the rea-world scenario, they are not practical for absence of source code [2].

Moreover, most existing research has focused solely on identifying event templates. However, dynamic variables provide crucial information for understanding and analyzing logs. A recent study manually analyzed benchmark data to identify 10 categories of variables and discovered through empirical research and surveys that variable categories record vital information for log analysis [3]. Furthermore, SemParser [4] applied the concept of semantically parsed parameters with templates for downstream log analysis, demonstrating a performance improvement. Research into parsing information from variables is limited, and existing studies have manually labeled categories or concepts, using the structure of LSTM to predict information at the token level. However, manual labeling is labor-intensive and potentially in accurate. Additionally, the oracle templates

used in benchmark datasets for manual labeling cannot guarantee accuracy or reliability [5] .

In this paper, we propose LoGPT (Log + GPT), a Large Language Model (LLM)-based novel variable-aware log parser leveraging prompt engineering techniques. LoGPT exploits the natural language and code comprehension abilities of LLM to transform the log parsing task into generating logging statements from log messages. LoGPT can identify log templates with semantically meaningful variable names in zero-shot manner, without additional training on corresponding source code or log messages. By using Python's f-string syntax, we instructed the LLM to generate a simple code consisting of variable declarations and an f-string line that could reproduce the user-provided log message. LoGPT obtains local variables by executing the code generated by the LLM, performs template identification and regular expression conversion through string parsing, and verifies whether it can reproduce the provided log message. We evaluate the framework on four datasets from the LogPAI [6] and report several interesting findings. To the best of our knowledge, this is the first attempt to leverage LLM for log parsing and to treat log parsing task as a form of code generation.

II. Related Work

A. Research on Log Parsing

There is a wealth of prior research on log parsing. Frequent patter mining techniques, such as Logram [7], exploit the fact that static words appear more frequently than dynamic parameters. Similarly, Yu *et al* [8] utilized the self-attention scores of BERT in each token, exploiting the fact that static words have a high frequency. Clustering algorithms identify templates are based on the idea that similar logs are likely to belong to the same template [9]. Furthermore, some studies apply heuristic-based parsing rules with other techniques such as fixed depth parse trees in Drain [10]. However, these previous studies merely distinguish between static and dynamic parts, without considering the semantics of the variables.

B. Semnatic Vairable aware Log Parsing

Recently, attempts have been made to conduct research on semantic log parsing, recognizing the importance of dynamic variables. VALB [3] analyzed the LogPAI dataset manually to study the characteristics of dynamic variables and identified 10 categories. They proposed an LSTM-based framework to identify each word's category in the log using a Named Entity Recognition scheme. The authors argued that the dynamic values recorded in the log provide valuable information for understanding and analyzing the log.

SemParser [4] manually labeled the concept of each dynamic value or instance in the log and proposed a Semantic Miner based on LSTM to predict the concept at the word level. They also suggested a Joint Parser that utilized a domain knowledge repository to identify instances not only within the log but also between logs based on these (concept, instance) pairs.

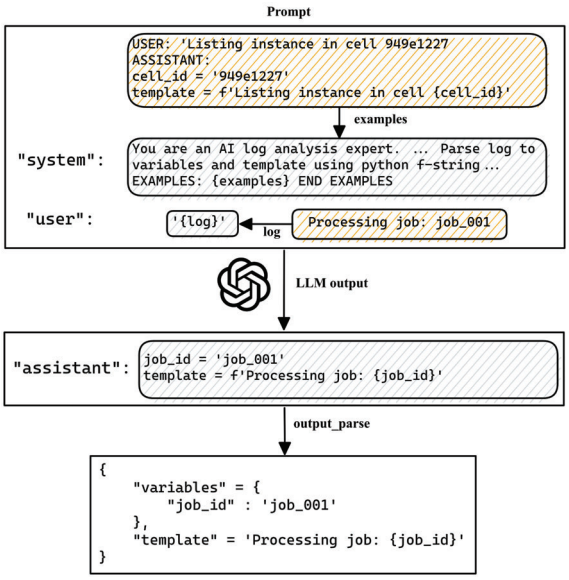

Figure 2. Overview of semantic log parsing by prompting LLM.

However, a significant amount of manual effort goes into labeling the meanings of dynamic variables, and there are limitations in fully identifying the subtly different meanings of variables depending on the template. This paper employs prompt engineering that leverages LLM's natural language and programming language knowledge to minimize manual effort and perform template identification through a code generation task, thereby extracting various variable semantics.

III. Methodology

A. Overview of LoGPT

Our framework, LoGPT, harnesses the power of LLMs through prompt engineering and integrates into a pipeline for efficient parsing of all logs. This process is illustrated in Fig. 2, which provides a walkthrough of our log parsing methodology using a concrete example. We utilized OpenAI's gpt-3.5-turbo model, an LLM that allows for controlled behavior through 'system' message and takes inputs via 'user' messages to produce outputs in the form of 'assistant' message.

Drawing from the latest advancements in prompt engineering, we instructed the model to perform the role of an 'AI log analysis expert'. This expert generates Python code, including variable declarations and f-string statements, that can recreate the user's log. Furthermore, we leveraged few-shot prompting technique with some demonstrations [11]. The demonstrations included a total of four types of examples derived from domain knowledges. It is noteworthy that these only four examples were utilized across all datasets.

The four types of examples were: (1) logs without variables, (2) logs with a single variable, (3) logs with multiple variables that may have the same name, including the creation of different variable names, and (4) logs that could recognize filenames or paths as a single variable. These examples can be customized to achieve higher performance or to apply specific naming

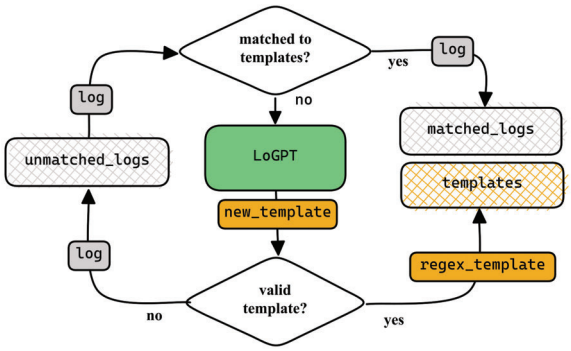

Figure 3. The pipeline of LoGPT

conventions. Not only do these examples help the model better understand the intention of the instructions, but they also enforce to emulate a consistent output format.

The log parsing process of LoGPT, exemplified by our running example, is delineated as follows in Fig. 2. First, when the log *`Processing job: job_001`* is input into the model, the LLM model outputs a Python code comprising a variable declaration line `job_id = job_001` and an f-string variable declaration line `template = f'processing job: {job_id}'`. While this generated code significantly differs from the original Java logging statement in Fig. 1, it excels in segregating descriptions and variables. Then the output code is executed within the framework (using `exec()`) to obtain an actual Python dictionary of variables and their values. The template is extracted by stripping the f-string and the results are exported by LoGPT in a JSON format. In this way, our framework allows logs to be parsed into templates and variables with semantic names.

B. *The LoGPT pipeline*

The computational cost and time associated with running LLMs generally correlate with the length of the input prompt and output message. In the case of LoGPT, this becomes especially relevant when many examples are used or when parsing exceedingly long log messages. To make the use of such LLMs efficient in the real-world, where immense volumes of log data are generated, we have devised a pipeline as depicted in Fig. 3.

The central concept involves transforming parsed templates by LoGPT into regular expression templates (regex templates) once they pass a validity check. If a regex template can successfully match any unmatched logs including input log, we consider the template is valid. Logs are fed into LoGPT only if they are not matched by regex templates. In the best case, this approach only requires LLM computations equal to the number of ground-truth templates.

Additionally, if not all logs are matched to a template within a given epoch, we enhance the diversity of the LLM for computations on unmatched logs in subsequent epochs. In our model, the first epoch runs with a temperature of 0.2 (for deterministic outcome), while from the next epoch onwards, 0.8 is applied to introduce more randomness.

IV. Evaluation and Findings

To validate our framework's capability to identify template along with semantic variable names, it is necessary not only to evaluate its log parsing ability, uch like traditional log parsers, but aloso to assess the semantic appropriateness of the generated variable names. The evaluation of the appropriateness of variable names in log templates has not been previously addressed; hense, we carried out a qualitative evaluation of meaningful results following our log parsing experiment.

For the evaluation dataset, we selected LogHub[6], a public log parsing benchmark widely used in previous log parser research. To minimize manual effort, we conducted our evaluation on log data from four projects, namely Apache, Proxifier, HDFS, and OpenSSH, chosen from a toatal of 16 projects, based on the criterion of having the smallest number of oracle templates. We assesed both the aforementioned apspects for these chosen projects.

A. *Experimental reuslt*

Our experimental results demonstrated that the framework could parse all logs into one or more templates across all datasets. However, due to the Zero-shot manner, there were parsing results that did not exactly match the oracle templates of the benchmark datasets. Note that these benchmark datasets are constructed by manual validations. Therefore, we decided to analyze these mismatches in parsing results and notable outcomes.

In the case of the Apache, LoGPT parsed templates identical to the oracle templates. However, only about 33 to 50% of templates were identical to the oracle templates in the other three datasets, and either generated a broader variety of templates or generated fewer templates of a wider scope. Our investigation revealed that the generation of broader templates was due to LoGPT interpreting static parts in the oracle templates as variables, leading the the consolidation of similar templates. The creation of more templates occurred when the model judeges variable parts as static, resulting in the division of one template into several.

These issues could be ameliorated by more elaborate prompt engineering, such as clustering similar log messages to include them in the prompt as input examples, or by adding more specialized demonstrations. It also seems effective to gradually enhance the LLM output results by applying strategies that match logs using templates generated in the LoGPT pipeline. Furthermore,

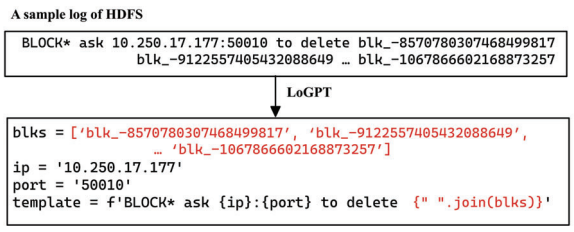

Figure 4. A log parsing example in HDFS

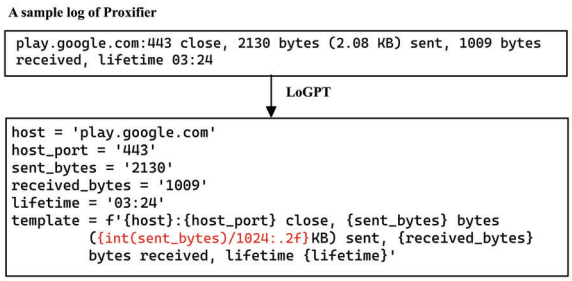

Figure 5. A log parsing example in Proxifier

improvements could potentially be achieved by fine-tuniing the model with log-related data.

B. Findings

In this section, we present novel findings from the results of performing semantic log parsing as a code generation task through our framework. These are outcomes that could not have emerged from traditional research identifying the categories or concepts of variables. The HDFS sample log in Fig. 4 represents a log where `blk_*` appears 100 times in the log message. LoGPT generated a string list variable called `blks` from the 100 `blk` instances and represented it in the template using the `' '.join(blks)` method. This template, given its generation from a list rather than a general variable declaration like `blk_ids`, not only simplifies template understanding for practitioners but also promises to be more effective for downstream log analysis tasks. Such results can be attributed to LoGPT's LLM utilizing Python knowledge to interpret long input values as list variables and applying the join method to convert them into strings suitable for f-string templates.

Another noteworthy example is the log sample from the Proxifier dataset in Fig. 5. This log contains units such as bytes and KB, and impressively, LoGPT declared a `sent_bytes` variable, understood the unit KB, and used `int(sent_bytes)/1024` instead of creating a new variable, thereby demonstrating its capability to understand numerical occurrences frequently found in log data, as well as units. Moreover, it even used f-string formatting syntax to format up to two decimal places according to the log message format.

These discoveries highlight that embedding deeper semantics in templates is possible not just by assigning semantics based on variable names or categories but by using program expressions. Such findings enhance the potential to further leverage the task of source code generation in future semantic log parsing research.

V. Conclusion and Future Work

In this paper, we proposed a novel semantic log parser, LoGPT. Our key idea is that considering log parsing task as code generation task. To demonstrate this, we leveraged prompt engineering techniques to instruct LLMs to generate python logging statements by given log message. Our qualitative experimental results show that LLMs can be used to parse log message into static template and semantic variables, which not limited to appropriate names, but also include program expressions. We expect to these findings will open up new horizons for the utilization of code generation tasks in semantic log parsing research.

In the future, we will evaluate LoGPT in large dataset and other LLM backends. Furthermore, we are going to refine the framework to output more stable and scalable to provide more actionable and feasible intelligent log analysis techniques to practitioners.

Acknowledgment

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2022-00155885, Artificial Intelligence Convergence Innovation Human Resources Development (Hanyang University ERICA)) and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF-2023R1A2C1006390).

References

[1] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, "Logzip: Extracting Hidden Structures via Iterative Clustering for Log Compression," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA: IEEE, Nov. 2019, pp. 863–873.

[2] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, Big Sky Montana USA: ACM, Oct. 2009, pp. 117–132.

[3] Z. Li *et al.*, "Did We Miss Something Important? Studying and Exploring Variable-Aware Log Abstraction." in ICSE '23. arXiv, Apr. 22, 2023. Accessed: May 01, 2023.

[4] Y. Huo, Y. Su, C. Lee, and M. R. Lyu, "SemParser: A Semantic Parser for Log Analysis." in ICSE '23. arXiv, Feb. 05, 2023. Accessed: Apr. 21, 2023.

[5] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in *Proceedings of the 44th International Conference on Software Engineering*, in ICSE '22. New York, NY, USA: Association for Computing Machinery, 5 2022, pp. 1095–1106.

[6] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics." arXiv, Aug. 14, 2020. Accessed: May 04, 2023.

[7] H. Dai, H. Li, C.-S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient Log Parsing Using nn-Gram Dictionaries," *IEEE Trans. Softw. Eng.*, vol. 48, no. 3, pp. 879–892, Mar. 2022

[8] S. Yu, N. Chen, Y. Wu, and W. Dou, "Self-supervised log parsing using semantic contribution difference," *J. Syst. Softw.*, vol. 200, p. 111646, Jun. 2023.

[9] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "LogMine: Fast Pattern Recognition for Log Analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, Indianapolis Indiana USA: ACM, Oct. 2016, pp. 1573–1582.

[10] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in *2017 IEEE International Conference on Web Services (ICWS)*, Jun. 2017, pp. 33–40.

[11] S. Min *et al.*, "Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?" arXiv, Oct. 20, 2022.