Optimized Log Parsing: Template Refinement and Semantic Generalization

Nafid Enan

A Thesis Submitted to the Faculty of Graduate Studies

In Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science

Graduate Program in Electrical Engineering and Computer Science

York University

Toronto, Ontario

April 2026

Abstract

Logs are a primary source of runtime information in modern software systems and are fundamental to tasks such as anomaly detection, failure diagnosis, performance monitoring, and reliability engineering. Log parsing, the process of converting unstructured log messages into structured templates, serves as the foundation of these downstream analyses. To assist in this, many novel approaches have been proposed.

This dissertation contains three sections: 1) an empirical study of existing log parsers, 2) SynLog+, and 3) GeLT.

In the empirical study, we evaluate syntax-based, semantic-based, and hybrid log parses. The observations from the study indicates two issues. Firstly, there is a gap between the parsing accuracy of the syntax- and semantic-based parsers that hybrid parsers aim to bridge. However, semantic-based and hybrid parsers leverage PLMs or LLMs, requiring heavy runtime costs. Secondly, semantic-based parsers lack in generalization ability over unseen logs.

With SynLog+, we propose a wrapper to existing log parsers that acts as a template refinement module aiming to improve the parsing accuracy without added runtime cost.

With GeLT, we aim to re-evaluate the existing generalizable log parser, Log3T, and identify its design flaws and propose an optimized generalizable semantic-based log parser.

Acknowledgements

I am deeply grateful to my supervisor, Dr. Gias Uddin, whose mentorship and encouragement have been central to the completion of this thesis. His guidance and steady support shaped both the direction and quality of this research. Throughout the process, he consistently pushed me to think more critically, question assumptions, and refine my ideas with rigor. Beyond his technical insight, he cultivated an atmosphere that valued independence, curiosity, and scholarly growth. Working under his supervision has been both a privilege and a formative experience in my development as a researcher.

I am also grateful to the members of my supervisory committee for their insightful evaluations and valuable recommendations, which significantly strengthened the quality and clarity of this dissertation.

Finally, I owe my sincerest gratitude to my family for their unwavering love, encouragement, and support. Their belief in me has been a constant source of motivation throughout my graduate studies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Software logs provide critical insights into runtime behavior, errors, and failures, but their volume and complexity make manual analysis impractical. To assist in log analysis, researchers have proposed a number of approaches to automate tasks such as anomaly detection [1, 2, 3], failure prediction [4, 5], and root cause analysis [6, 7]. In automating log analysis, log parsing often serves as the first step.

The aim of log parsing is to convert the raw log data into log templates by identifying the constants and variables. The logging statement in the source code contains both constants and dynamic variables. Log parsing converts the log content into a log template which delineates between the constants and the variables in the logging statement. For instance, in Figure 1.1, the log content "*Reading broadcast variable 11 took 15 ms*" is parsed into the log template "*Reading broadcast variable <*> took <*> ms*" with template parameters *11* and *15* where the parameters are the values of the variables in the log template.

Since it is a fundamental step for downstream log analysis tasks, extensive research has been done in log parsing. Initially, the research focused on syntax-based log parsers, which utilized frequency, similarity, and heuristics to cluster similar log messages and identify the

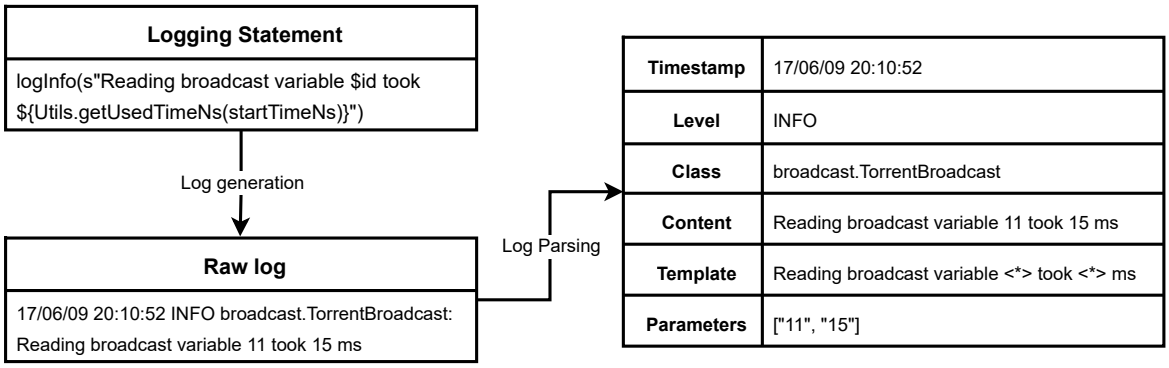


Figure 1.1: An example of log parsing from Spark dataset

templates of the log groups [8, 9, 10]. However, with the advances in deep learning and language model techniques in recent years, the research in log parsing has shifted to utilize semantic-based approaches [11, 12], which formulate the task of log parsing as a token classification problem.

Besides the techniques used to assess the underlying token structures, log parsers can also be categorized based on their architectural design. Traditional approaches often adopt a grouping-based or a classification-based architecture, treating log parsing as a unified task of either grouping similar messages or classifying tokens as constants or variables. However, due to the high cost of runtime for token classification models, recent log parsers adopt a hybrid architecture, integrating log grouping and token classification tasks with two separate modules [13]. This reduces the inference cost of semantic models and improves grouping accuracy by incorporating global information.

Due to the diversity of existing log parsers, it is necessary to evaluate and analyze them not only by their parsing techniques but also by their architectural design. Zhu et al. [14] conducted the first comparative study of 13 log parsers using a benchmark of 16 datasets, each containing 2,000 log messages (Loghub-2k). More recently, Jiang et al. [15] proposed an evaluation study of 15 log parsers on a benchmark of 14 datasets each containing an average of 200K logs. However, the aforementioned studies neglected LLM-based log parsers such as LLMParser and LILAC [16, 13]. Moreover, they only evaluated the log parsing techniques.

It is equally important to evaluate the log parsers based on their underlying architectures.

In this chapter, we present a comprehensive empirical study comparing log parsing techniques of state-of-the-art log parsers, focusing on both syntax-based and semantic-based approaches, along with grouping-based, classification-based, and hybrid log parsing architectures. Our results show that syntax-based log parsing techniques achieve superior grouping accuracy and runtime efficiency, while semantic-based log parsing techniques lead in parsing accuracy but incur significantly higher computational cost. Specifically, AEL [17] and Drain [10] achieve a grouping accuracy (GA) of 0.8 and a parsing accuracy (PA) of 0.4, while LogPPT [11] and LLMParser [16] achieve GA of 0.7 and PA of 0.8. LILAC achieves the best grouping and parsing accuracy while retaining better efficiency than other semantic-based log parsers by utilizing hybrid parsing architecture. The experiments demonstrate that compared to grouping- and classification-based parsing, hybrid parsing obtains better accuracy across all four metrics, with average improvement of 27% in GA, 6% in PA, 49% in FGA, and 81% in FTA. Moreover, semantic-based parsers show a lack of generalization over unseen log data. These findings underscore the practical trade-offs between efficiency and accuracy in syntax- and semantic-based log parsing techniques and highlight the effectiveness of hybrid architectures.

Based on the findings of the empirical study, we introduce SynLog+, a template refinement pipeline. SynLog+ improves the parsing accuracy of syntax-based log parsers by 236% on average while retaining their efficiency. The improvement in PA for semantic-based parsers is low, an average of 20.6%, which is because the parsing accuracies of semantic-based log parsers are already close to their grouping accuracies. The average improvement in GA, PA, FGA, and FTA achieved by employing SynLog+ is 17%, 157%, 181%, and 553%.

Additionally, motivated by the finding of the empirical study that semantic-based log parsers lack in generalization ability, we conduct a case study of existing generalizable log parser, Log3T, which uses test-time training. The study reveals design flaws, which guided us to propose GeLT, an optimized approach toward generalizable semantic-based log parsing

with test-time training. GeLT improves the F1 GeLT improves F1 grouping accuracy (FGA) in 3 and F1 template accuracy (FTA) in 2 datasets out of 16 benchmark datasets, compared to Log3T's improvement of FGA in 2 datasets and FTA in 0 datasets while dropping in FGA and FTA in 5 and 4 datasets respectively.

1.2 Research Hypothesis

This thesis is guided by the following hypothesis.

> A systematic and comprehensive evaluation of existing log parsing approaches can reveal previously underexplored opportunities for improving the efficiency, accuracy, and generalizability of log parsing techniques across both syntax-based and semantic-based paradigms.

The key expectations from the thesis are as follows.

1. A comprehensive empirical analysis of existing log parsers will provide deeper insights into the strengths, limitations, and trade-offs of different categories of log parsing technique
2. The insights derived from this empirical study will help identify underexplored design choices and optimization opportunities for improving log parsing methodologies
3. A critical study of the existing generalizable semantic-based log parser will inform the design of a more optimized generalizable framework for semantic-based log parsing

1.3 Thesis Overview

Chapter 2: Empirical study. We conduct a comprehensive empirical study of existing log parsing techniques (syntax-based and semantic-based), architectures (grouping-based, classification-based, and hybrid), and the characteristics of the benchmark datasets. The

results from the study reveals key insights toward improving log parsing in both syntax- and semantic-based techniques, the former by improving the template identification strategy and the latter by improving the generalization ability.

Chapter 3: SynLog+. Based on the insights from the empirical study, we propose a template refinement pipeline, which improves the parsing accuracy of existing log parsers, specially in cases where template identification is inaccurate with correct log grouping without requiring high computation time.

Chapter 4: GeLT. We present a case study of the generalizable log parser, Log3T, and diagnose critical flaws in its design choices. Based on these insights, we propose an optimized generalizable log parser with test-time training.

1.4 Thesis Contribution

The contributions of this thesis are as follows.

1. **A comprehensive empirical study of log parsing.** We conduct a systematic and comprehensive empirical study of existing log parsing approaches and architectures and the benchmark datasets. Our study reveals key strengths and limitations of each category of log parsing techniques and architectures and provides insight into the characteristics of the variables in the benchmark datasets.

2. **SynLog+, a template refinement pipeline to improve parsing accuracy.** Based on the insights of the empirical study, we propose a regex-based approach to improving the parsing accuracy by better template identification. Overall, SynLog+ improves the accuracy of the benchmark log parsers by 157% in parsing accuracy (PA) and 553% in F1 template accuracy (FTA).

3. **GeLT, an optimized approach to generalizable log parsing with test-time training.** We first conduct a case study of existing generalizable log parser and diagnose critical flaws in its design. We then propose an optimized generalizable log

parsing with test-time training and evaluate its effectiveness. GeLT improves the F1 GeLT improves F1 grouping accuracy (FGA) in 3 and F1 template accuracy (FTA) in 2 datasets out of 16 benchmark datasets, compared to Log3T's improvement of FGA in 2 datasets and FTA in 0 datasets while dropping in FGA and FTA in 5 and 4 datasets respectively.

1.5 Thesis Organization

We organized this thesis as follows.

- **Chapter 2** presents the empirical study, which provides insights into existing log parsing techniques, architectures. and datasets.
- **Chapter 3** presents SynLog+, which proposes a template refinement pipeline for improving parsing accuracy, guided by the insights from the empirical study. The chapter contains the design rationale of SynLog+ and its workflow and provides an evaluation of its effectiveness.
- **Chapter 4** presents GeLT, which diagnoses issues in existing generalizable log parsing technique with test-time training and proposes an optimized generalizable log parser with test-time training. The chapter consists of a case study of Log3T, the existing generalizable log parser, and the methodology of GeLT along with its evaluation.
- **Chapter 5** concludes this thesis and discusses future research directions.

Chapter 2

Empirical Study of Existing Log Parsing Approaches

2.1 Introduction

Since it is a fundamental step for downstream log analysis tasks, extensive research has been done in log parsing. Initially, the research focused on syntax-based log parsers, which utilized frequency, similarity, and heuristics to cluster similar log messages and identify the templates of the log groups [8, 9, 10]. However, with the advances in deep learning and language model techniques in recent years, the research in log parsing has shifted to utilize semantic-based approaches [11, 12], which formulate the task of log parsing as a token classification problem.

Besides the techniques used to assess the underlying token structures, log parsers can also be categorized based on their architectural design. Traditional approaches often adopt a grouping-based or a classification-based architecture, treating log parsing as a unified task of either grouping similar messages or classifying tokens as constants or variables. However, due to the high cost of runtime for token classification models, recent log parsers adopt a hybrid architecture, integrating log grouping and token classification tasks with two separate modules [13, 18, 19]. This reduces the inference cost of semantic models and improves

grouping accuracy by incorporating global information.

Due to the diversity of existing log parsers, it is necessary to evaluate and analyze them not only by their parsing techniques but also by their architectural design. Zhu et al. [14] conducted the first comparative study of 13 log parsers using a benchmark of 16 datasets, each containing 2,000 log messages (Loghub-2k). More recently, Jiang et al. [15] proposed an evaluation study of 15 log parsers on a benchmark of 14 datasets each containing an average of 200K logs. However, the aforementioned studies neglected LLM-based log parsers such as LLMParser and LILAC [16, 13]. Moreover, they only evaluated the log parsing techniques. It is equally important to evaluate the log parsers based according to their underlying architectures.

In this chapter, we present a comprehensive empirical study comparing log parsing techniques of state-of-the-art log parsers, focusing on both syntax-based and semantic-based approaches, along with grouping-based, classification-based, and hybrid log parsing architectures. Our results show that syntax-based log parsing techniques achieve superior grouping accuracy and runtime efficiency, while semantic-based log parsing techniques lead in parsing accuracy but incur significantly higher computational cost. Specifically, AEL [17] and Drain [10] achieve a grouping accuracy (GA) of 0.8 and a parsing accuracy (PA) of 0.4, while LogPPT [11] and LLMParser [16] achieve GA of 0.7 and PA of 0.8. LILAC achieves the best grouping and parsing accuracy while retaining better efficiency than other semantic-based log parsers by utilizing hybrid parsing architecture. The experiments demonstrate that compared to grouping- and classification-based parsing, hybrid parsing obtains better accuracy across all four metrics, with average improvement of 27% in GA, 6% in PA, 49% in FGA, and 81% in FTA. These findings underscore the practical trade-offs between efficiency and accuracy in syntax- and semantic-based log parsing techniques and highlight the effectiveness of hybrid architectures.

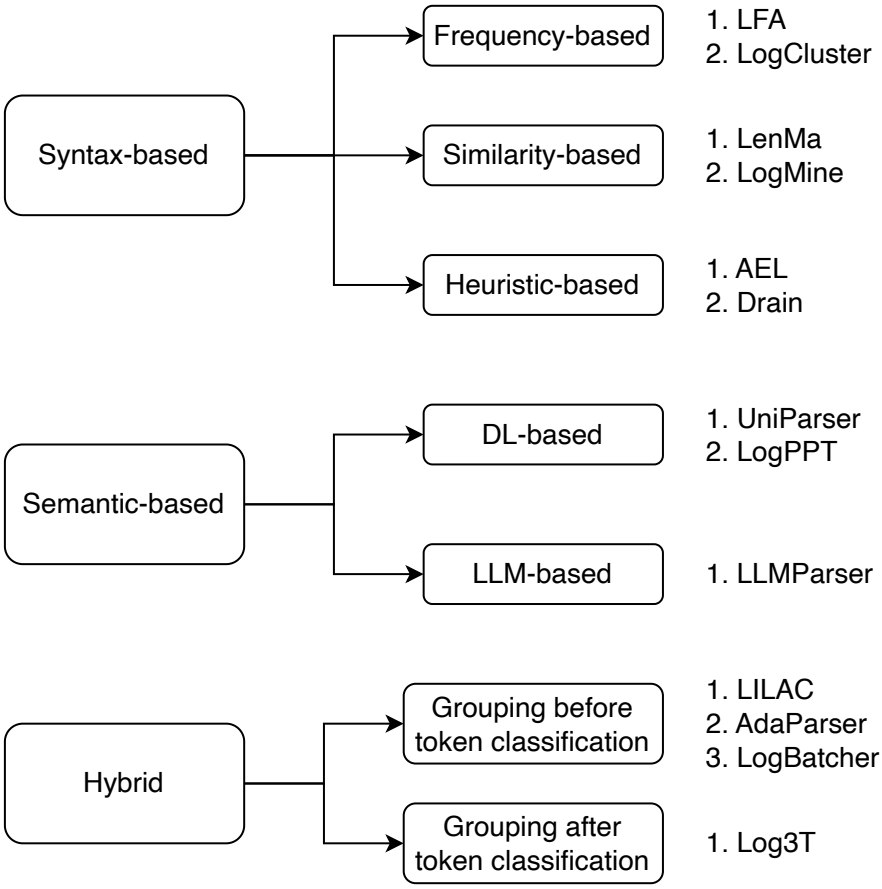


Figure 2.1: Hierarchical categorization of log parsers

2.2 Background and Related Work

Log parsers can be categorized in two ways: 1) based on techniques to analyze the tokens, and 2) based on architectures. Figure 2.1 presents a hierarchical categorization of the existing log parsers.

2.2.1 Log Parsing Techniques

Based on how the log parsers assess the underlying structure of the tokens in a log message, existing log parsing techniques can be categorized into two categories: (1) syntax-based and (2) semantic-based. Figure 2.2 shows the overview of syntax- and semantic-based log parsing

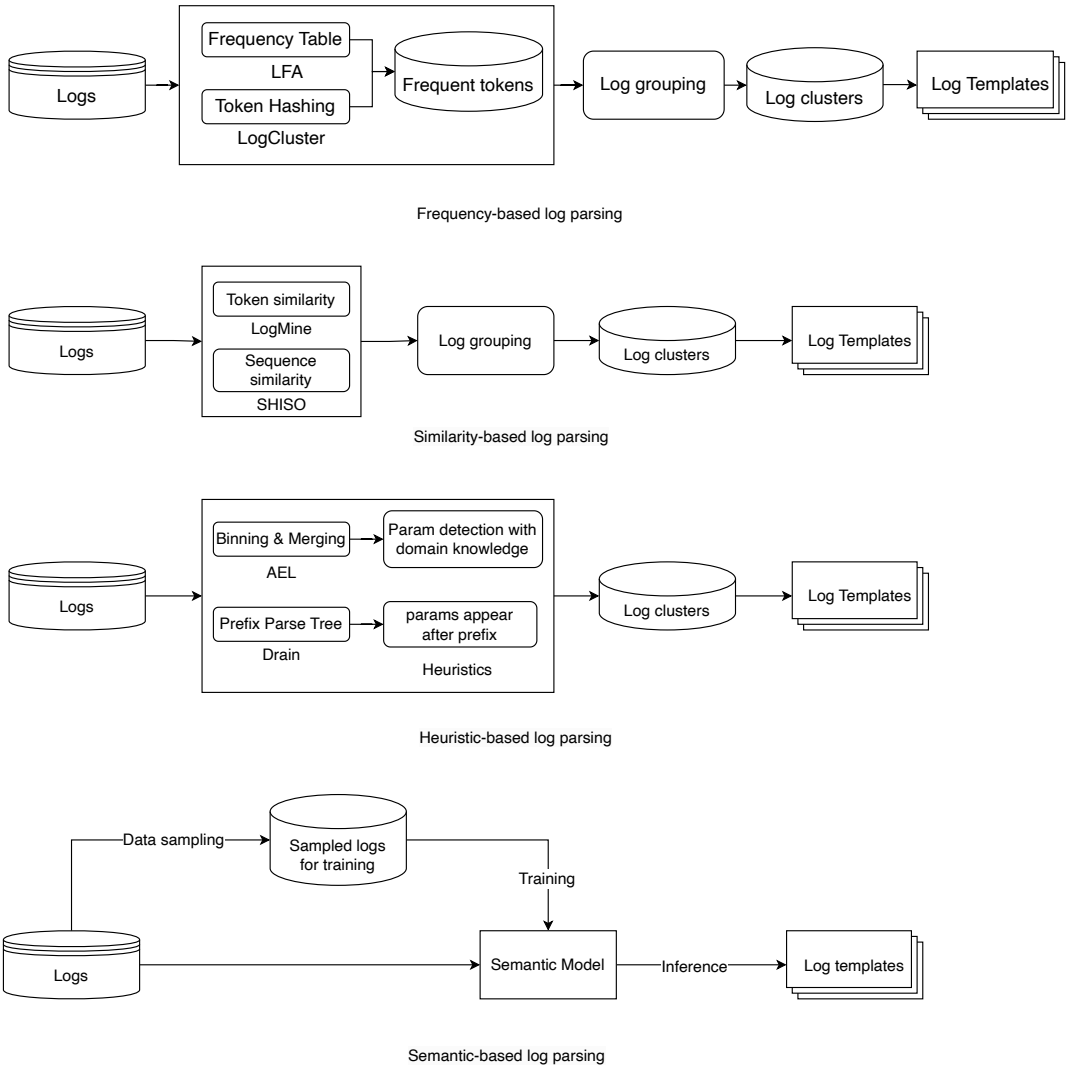


Figure 2.2: Overview of syntax-based and semantic-based log parsers

techniques.

Syntax-based log parsers

Formulating the log parsing task as a clustering problem, Syntax-based log parsers rely on statistical features like token frequency, position, and patterns to group similar logs together and identify the template of the log groups. Syntax-based methods can be divided into three types: (a) frequency-based, (b) similarity-based, and (c) heuristic-based.

Frequency-based log parsers operate under the assumption that constant tokens in log messages occur more frequently than variable tokens. Based on this assumption, they employ pattern mining techniques to identify frequent tokens. Three prominent examples of this category are LFA, LogCluster, and Logram [8, 20, 21].

Similarity-based log parsers assume that log messages generated from the same event template share similar sequences of tokens. Logs are clustered based on the similarity between each pair of logs. LenMa [22], LogMine [23], and SHISO [9] are three examples of this category.

Heuristic-based log parsers utilize rule-driven methods that rely on assumptions about the log messages. For example, Drain relies on the heuristic assumption that the first tokens of logs are constants. Three examples of this category are AEL, Spell, and Drain [17, 24, 10].

Semantic-based log parsers

Instead of the syntactical structure of the tokens, semantic-based log parsers leverage deep learning models (DNNs) or pretrained language models (PLMs) or recent Large Language Models (LLMs) to learn the contextual meaning of tokens. They identify the log template by using the model to classify the tokens as either constants or variables.

DNN-based log parsers aim to automatically learn the structural patterns and semantic relationships within log messages using deep neural network models. Examples of DNN-based log parsers are UniParser [25], which uses a bi-directional LSTM and CNN model as a token classifier.

PLM-based log parsers aim to automatically learn the structural patterns and semantic relationships within log messages using pre-trained language models. Examples of PLM-based log parsers are Log3T [26] and LogPPT [11]. Both use BERT-based models for token classification.

LLM-based log parsers use LLM for token classification or template generation, either training it with fine-tuning or prompting it with in-context learning. Examples of LLM-

based log parsers are LLMParser [16] and LILAC [13].

2.2.2 Log Parsing Architectures

Based on the underlying architecture used to identify the log templates, log parsers can be categorized into three categories: 1) Grouping-based, 2) Classification based, and 2) Hybrid.

Grouping-based architectures

Syntax-based parsers formulate log parsing as a clustering problem, where log messages are first grouped according to structural similarity and templates are subsequently inferred from the grouped messages [22, 17]. In these architectures, template identification is performed by identifying token positions that vary across messages within a log group and marking them as variables. Because template inference occurs after grouping, these architectures primarily rely on global information present in the log groups. For instance, Drain follows this formulation by first organizing log messages into groups and then deriving templates from shared token patterns. Prior works have demonstrated that relying only on global information leads to inaccurate template identification [27, 25].

Classification-based architectures

In contrast, semantic-based parsers formulate log parsing as a token classification problem. These methods predict whether individual tokens correspond to constants or variables using learned semantic representations. Templates are then constructed from the predicted token labels for each log message independently. This formulation enables the use of contextual information at the token level and has been shown to improve template identification accuracy in comparison to architectures that rely solely on global information. Parsers such as LogPPT [11] adopt this formulation by leveraging neural models to classify tokens within individual log messages.

Hybrid architectures

Recent work has explored combining these two formulations by incorporating both token-level predictions and group-level structural information during template identification (e.g., Log3T [26], LILAC [13], AdaParser [18]). In these architecture, token-level predictions may be refined using information derived from groups of structurally similar log messages, or grouping mechanisms may be used to regularize independently inferred templates. By integrating both global and token-level signals, such hybrid architectures aim to balance the efficiency of grouping-based methods with the template identification accuracy of token classification–based methods.

2.2.3 Other Related Work

Several studies have systematically evaluated log parsing approaches in addition to proposing new parsing techniques. Zhu et al. [14] introduced the Loghub-2k benchmark, consisting of 16 datasets with 2,000 log messages each, and evaluated a range of existing parsers. While Loghub-2k became a widely adopted benchmark, subsequent analysis revealed inaccuracies in its ground truth. Khan et al. [28] addressed these issues by applying handcrafted correction rules, such as merging dot-separated variables and consolidating consecutive variables into single placeholders. Although this effort improved annotation quality, it did not address the benchmark's limited scale.

To overcome this limitation, Jiang et al. [15] released Loghub-2, which substantially expanded both dataset volume and diversity, providing 14 datasets ranging from 21,000 to 16 million log messages. Their comprehensive evaluation of syntax-based and semantic-based parsers demonstrated that earlier conclusions drawn from Loghub-2k were constrained by insufficient log volume and template diversity. In particular, larger-scale evaluation exposed weaknesses in template identification that were not apparent in smaller datasets. However, this study did not consider LLM-based log parsers, which have recently emerged as a prominent direction in log analysis research.

Recent work has begun to explore the use of large language models for log parsing. Ma et al. [16] evaluated LLMs of varying parameter sizes and showed that larger models do not necessarily yield better parsing accuracy. While their pipeline demonstrated the feasibility of LLM-driven parsing, it suffered from significant efficiency limitations. Jiang et al. addressed this concern in LILAC [13] by introducing a caching mechanism to reduce redundant LLM invocations and improve grouping efficiency. Although this hybrid design improved runtime and grouping performance, it retained the high computational cost associated with large language models, limiting its practicality for large-scale deployments.

More recently, Le et al. [29] demonstrated that semantic-based parsers built upon small pre-trained language models (PLMs) can achieve performance comparable to state-of-the-art LLM-based parsers while being more efficient and cost-effective. This line of work narrows the gap between PLM-based and LLM-based semantic parsers. Nevertheless, the broader architectural divide between syntax-based and semantic-based approaches remains insufficiently explored. Syntax-based parsers offer strong efficiency and scalability, whereas semantic-based methods improve template identification accuracy but incur higher computational overhead.

Despite advances in benchmarking and model design, prior studies have primarily focused on improving either semantic modelling capability or runtime optimization within semantic frameworks. There remains a lack of systematic investigation into how the complementary strengths of syntax-based grouping and semantic-informed template identification can be reconciled without incurring the cost of large neural models. In particular, it remains unclear whether the template identification weaknesses of syntax-based parsers can be addressed through lightweight, structure-aware refinement mechanisms, rather than through increasingly complex or costly semantic models.

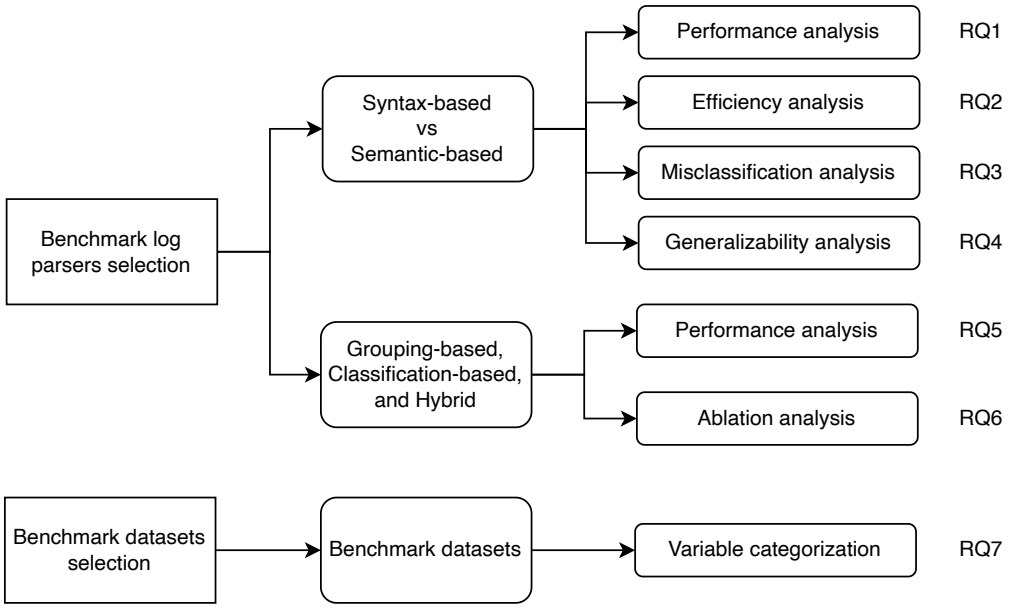


Figure 2.3: Empirical study

2.3 Methodology

2.3.1 Research Questions

To understand the strengths of current log parsing tools, we conduct an empirical study comparing syntax-based and semantic-based log parsers, as well as grouping-based, classification-based, and hybrid parsing architectures. Figure 2.3 presents an overview of the empirical study.

Category 1. Log Parsing Approaches.

RQ1.1. How do syntax-based log parsers perform compared to semantic-based log parsers?

RQ1.2. How efficient are syntax-based log parsers compared to semantic-based log parsers in terms of runtime?

RQ1.3. How much do invariant variables in a log group impact the loss of accuracy in syntax-based compared to semantic-based parsers?

RQ1.4. How do semantic-based log parsers perform on unseen log data?

Category 2. Log Parsing Architectures.

RQ1.5. How does hybrid log parsing compare with classification-based parsing?

RQ1.6. What is the impact of the order of the log grouping and the token classification modules in hybrid log parsing on the accuracy?

Category 3. Benchmark Datasets Characteristics.

RQ1.7. How prevalent are structurally regular variable types within the Loghub-2k benchmark datasets?

2.3.2 Study Setup

Evaluation Metrics

Following Jiang et al.[15], we use two categories of metrics, namely message-level and template-level metrics. Message-level metrics assess the parsing of each log message individually, thus favouring templates with high volume of log messages. Template-level metrics assess the parsing of the log templates, therefore eliminating the bias of uneven templates in terms of log volume. For both categories of metrics, we choose two metrics: one for assessing the clustering ability and another for the token classification ability.

Message-Level Metrics. Following existing studies, we utilize two popular message-level metrics, GA and PA.

Grouping Accuracy (GA) Proposed by Zhu et al. [30], GA assesses the ability to correctly group log messages belonging to the same template. GA is defined as the ratio of correctly grouped log messages over the total number of log messages, where a log message is regarded as correctly grouped if and only if the set of log messages with the same template corresponds to the same set of log messages in the ground truth.

$$GA = \frac{\#\ correctly\ grouped\ log\ messages}{\#\ log\ messages}$$

Parsing Accuracy (PA) Proposed by Dai et al.[21], PA assesses the ability to correctly identify the constant parts and variable parts of each log message. It is defined as the ratio of correctly parsed log messages over the total number of log messages, where a log message is correctly parsed if and only if all constant and variable tokens are correctly identified.

$$PA = \frac{\#\ correctly\ parsed\ log\ messages}{\#\ log\ messages}$$

Template-Level Metrics. Following the recent study by Jiang et al. [15], we use two template-level metrics, FGA and FTA.

F1-score of Group Accuracy (FGA) Proposed by Jiang et al. [15], FGA focuses on the proportion of correctly grouped templates rather than log messages. FGA is the harmonic mean of PGA (Precision of Group Accuracy) and RGA (Recall of Group Accuracy). PGA is defined as the ratio of correctly grouped log templates over the total number of log templates generated by the log parser, and RGA is defined as the ratio of correctly grouped log templates over the total number of log templates in the ground truth. A log template is considered correctly grouped if and only if the set of log messages belonging to this template corresponds to the same set of log messages in the ground truth.

$$FGA = \frac{\#\ correctly\ grouped\ log\ templates}{\#\ log\ templates}$$

F1-score of Template Accuracy (FTA) Proposed by Khan et al. [28], FTA focuses on the proportion of correctly parsed templates rather than log messages. FTA is the harmonic mean of PTA (Precision of Template Accuracy) and RTA (Recall of Template Accuracy). PTA is defined as the ratio of correctly parsed log templates over the total number of log templates generated by the log parser, whereas RTA is defined as the ratio of correctly parsed log templates over the total number of log templates in the ground truth. A log template is considered correctly parsed if and only if all the log messages belonging to this template are correctly grouped and all the tokens of the template are the same as those of the ground-truth template.

$$FTA = \frac{\#\ correctly\ parsed\ log\ templates}{\#\ log\ templates}$$

Log Parsers

Table 2.1: Studied Log Parsers

Log Parser	Author	Year	Rationale
LFA	Nagappan et al. [8]	2010	State-of-the-art frequency-based log parser capable of parsing Loghub-2 datasets in a 12-hour time limit, used as a benchmark by Jiang et al. [15]
LogCluster	Vaarandi et al. [20]	2015	
LenMa	Keiichi Shima [22]	2016	State-of-the-art similarity-based log parser capable of parsing Loghub-2 datasets in a 12-hour time limit, used as a benchmark by Jiang et al. [15]
LogMine	Hamooni et al. [23]	2016	
AEL	Jiang et al. [17]	2008	State-of-the-art heuristic-based log parser capable of parsing Loghub-2 datasets in a 12-hour time limit, used as a benchmark by Jiang et al. [15]
Drain	He et al. [10]	2017	
UniParser	Liu et al. [25]	2022	Most recent state-of-the-art semantic-based log parsers, used as benchmark by Jiang et al. [15]
LogPPT	Le et al. [11]	2023	
LLMParser	Ma et al. [16]	2024	State-of-the-art LLM-based log parser using a seq2seq model
LILAC	Jiang et al. [13]	2024	State-of-the-art LLM-based log parser incorporating a grouping-based cache for efficiency
Log3T	Yu et al. [26]	2023	Hybrid log parser incorporating both classification-based and grouping-based techniques

Table 2.1 displays the log parsers that we have studied along with rationale for selecting those. For RQ1.1-RQ1.4, we choose six state-of-the-art syntax-based log parsers, two for each of the three categories: AEL and Drain for heuristic-based log parsers, SHISO and LogMine for similarity-based log parsers, LFA and LogCluster for frequency-based log parsers. For the semantic-based log parsers, we choose two state-of-the-art PLM-based log parsers UniParser and LogPPT, and two LLM-based log parsers: LLMParser and LILAC. 8 of these log parsers have previously been studied by Jiang et al. [15]. We have included LLMParser and LILAC to incorporate LLM-based log parsing in our study. Other examples of LLM-based hybrid log parsers include AdaParser [18] and LogBatcher [19], which we excluded from our empirical study because our empirical study aims to characterize performance and efficiency differences across parsing formulations rather than compare individual implementations within each category.

For RQ1.5-RQ1.6, we study the hybrid log parser Log3T [26], which includes both a token classifier module for identifying constants and variables and a log grouping module for log clustering. By studying the impact of each of the modules in the effectiveness of log

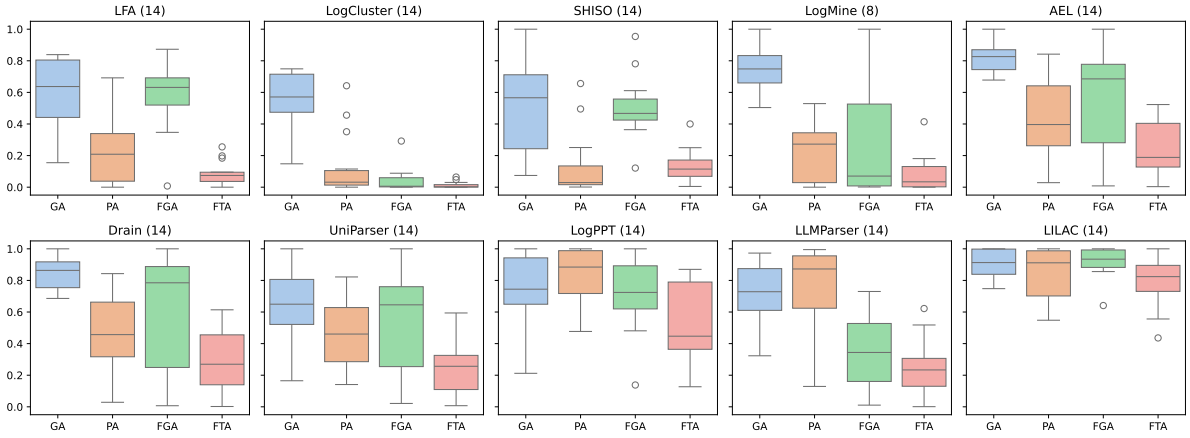


Figure 2.4: Performance evaluation of benchmark syntax-based and semantic-based log parsers (number of datasets processed by the parser in parentheses)

parsing, we aim to understand the impact of grouping-based, classification-based, and hybrid log parsing architectures.

Datasets

Table 2.2 shows the benchmark datasets used in this study. For RQ1.1-RQ1.4, we use the Loghub-2 benchmark datasets [15]. Loghub-2 contains 14 large-scale log datasets, containing varying sizes of log data ranging from 21K to more than 16M.

For RQ1.5-RQ1.7, we use the Loghub-2k datasets [14] instead of Loghub-2 datasets because Log3T fails to parse any of the datasets in Loghub-2 within a reasonable runtime of 12 hours, a threshold previously decided upon by Jiang et al. [15]. Loghub-2k datasets comprise of log data from 16 systems. Each dataset contains 2,000 log messages. Khan et al. [28] detected some issues in the Loghub dataset and published a corrected benchmark. Following recent research [11, 16], we use the corrected benchmark dataset.

Table 2.2: Studied Benchmark datasets

	Loghub-2k [14]		**Loghub-2** [15]	
Dataset	**#Templates**	**#Logs**	**#Templates**	**#Logs**
Hadoop	114	2,000	236	179,993
OpenStack	43	2,000	48	207,632
Zookeeper	50	2,000	89	74,273
HPC	46	2,000	74	429,987
Linux	118	2,000	338	23,921
Mac	341	2,000	626	100,314
Apache	6	2,000	29	51,977
OpenSSH	27	2,000	38	638,946
HealthApp	75	2,000	156	212,394
Proxifier	8	2,000	11	21,320
BGL	120	2,000	320	4,631,261
HDFS	14	2,000	46	11,167,740
Spark	36	2,000	236	16,075,117
Thunderbird	149	2,000	1,241	16,601,745
Android	166	2,000	—	—
Windows	50	2,000	—	—

2.4 Results

2.4.1 RQ1.1: Performance comparison of syntax- and semantic-based log parsers

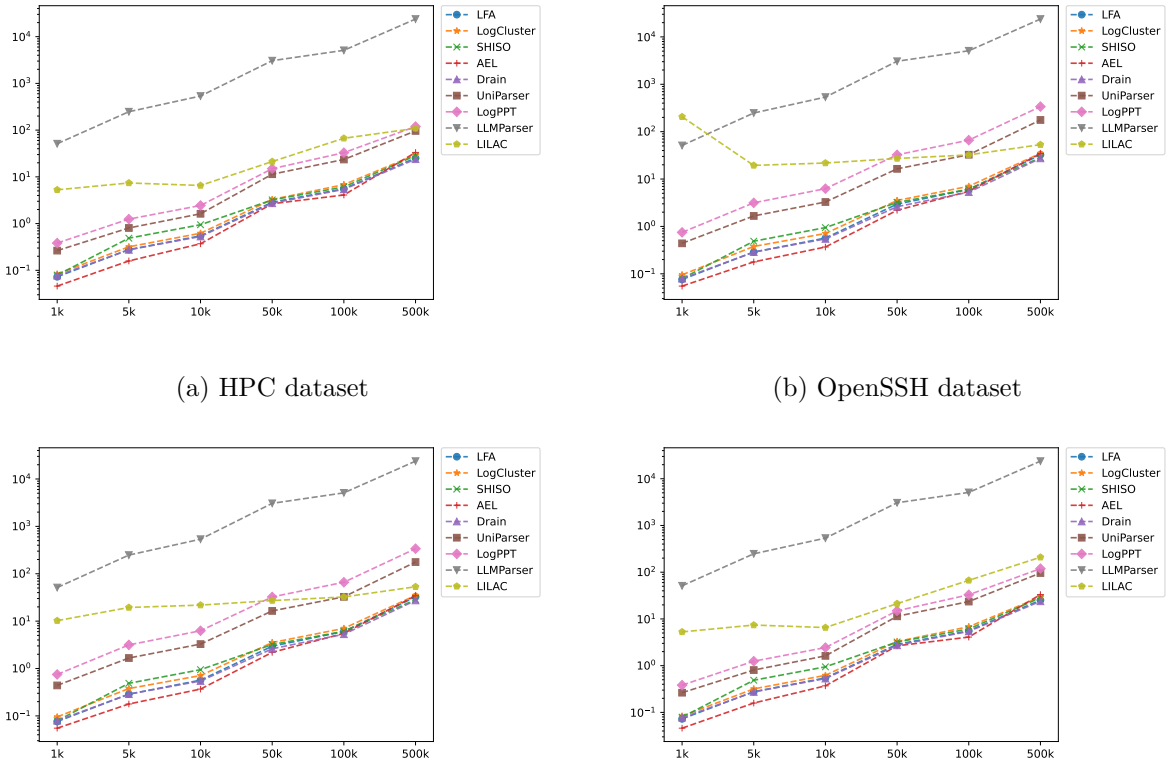
Motivation In this RQ, we examine the performances of syntax-based log parsers and semantic-based log parsers. As discussed in the Section 2.3.2, syntax-based log parsers employ various pattern mining methods, e.g., token matching, frequent pattern mining, longest common subsequence, etc., to cluster log messages so that the log messages of each cluster has the same log template [10, 17]. On the other hand, semantic-based log parsers train a neural network on sample log data or employ an LLM with fine-tuning or in-context learning to identify the constant and the variable tokens [25, 11, 16, 13]. We aim to analyze the strengths and shortcomings of each category of log parsers by conducting a comprehensive evaluation with four accuracy metrics. Extending prior comparative study on Loghub-2 [15], we additionally include recent LLM-based parsers to provide an updated and broader comparison.

Approach The experiments of this RQ are conducted on the Loghub-2 benchmark dataset [15]. For the 6 syntax-based parsers (LFA, LogCluster, SHISO, LogMine, AEL, Drain) and 2 of the semantic-based parsers (UniParser, LogPPT), we reuse the implementations provided by Jiang et al. [15]. For LLMParser and LILAC, we use the implementations provided in their replication packages [16, 13]. For the semantic-based parsers, we do not alter their sampling algorithm or the number of logs chosen for training set.

Result Figure 2.4 presents the effectiveness of the log parsers in terms of the four accuracy metrics. We also present the number of datasets each log parser can finish processing within 12 hours in parenthesis, which is the cutoff time chosen in a prior work by Jiang et al. [15]. According to the figure, the syntax-based log parsers achieve drastically different GA and PA, whereas the GA and PA achieved by the semantic-based log parsers are comparatively closer. Specifically, the syntax-based parsers on average achieve 0.78 and 0.27 GA and PA respectively, thus taking a 51% hit in performance when identifying the log template. On the other hand, semantic-based log parsers achieve on average 0.75 and 0.72 GA and PA respectively, thus achieving similar performance in both log grouping and template identification.

Similar characteristics is seen in template-level accuracy. The syntax-based log parsers achieve an average of 0.43 and 0.14 FGA and FTA respectively, with a 67% hit in identifying the template. On the other hand, the semantic-based log parsers obtain on average of 0.63 and 0.45 FGA and FTA respectively, with a 28% drop in template identification performance. Although the decline is much more prominent for both categories of log parsers in template-level metrics than in message-level metrics, the characteristics remain that semantic-based log parsers achieve comparable performances in terms of both log grouping and template identification, whereas syntax-based log parsers exhibit a sharp decline in the latter compared to the former.

For syntax-based log parsers, log grouping is the primary task. After the clustering, the template for each log group is identified. However, from the stark drop in PA from GA, it is clear that among the logs that are correctly grouped, the log templates of a vast amount



(a) HPC dataset

(b) OpenSSH dataset

(c) BGL dataset

(d) HDFS dataset

Figure 2.5: Runtime efficiency of benchmark syntax-based and semantic-based log parsers

are not correctly identified. Improving the template identification process of syntax-based log parsers may drastically improve their PA and FTA.

Observation 1.1. Semantic-based approaches are better at identifying log template of individual log messages, with 0.75 GA and 0.72 PA on average. Syntax-based approaches are better at grouping the log messages but not as accurate at correct template identification, showing 51% hit in PA compared to GA.

2.4.2 RQ1.2: Efficiency comparison of syntax- and semantic-based log parsers

Motivation Beside accuracy, efficiency is a major factor in evaluating the effectiveness of a log parser in large-scale log parsing. Due to the nature of different log parsing techniques

used in existing log parsers, it is essential to compare their runtime efficiencies.

Approach In this RQ, we compare the benchmark syntax- and semantic-based log parsers in terms of their runtime efficiency. Following prior works [11, 10], we evaluate runtime efficiency using four datasets, HPC, OpenSSH, BGL, and HDFS, since they are relatively large for comparing the efficiency of the benchmark parsers.

Result Figure 2.5 presents the runtime of the three best-performing benchmark log parsers (AEL, Spell, and Drain) on the four datasets in a logarithmic scale. The figure shows that syntax-based log parsers require significantly smaller execution time compared to the inference time of the semantic-based log parsers. Specifically, the LLM-based log parser, LLMParser, requires $10^2 = 100$ times more computation time compared to the PLM-based log parsers. LLM-based log parsers are clearly inadequate for the demands of large-scale log parsing. The syntax-based log parsers, on the other hand, require 10 times less computation time than PLM-based log parsers and 1K times less computation time than the LLM-based log parser. Hence, in terms of efficiency, syntax-based log parsers are the most suitable for large-scale parsing.

Observation 1.2. Syntax-based parsers are significantly faster, requiring $10\times$-$1,000\times$ less time than semantic-based parsers. The runtime efficiency makes semantic-based parsers less suitable for scalable large-scale use.

2.4.3 RQ1.3: Impact of invariant variables in parsing accuracy

Motivation Prior works [25, 11] have argued that syntax-based log parsers only consider the static and dynamic properties to identify the templates, which causes misclassifications of certain variables as constants. Since the values of these variables do not vary across the log messages, we refer to these variables as invariant variables in this study. In this RQ, we aim to investigate how many of the invariant variables each of the benchmark log parsers are unable to correctly identify as a variable, with the motivation that the results would shed

lights into the cause of inaccurate template identification across different categories of log parsers.

Approach To investigate the extent of the impact of invariant variables in the logs of the same template, we conduct the following experiment. For each dataset in the Loghub-2 benchmark, we identify the invariant variable positions from the ground truth. Then, for each of these positions, we validate the log templates identified by the benchmark log parsers. Specifically, we calculate how many of the invariable variables were misidentified by the log parser.

Table 2.3: Number of invariant variable positions misidentified as a constant by the log parser

Datasets	LFA	LogCluster	SHISO	LogMine	AEL	Drain	UniParser	LogPPT	LLMParser	LILAC
Apache	5,941	12,994	12,864	12,994	12,864	12,864	6,204	45	2	0
BGL	22,845	178,628	178,628	29,194	111,169	75,936	6,260	2,281	178,483	2
Hadoop	20,630	45,871	45,602	40,499	45,068	45,080	19,387	1,226	2,120	904
HDFS	0	31	12	21	11	2	7	0	3	0
HealthApp	12,909	29,870	29,870	14,381	24,595	28,617	6,684	5	272	33
HPC	100	108	108	—	106	108	50	0	2	1
Linux	4,517	7,625	7,559	4,066	4,324	7,104	4,045	1,160	807	61
Mac	18,479	82,417	76,320	61,252	49,858	57,349	24,463	4,810	13,633	7,580
OpenSSH	636,445	976,454	834,459	—	573,204	573,204	723,742	37,362	0	0
OpenStack	6,788	143,481	2,073	143,481	8	8	5,225	0	15	0
Proxifier	206	11,266	362	11,266	113	329	0	0	0	0
Spark	13,334	28,835	27,975	—	24,328	23,488	3,599	63	1,833	333
Thunderbird	152,092	396,715	390,319	—	284,926	289,420	93,099	51,158	73,224	40,807
Zookeeper	10,687	11,768	11,688	11,758	11,574	11,250	626	10,486	0	10,424
Average	64,641	137,576	115,560	32,891	81,582	80,340	63,814	7,757	19,314	4,296

Result Table 2.3 presents the number of invariant variable positions in the benchmark datasets with the same value of the variables that are incorrectly identified as constants by the respective log parsers. Syntax-based log parsers misidentify 85k variable positions on average in the benchmark datasets. Semantic-based parsers, on the other hand, misidentify 23K invariant variable positions on average. Specially, LogPPT and LILAC shows the most accuracy in this regard with only 7,757 and 4,296 misclassifications on average.

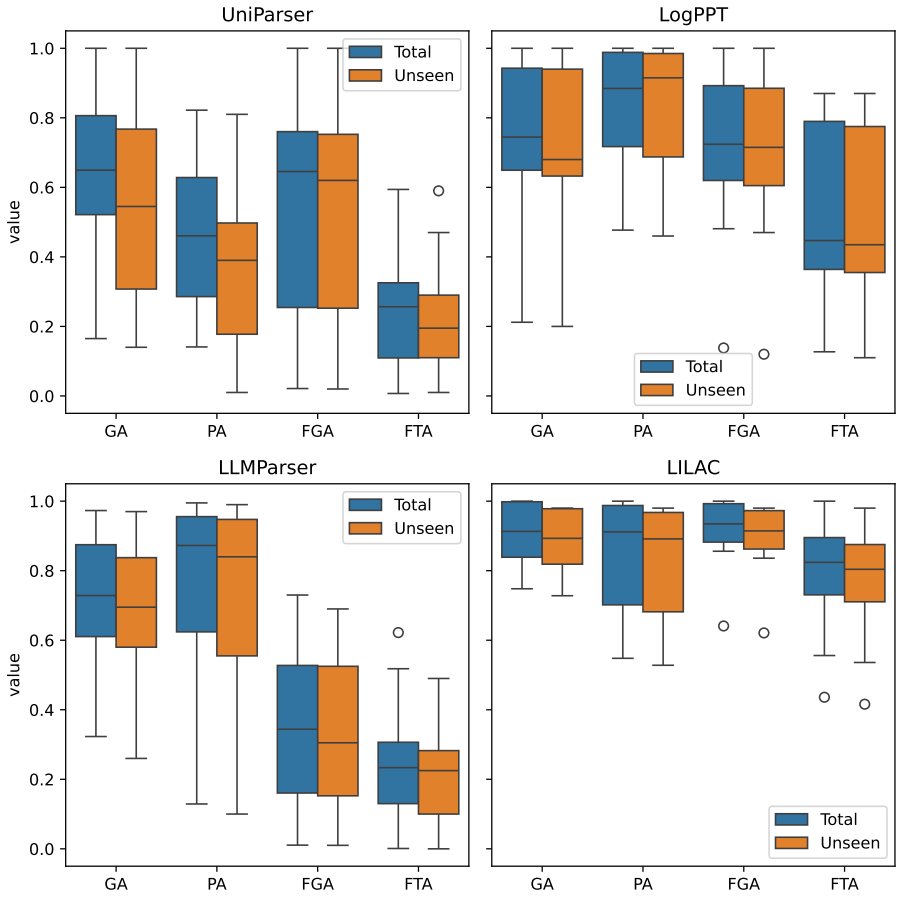


Figure 2.6: Performance of semantic-based log parser on unseen log data

Observation 1.3. Syntax-based parsers misidentify 85,432 variable positions on average in the Loghub-2 datasets, while semantic-based parsers misidentify 23,795 variable positions.

2.4.4 RQ1.4: Performance of semantic-based parsers on unseen log data

Motivation Semantic-based log parsers require training on a set of sample log data. The log parser trains on this sample log data and learns the semantic features of the whole dataset. But prior studies have shown that semantic-based log parsers face challenges generalizing over unseen log data, particularly when the training set is limited [15]. With the recent introduction of LLM-based parsers, it is necessary to revisit and reassess the generalizability

Table 2.4: Comparison of classification-based and hybrid log parsing

Dataset	GA		PA		FGA		FTA	
	Classification based	Hybrid	Classification based	Hybrid	Classification based	Hybrid	Classification based	Hybrid
Apache	**0.86**	0.84	0.16	**0.17**	0.72	**0.85**	0.15	**0.31**
Android	**1.00**	**1.00**	**0.98**	0.69	**1.00**	**1.00**	**0.67**	0.50
BGL	0.88	**0.98**	0.33	**0.35**	0.66	**0.89**	0.15	**0.24**
HDFS	0.81	**1.00**	0.50	**0.90**	0.05	**1.00**	0.04	**0.71**
HPC	0.90	**0.90**	0.63	**0.66**	0.26	**0.78**	0.14	**0.47**
Hadoop	**0.98**	0.95	0.35	**0.37**	**0.88**	0.87	0.28	**0.50**
HealthApp	1.00	**1.00**	0.18	**0.18**	0.98	**1.00**	0.35	**0.37**
Linux	0.29	**0.83**	**0.16**	0.11	**0.81**	0.79	0.28	**0.49**
Mac	0.71	**0.90**	0.28	**0.29**	0.72	**0.82**	0.20	**0.27**
OpenSSH	0.39	**0.82**	0.24	**0.30**	0.18	**0.31**	0.03	**0.08**
OpenStack	0.25	**0.48**	**0.11**	0.10	0.21	**0.79**	0.09	**0.13**
Proxifier	0.00	**1.00**	**0.00**	**0.00**	0.00	**1.00**	**0.00**	**0.00**
Spark	0.74	**0.92**	0.33	**0.40**	0.24	**0.87**	0.09	**0.55**
Thunderbird	0.92	**0.95**	0.03	**0.04**	0.68	**0.80**	0.24	**0.39**
Windows	0.71	**0.99**	0.14	**0.16**	0.73	**0.82**	0.15	**0.42**
Zookeeper	0.97	**0.99**	0.50	**0.50**	0.86	**0.89**	0.40	**0.48**
Average	0.71	**0.91**	0.31	**0.33**	0.56	**0.84**	0.20	**0.37**

of semantic-based approaches.

Approach In this RQ, we study the difference in performance of semantic-based log parsers when evaluated on unseen log data compared to when evaluated on log data including the training samples. Since we reuse the original implementations of the benchmark semantic-based log parsers, we also use their specific log sampling algorithm to choose the training set. We record of the training set chosen by the individual sampling algorithms of each parser and when evaluating their accuracies on unseen logs, we exclude the training set from the benchmark datasets.

Result Figure 2.6 presents the performances of the semantic-based log parsers on the whole benchmark dataset and on only the unseen log data. All three benchmark semantic-based log parsers demonstrate a drop in performance when evaluated only on unseen log data. While LogPPT and LILAC shows a reasonable drop ranging from 1% to 3%, both UniParser and LLMParser suffer a drop ranging from 12% to 23% and from 4% to 15% respectively. The

results indicate the lack of generalization ability of semantic-based log parsers across unseen log data.

Observation 1.4. All the benchmark semantic-based log parsers show performance drops across all metrics when evaluated only on unseen log data, ranging from upto 3% in LogPPT and LILAC and upto 23% in UniParser and LLMParser.

2.4.5 RQ1.5: Performance comparison of classification-based and hybrid log parsing

Motivation In this RQ, we compare the performances of classification-based and hybrid parsing architectures. In classification-based log parsing, only semantic log parsing technique is used, whereas in hybrid log parsing, both syntax-based log grouping and semantic-based token classification is used. For example, Log3T is comprised of two modules: (1) a BERT-based classifier that classifies the top k tokens in a log as most likely to be constants and (2) a grouping module that groups logs with the same constant tokens in the same order. Since classification-based architecture have shown impressive parsing accuracy, it is important to compare its performance with that of a hybrid architecture with the same classifier model.

Approach To study the impact of hybrid parsing in comparison to classification-based parsing, we modify Log3T and remove the grouping module, effectively transforming it into a classification-based log parser. The BERT-based classifier is trained on a set of training log data to learn the semantic representations and is able to classify each token as a variable or a constant.

Result Table 2.4 presents the performances obtained by classification-based parsing and hybrid parsing. Hybrid parsing achieves better accuracy across all four metrics, with the performance improving from 27% in GA to 81% in FTA. The improvement in PA, however, is only 6.17%, which is statistically insignificant. The results suggest that the hybrid architecture substantially enhances a log parser's ability to correctly group log messages belonging

Table 2.5: Impact of the order of Log Grouping and Token Classification modules in hybrid log parsing

	GA		PA		FGA		FTA	
Dataset	**Log3T**	**Log3T'**	**Log3T**	**Log3T'**	**Log3T**	**Log3T'**	**Log3T**	**Log3T'**
Apache	**0.84**	0.81	**0.17**	0.13	**0.85**	0.82	**0.31**	0.29
Android	**1.00**	**1.00**	**0.69**	**0.69**	**1.00**	**1.00**	**0.50**	**0.50**
BGL	0.98	**0.98**	**0.35**	**0.35**	0.89	**0.90**	0.24	**0.24**
HDFS	**1.00**	**1.00**	**0.90**	**0.90**	**1.00**	**1.00**	**0.71**	**0.71**
HPC	0.90	**0.91**	**0.66**	**0.66**	0.78	**0.83**	0.47	**0.49**
Hadoop	**0.95**	**0.95**	**0.37**	**0.37**	**0.87**	**0.87**	**0.50**	**0.50**
HealthApp	**1.00**	**1.00**	**0.18**	**0.18**	**1.00**	**1.00**	**0.37**	**0.37**
Linux	**0.83**	0.77	**0.11**	0.10	**0.79**	0.75	**0.49**	0.46
Mac	**0.90**	**0.90**	**0.29**	**0.29**	0.82	**0.83**	**0.27**	**0.27**
OpenSSH	**0.82**	**0.82**	**0.30**	**0.30**	**0.31**	**0.31**	**0.08**	**0.08**
OpenStack	0.48	**0.95**	**0.10**	**0.10**	0.79	**0.85**	0.13	**0.14**
Proxifier	0.52	**1.00**	**0.00**	**0.00**	0.80	**1.00**	**0.00**	**0.00**
Spark	**0.92**	0.92	**0.40**	0.40	**0.87**	0.82	0.48	**0.55**
Thunderbird	0.95	**0.95**	0.04	**0.04**	0.80	**0.81**	0.39	**0.39**
Windows	**0.99**	**0.99**	**0.16**	**0.16**	**0.82**	**0.82**	**0.42**	**0.42**
Zookeeper	**0.99**	0.99	**0.50**	0.50	**0.89**	0.86	**0.48**	0.45
Average	0.88	**0.93**	**0.33**	**0.33**	0.83	**0.84**	0.36	**0.37**

to the same template, although it has limited impact on the message-level accuracy of the log templates.

Observation 1.5. Hybrid log parsing achieves better accuracy across all four metrics, with 27% and 81% improvement in GA and FTA, respectively.

2.4.6 RQ1.6: Impact of the order of log grouping and token classification modules

Motivation Besides comparing the performance of a hybrid architecture with that of a classification-based architecture, it is also important to compare the performance of the hybrid architecture based on the order of the log grouping and the token classification modules in its pipeline.

Approach In this RQ, we modify Log3T, hereby referred to as Log3T-mod, by moving the grouping module before the classification module. In the original Log3T, at first the tokens

of the log message is given a likelihood of being a constant or a variable. The top k tokens most likely to be constants are then used to cluster the log message. In the modification for this RQ, Log3T-mod first tries to match the log message with existing log templates, i.e., templates of the log messages that have already been parsed, and if no match is found, Log3T-mod uses the classifier probabilities to find a match. If no match is found this time either, a new log group is created with the current log message as its sole member (this last step is unchanged from Log3T).

Result Table 2.5 shows that Log3T-mod outperforms Log3T in all but one metric. Specifically, on average, it achieves 5.4% higher GA, 1.2% higher FGA, and 2.7% higher FTA, while retaining the same average accuracy in PA. Although the improvements are not significant, the fact that there is improvement just by changing the order of the log grouping and token classification modules in the original Log3T suggests that prioritizing grouping before classification would yield better accuracy. The original design of Log3T relies on the token classification being done prior to grouping. So, it stands to reason that changing their order would drastically reduce its performance. However, the opposite happens. That is indicative of the importance of the order of the log grouping and token classification modules. Recent log parsers, e.g., LILAC and AdaParser, have adopted a hybrid pipeline with log grouping preceding token classification.

Observation 1.6. Placing the log grouping module before the token classification module in a hybrid log parser provides a performance boost in 3 metrics, while retaining the same performance in the other. It achieves 5.4% higher GA, 1.2% higher FGA, and 2.7% higher FTA.

Table 2.6: Regex patterns for common variables

Variable Category	Variable Type	Regex Pattern
Location Identifier	IP address	`(?:[-0-9a-zA-Z]+\.){2,}[-0-9a-zA-Z]+(?::?:\d+)?`
	MAC address	`([A-Fa-f0-9]{2}:){5,11}[A-Fa-f0-9]{2}`
	Email address	`[0-9a-zA-Z]+@([0-9a-zA-Z]+\.)+[0-9a-zA-Z]+`
	Unix path	`(\/[\d+\w+\-_\.\\\#\$]*[\/\.][\d+\w+\-_\.\\\#\$\/*]*)+(\sHTTPS?\/\d\.\d)?`
	Windows path	`([a-zA-Z]\:[\/\\][\d+\w+\-_\.\\\#\$]*([\/\\\.][\d+\w+\-_\.\\\#\$\\\/*]*)?)`
Time/Duration	Date & Time	`(\d{1,4}(-|/)\d{1,2}(-|/)\d{1,4})`
	Duration	`[+-]?(\d+s(\d+\s?ms)?|\d+\s?ms)`
Computing Resource	Memory	`(\d+(\.\d+)?)\s?[kmgKMG][bB]?((\/s)|(ytes))?`

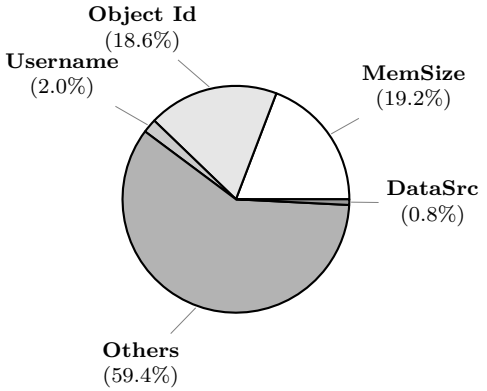


Figure 2.7: Distribution of variables unmatched by regex patterns

2.4.7 RQ1.7: Regularity of structural patterns in variables in benchmark datasets

Motivation Li et al. categorized log variables into 10 variable types: 1) Object ID (OID), 2) Location Indicator (LOI), 3) Object Name (OBN), 4) Type Indicator (TID), 5) Switch Indicator (SID), 6) Time/Duration of an Action (TDA), 7) Computing Resources (CRS), 8) Object Amount (OBA), 9) Status Code (STC), 10) Other Parameters (OTP). Among them, Time/Duration, Computing Resources (e.g, 126MB, 256KBytes), and Location indicators (e.g., IP addresses, file paths) have well defined formats that can be captured by deterministic and robust techniques such as regex pattern matching. In this RQ, we aim to investigate

Table 2.7: Frequency analysis of variable types in Loghub-2k

Dataset	Total	MAC Address	Email Address	IP Address	Time Duration	Memory Size	Date & Time	File Paths
Proxifier	5,146	0	0	3,022	0	999	0	0
Linux	3,901	0	0	1,686	0	10	910	9
Apache	633	0	0	32	0	0	0	601
Zookeeper	673	0	0	614	40	0	0	5
Hadoop	1,928	0	0	73	2	0	0	4
HealthApp	271	0	0	4	0	0	0	0
OpenStack	3,270	0	0	847	0	60	0	378
HPC	214	0	0	93	0	0	0	0
Mac	1,287	20	0	92	0	0	0	141
OpenSSH	2,939	0	0	1,826	0	0	0	0
Spark	922	0	0	0	0	298	0	1
Thunderbird	1,902	40	0	654	0	17	1	55
BGL	220	10	0	36	0	1	0	119
HDFS	2,806	0	0	750	0	0	0	715
Android	1,503	0	0	88	3	0	0	0
Windows	637	0	0	8	0	0	0	6
Total	28,252	70	0	9,825	45	1,385	911	2,034

what portion of the variables in the benchmark log datasets of Loghub-2k belong to the categories of variables that have well-defined formats.

Approach In Table 2.6, we present a set of regex patterns to identify the variable types which have well-defined formats. We measure the portion of the variables in the benchmark log datasets of Loghub-2k that are captured by these regex patterns. We choose Loghub-2k datasets as benchmark for this RQ, so that we can use Loghub-2 as benchmarks for other experiments without the possibility of data leakage.

Result The frequency of each variable type presented in Table 2.7 show that 50% of the variables can be identified with the proposed regex patterns. Specifically, in Apache and Zookeeper, more than 90% of the variables are file paths. Moreover, IP addresses and file paths together cover 50% of the variables in total.

Figure 2.7 presents the distribution of the 50% variables that remain unmatched by SynLog+ regex patterns. 38% of them are memory sizes (‘1038 bytes (1.03 KB)’) and object

identifiers ('blk_7236'), whose patterns are domain- or dataset-specific. Usernames ('cyrus', 'aadmin1') and data sources ('cn602') consists 2.7% of the unmatched variables. The rest of them are of no distinguishable patterns.

Observation 1.7. 50% of the variables in Loghub-2k datasets can be identified using regex patterns in Table 2.6 based on the variable types identified by Li et al.

2.5 Discussion

2.5.1 Takeaways from the Empirical Study

From the study presented in this chapter of log parsing techniques and architectures, we have the following takeaways (C = Conclusion).

C1.1. Syntax-based methods are better for efficient and accurate log grouping. RQ1.1 and RQ1.2 have shown that syntax-based log parsing techniques provide better grouping accuracy (Obs. 1.1) and better efficiency (Obs. 1.2). Semantic-based log parsers, on the other hand, obtain better parsing accuracy but incur 10 to 1,000 times more runtime costs (Obs. 1.2). Given the high throughput requirements of large-scale log analysis systems, we recommend that log parsers favour syntax-based techniques for matching incoming logs to existing templates instead of invoking the semantic model for every new log message.

C1.2. Semantic-based methods are better for accurate template identification. RQ1.1 and RQ1.2 have also shown that semantic-based log parsers are better at accurately identifying the log templates (Obs. 1.1). Although this accuracy comes at the cost of a decreased efficiency (Obs. 1.2), log parsers should favour semantic-based methods to accurately identify the constant and variable tokens of the log templates.

C1.3. Generalizable semantic-based log parsers require more study. Semantic-based log parsers suffer a hit in accuracy when evaluating their performance only on unseen log data (Obs. 1.3). Semantic-based models require a subset of log data to train on. Overfitting on these training data can cause the performance to appear inflated. Further research

should be conducted to increase the generalizability of the semantic-based log parsers.

C1.4. Hybrid architecture can leverage strengths of both grouping-based and classification-based parsing. The experimental results indicate that decoupling the grouping and template identification steps can improve accuracy, obtaining 27%, 49%, and 81% improvement in GA, FGA, and FTA, respectively (Obs. 1.4). LILAC, the baseline log parser with the highest accuracy, is a hybrid log parser. Based on these observations, we recommend that log parsers employ hybrid log parsing architecture.

C1.5. Log grouping before template identification. LILAC achieves the highest accuracy among all the baseline log parsers. The cause behind LILAC's impressive accuracy is its hybrid architecture, specifically its template cache acting as the log grouping module followed by an LLM acting as the template identification module. Moreover, a case study of Log3T reveals that log grouping before token classification yields performance boosts in all accuracy metrics (Obs. 1.5, 1.6). Based on these findings, we recommend that log parsers Perform log grouping before template identification.

C1.6. Domain-agnostic heuristics may improve variable identification. While recent log parsers have moved away from regex-based pre-processing due to concerns over domain specificity, Li et al. have shown that log variables often fall into a limited set of categories [31]. Notably, three of these categories can be identified with regex pattern matching, such as IP addresses, time/duration, memory size or clock cycles, etc. Experimental results show that 50% of the variables in Loghub-2k can be identified by general-purpose regex patterns. Log parsers should first consider rule-driven pre-processing before resorting to complex models.

2.5.2 Threats to Validity

Internal validity 8 out of the 10 selected benchmark log parsers are implemented based on the implementations provided by Jiang et al. [15]. The hyperparameters provided in Loghub-2, which we reuse without modification, may not be tuned to the benchmark log

datasets Loghub-2 we use in this study, since many of them were published with evaluations on an earlier and smaller dataset, Loghub-2k. Nevertheless, both Loghub-2 and Loghub-2k contain logs from the same systems. Hence, the values of the hyperparameters should not differ much between the two datasets.

External validity For RQ1.4 and RQ1.5 in our empirical study, we conduct the experiments only on Log3T, which may bring to question the applicability of the results in general. However, we have set the experiments by analyzing and comparing all possible combinations of both grouping- or classification-based and hybrid architectures, thereby eliminating the concern of general applicability.

Construct validity The selection of log parsers is limited, as not all existing log parsers are open-sourced due to industry confidentiality reasons [32]. Nevertheless, the selected parsers include state-of-the-art log parsers published at top-tier conferences and cover all existing categories of technology. The benchmark log parsers selected for evaluation are limited in range since only open-source parsers are selected. However, the selected parsers are state-of-the-art parsers in their respective categories that should adequately represent all existing log parsing approaches in practice.

2.6 Conclusion

In this chapter, we conducted an empirical study which analyzes the performances of syntax-based and semantic-based log parsers in addition to the impact of grouping and template identification in hybrid log parsing architectures. The results of the study demonstrate that syntax-based log parsers achieve better grouping accuracy and efficiency but are outperformed by semantic-based log parsers in parsing accuracy, although semantic-based parsers lack in generalization. Additionally, the study shows that compared to hybrid architecture, hybrid architecture obtains better accuracy and efficiency.

Chapter 3

SynLog+: Optimized Log Parsing with Syntactic Modifications

3.1 Introduction

Traditional log parsing approaches primarily rely on syntactic similarities among log messages to identify log templates [10, 17, 22]. However, relying solely on syntactic clustering often leads to incorrect template inference, particularly when variables are mistakenly treated as constants [25, 11]. To address this limitation, recent log parsing research has increasingly explored semantic-based techniques [25, 11], which leverage semantic understanding to improve template extraction.

While semantic-based parsers can achieve higher parsing accuracy, they also introduce practical challenges. These methods typically require training data and significantly higher computational resources compared to traditional syntax-based parsers, making them less suitable for large-scale log analysis [15]. More recently, LLM-based log parsing approaches have demonstrated promising accuracy [16, 13]. However, as shown in the empirical study presented in Chapter 2, these approaches incur substantially higher computation costs. Consequently, current research efforts have focused on improving the efficiency of semantic-based

parsing, for example by incorporating syntax-based mechanisms such as tree-based caches to reduce the number of expensive inference calls [13, 18].

Our empirical study in Chapter 2, however, suggests an alternative direction for improving log parsing performance. Instead of focusing solely on reducing the computational cost of semantic-based approaches, the study reveals opportunities to improve the parsing accuracy of efficient syntax-based parsers. In particular, we observe that many parsers achieve relatively high grouping accuracy while still producing incorrect templates due to errors in variable identification (Obs. 1.4). In many cases, tokens that remain constant within a group of log messages are incorrectly inferred as part of the template, even though they represent variable components. Furthermore, the study shows that a significant portion of variables in the Loghub-2k benchmark datasets follow recognizable patterns, such as IP addresses or file paths, which can be reliably detected using regular expressions (Obs. 1.7). These observations suggest that template inference errors can often be corrected using lightweight syntactic refinements without requiring expensive semantic analysis.

Motivated by these insights, this chapter introduces **SynLog+**, a template refinement pipeline designed to reduce the discrepancy between grouping accuracy (GA) and parsing accuracy (PA) observed in existing log parsers. SynLog+ operates as a post-group refinement step that improves template inference after log messages have already been grouped. By refining incorrectly inferred templates, SynLog+ significantly improves the parsing accuracy of syntax-based log parsers while preserving their computational efficiency.

Across the benchmark parsers, SynLog+ improves the parsing accuracy of syntax-based approaches by 236% on average while maintaining their efficiency. In contrast, the improvement for semantic-based parsers is smaller (20.6% on average), largely because their parsing accuracies are already close to their grouping accuracies. Overall, employing SynLog+ on the baseline log parsers yields average improvements of 17% in GA, 157% in PA, 181% in FGA, and 553% in FTA.

The remainder of this chapter presents the design and implementation of SynLog+. We

first describe the overall architecture of the framework and its design rationale guided by the empirical study in Chapter 2, followed by the detailed methodology of each component. Finally, we evaluate the effectiveness of SynLog+.

3.2 Methodology

In this section, we present SynLog+, a novel technique that can act as a wrapper around any existing log parsing techniques to improve their parsing accuracy without modifying their internal logic. SynLog+ achieves this by improving the log template identification process in the techniques. This decoupled design enables plug-and-play deployment, preserves the efficiency and scalability characteristics of the base parser, and avoids re-engineering mature systems already used in practice.

We focus specifically on improving template identification because empirical analysis shows that many parsing errors stem not from incorrect grouping but from inaccurate identification of constant and variable tokens within clusters (Obs 1.3). By targeting this stage, SynLog+ directly addresses a dominant source of parsing inaccuracy while remaining lightweight and complementary to existing approaches.

In this section, we first discuss the design rationale behind SynLog+ in Section 3.2.1. Then, we present the methodology behind SynLog+ in Section 3.2.2.

3.2.1 Design Rationale Behind SynLog+

The findings from our empirical study reveal several consistent patterns across log parsing techniques and architectures. These observations directly inform the design of SynLog+. Table 3.1 summarizes the key insights from our empirical study, which influenced the design decisions of our technique, SynLog+. We discuss the rationales below.

Table 3.1: Observations from the empirical study motivating the design choices of SynLog+

Observation	**SynLog+ Design Rationale**	**SynLog+ Module**
Obs 1. Syntax-based parsers achieve high grouping accuracy but low parsing accuracy, indicating failures in template identification after grouping	Improve template identification within correctly clustered log groups without altering the grouping mechanism itself	Variable inference module Constant verification module Post-processing module
Obs 2. Syntax-based parsers are 10–1,000× faster than semantic-based parsers	Preserve the runtime efficiency of syntax-based grouping methods for scalable log parsing	Log grouping module Log sampling module
Obs 3. Syntax-based parsers misidentify 3.5× variable positions on average in the Loghub-2 datasets compared to semantic-based parsers	Improve variable identification accuracy within syntax-based pipelines without incurring the overhead of semantic models	Anonymization module
Obs 7. Approximately 50% of variables correspond to recurring structural patterns detectable via regular expressions	Recover common variable types using pattern-based anonymization before template inference	

Addressing the Template Identification Gap (Obs. 1.1)

Our results show that syntax-based parsers achieve high grouping accuracy (GA) but substantially lower parsing accuracy (PA). This indicates that many log messages are correctly clustered, yet their templates are incorrectly identified. In other words, the primary limitation of syntax-based approaches lies not in grouping, but in post-group template identification.

This observation suggests that improving template identification within correctly formed log groups could significantly increase overall parsing accuracy without modifying the grouping mechanism itself.

Preserving Scalability (Obs. 1.2) and Generalization (Obs. 1.4)

While semantic-based parsers achieve stronger template identification performance, they incur substantially higher runtime costs (Obs. 1.2) and demonstrate performance degradation on unseen log data (Obs. 1.3). These limitations make semantic approaches less suitable for large-scale or evolving log environments.

Therefore, the design of SynLog+ avoids reliance on semantic token classification models. Instead, it builds upon the efficiency and structural robustness of syntax-based grouping methods while targeting their template identification weakness (Obs. 1.1).

Leveraging Architectural Insights (Obs. 1.5, 1.6)

Our architectural analysis shows that hybrid approaches combining grouping and token-level signals outperform single-formulation methods (Obs. 1.4). Furthermore, performing grouping before template identification improves grouping-related metrics (Obs. 1.5).

These findings suggest two important design principles: 1) Log grouping should remain the first stage of the pipeline, and 2) Template identification should utilize both global and token-level information.

Accordingly, SynLog+ isolates log grouping as a first-stage module and performs template refinement within each group, rather than attempting independent message-level parsing.

Exploiting Structured Variable Patterns (Obs. 1.6)

Finally, our analysis of benchmark datasets shows that approximately half of the variables correspond to recurring structural patterns such as IP addresses, file paths, memory sizes, and timestamps (Obs. 1.6). These patterns can be identified efficiently using regular expressions.

This indicates that a substantial portion of template identification errors can be corrected using lightweight, pattern-based techniques without requiring learned semantic models.

SynLog+ therefore incorporates regex-based anonymization rules to recover common variable types prior to template refinement.

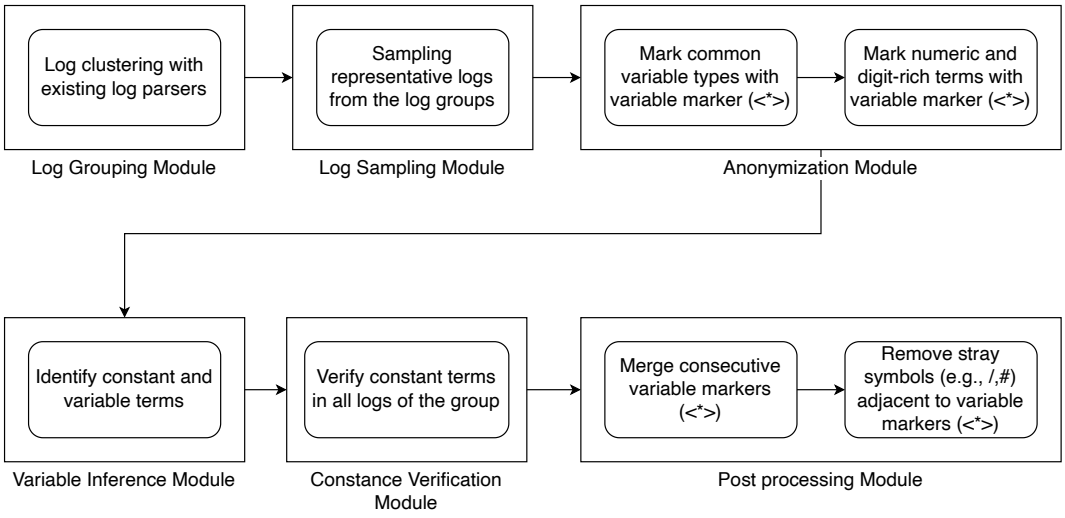


Figure 3.1: SynLog+ workflow

3.2.2 SynLog+ Methodology

SynLog+ acts as a wrapper around any existing log parser to improve the parser's accuracy. Figure 3.1 displays the modules and workflow of SynLog+, while Algorithm 1 details its procedural steps. The last column of Table 3.1 maps the observations from our empirical study to specific modules of SynLog+, i.e., how each module is designed to address the problems we observed in our empirical study.

Log grouping module

Line 1 in Algorithm 1 show the log grouping module of SynLog+, where the clustering of similar logs is done by an existing log parser. The subsequent modules of SynLog+ then identifies the log template of each log group by identifying the constant and variable terms in the constituent logs of the log groups.

Log sampling module

The overall objective of SynLog+ is to identify variables among the logs in a log group. For this, we have to iterate over all the constituent log messages. But this would be severely time consuming and cost efficiency. So, instead of iterating over all the logs in a log group,

Algorithm 1: SynLog+ Methodology

```
Input: Log messages 𝓛 = {l_1, ..., l_n}
Input: An existing log parser 𝒫
Output: Refined templates 𝒯
// Module 1: Log Grouping Module
1  𝒢 = get_log_groups(𝓛, 𝒫)
2  foreach log group g ∈ 𝒢 do
3      Let T be the template of g
       // Module 2: Log Sampling Module
4      L ← sample_representative_logs(g, 𝓛)
       // Module 3: Anonymization Module
5      foreach log l in L do
           // Module 3a: Anonymizing common variable patterns
6          foreach regex r in R do
7              Replace the matched text with "<*>"
           // Module 3b: Anonymizing numbers
8          foreach token t in l do
9              if t is a number then
10                 Replace t with "<*>"
       // Module 4: Variable Inference Module
11     foreach token position i in sampled logs L do
12         if tokens at position i differ across logs in L then
13             Mark position i as "<*>"
       // Module 5: Constant Verification Module
14     foreach log l in L do
15         if values of t differs across g then
16             Replace t with "<*>"
       // Module 6: Post-processing Module
17     Replace multiple consecutive "<*>" with a single variable marker "<*>"
18     Remove stray symbols (e.g., !, #) adjacent to variable marker "<*>"
19 return refined template set 𝒯
```

we choose n log messages as representative logs from the set of unique logs in a log group and follow the next steps. Line 3 shows the log sampling module in Algorithm 1.

Regarding the value of n, a higher value may reduce the efficiency of the parser, while a lower value may reduce the accuracy. To evaluate the impact of the number of representative logs selected for this step, we design an experiment described in Section 3.3.

Anonymization module

Following AEL [17], we begin with anonymization, i.e., identifying likely variable tokens in the logs. Lines 5 to 10 in Algorithm 1 show the anonymization module. Anonymization boosts template accuracy by marking variables with heuristics before employing any syntax-

or semantic-based methods. To anonymize variable tokens, we replace them with the variable marker "<*>". In this step, we design the anonymize step based on the following observations.

Regex for common variable patterns. Although initially most of the log parsers made use of regex patterns to preprocess the log messages [10], recently researchers [11] have argued that using regex patterns are not feasible in real life scenarios because of two issues: 1) they are specific to individual log datasets, and 2) they need to be updated with the update of the software system. Although these two arguments are valid, they concern only a subset of regex patterns. Some regex patterns are indeed specific to a dataset and require regular updates to match the evolution of the software (e.g., "blk_2345" in BGL), but many variable terms consist of common patterns across any and all log dataset (e.g., IP addresses, MAC addresses, file paths, etc.). Since the issues argued against the use of regex patterns in log parsing do not apply to these common patterns, we used these regex patterns to anonymize the log messages.

Numbers are variables. Most log parsers, even the semantic-based ones, regard numbers as variables. Some log parsers only consider pure numbers (i.e., real or floating numbers in decimal or hexadecimal) as variables, whereas some log parsers consider terms with $p\%$ digit characters as variables. In SynLog+, we consider those terms as variables which are pure numbers in octal, decimal, or hexadecimal.

Anonymizing common variable patterns As shown in lines 6 to 7, we match for common variable patterns with regex matching. Li et al. conducted a study exploring the characteristics of dynamic variables in log messages [31]. Among the 9 categories of variables in their study, 5 are numbers, which we have already anonymized in the previous step. Among the other 4 categories, Object Name (OBN) do not follow any universal pattern. The other three categories include (1) Location Indicator, (2) Time/Duration of an Action, and (3) Computing Resources. Location Indicator can be path information, a URI, IP, or MAC address. Time/Duration of an Action shows the execution time or duration of an action. It can either be a precise date and time information (e.g., "Sat Jun 18 02:08:10 2005") or a

duration of time in seconds, minutes, or hours (e.g., "Scheduled snapshot period at 10ms"). Computing Resources include the amount of memory (e.g., 126MB) or clock cycle (e.g., 1.3 MHz) consumed by a process. We design regex patterns, presented in Table 2.6, to capture these three categories of variables in the log messages.

Anonymizing numbers As shown in lines 8 to 10, we anonymize the terms that are pure numbers in octal, decimal, or hexadecimal.

Variable inference module

Lines 11 to 13 in Algorithm 1 show the variable inference module. After the Anonymization module, we iterate over the terms of the n representative log message for each log group and identify those terms as constants which are present in both log messages *in the same order* and those terms as variables which are not present in both log messages. The variable terms identified in this manner are indeed variable terms in the log template.

Constant verification module

The constant terms thus far identified may not be constant terms in the log template, since they may be absent in the other logs in the log groups. Thus, as shown in lines 14 to 16 in Algorithm 1, we verify the constant terms in the template. After identifying the variables in the n representative logs, we iterate over the constant terms and search for them in all the logs. If absent in any of the logs, we mark the constant term as variable instead.

Post-processing module

Lines 17 to 18 in Algorithm 1 show the post-processing module in SynLog+. In this step, multiple consecutive variable markers are substituted with a single variable marker (`<*>`). Moreover, due to imprecise tokenization, the variable markers in the log template may have stray symbols, e.g., slashes or pounds, before or after it. In this step, we remove these stray characters. In other words, we make these a part of the variable terms.

Table 3.2: Improvement in GA. The columns with ✗ and ✓ indicate the results without and with SynLog+, respectively.

	LFA		LogCluster		SHISO		LogMine		AEL		Drain		UniParser		LogPPT		LLMParser		LILAC	
	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓
Proxifier	**0.35**	**0.35**	0.66	**0.67**	0.69	**0.98**	0.50	**0.52**	**0.83**	**0.83**	**0.69**	**0.69**	**0.51**	**0.51**	**0.99**	**0.99**	**0.96**	**0.96**	**1.00**	**1.00**
Linux	0.23	**0.69**	0.60	**0.65**	0.07	**0.52**	0.74	**0.76**	0.68	**0.70**	0.69	**0.71**	0.20	**0.68**	0.21	**0.22**	0.50	**0.96**	**0.83**	0.83
Apache	**0.81**	**0.81**	0.55	**0.99**	0.57	**0.59**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.17	**0.59**	0.80	**0.99**	0.88	**1.00**	**1.00**	**1.00**
Zookeeper	0.84	**0.98**	0.74	**0.99**	0.82	**0.96**	0.70	**0.99**	1.00	**1.00**	**0.99**	**0.99**	0.99	**1.00**	0.98	**0.99**	0.89	**1.00**	**0.99**	**0.99**
Hadoop	0.83	**0.91**	0.48	**0.90**	0.72	**0.77**	0.83	**0.91**	0.82	**0.94**	0.92	**0.94**	0.59	**0.94**	0.68	**0.93**	0.87	**0.97**	**0.92**	**0.92**
HealthApp	0.80	**0.94**	0.73	**0.94**	0.08	**0.37**	0.55	**0.68**	0.73	**0.97**	0.86	**0.99**	0.59	**0.87**	**1.00**	**1.00**	0.74	**1.00**	**0.99**	**0.99**
OpenStack	**0.67**	**0.67**	0.69	**0.86**	0.81	**0.89**	0.76	**0.98**	**0.74**	**0.74**	0.75	**0.82**	0.56	**1.00**	**1.00**	**1.00**	0.74	**1.00**	**1.00**	**1.00**
HPC	**0.73**	**0.73**	0.73	**0.73**	0.08	**0.08**	—	—	0.75	**0.80**	0.79	**0.84**	**0.81**	**0.81**	0.76	**0.84**	0.64	**0.70**	**0.87**	**0.87**
Mac	0.59	**0.70**	0.46	**0.80**	0.61	**0.79**	0.85	**0.88**	0.80	**0.80**	0.76	**0.90**	0.85	**0.90**	0.58	**0.82**	0.60	**0.85**	0.82	**0.91**
OpenSSH	0.16	**0.49**	0.22	**0.50**	0.40	**0.70**	—	—	**0.71**	**0.71**	**0.71**	**0.71**	0.28	**0.34**	0.28	**0.39**	0.32	**0.60**	**0.75**	**0.75**
HDFS	0.81	**0.81**	0.49	**0.76**	**1.00**	**1.00**	—	—	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.82	**1.00**	0.97	**1.00**	**1.00**	**1.00**
Spark	0.50	**0.84**	0.15	**0.76**	0.19	**0.46**	—	—	0.83	**0.87**	0.87	**0.87**	0.77	**0.77**	0.69	**0.99**	0.72	**0.98**	0.79	**0.79**
Thunderbird	0.61	**0.72**	0.47	**0.62**	0.50	**0.57**	—	—	0.83	**0.86**	**0.88**	0.87	0.79	**0.80**	0.64	**0.70**	0.67	**0.73**	**0.90**	0.89
BGL	0.42	**0.43**	0.75	**0.88**	0.56	**0.65**	—	—	0.88	**0.89**	0.91	**0.91**	0.71	**0.71**	0.73	**0.73**	0.58	**0.71**	**0.91**	**0.91**
Average	0.60	**0.72**	0.55	**0.79**	0.51	**0.67**	0.74	**0.78**	0.83	**0.86**	0.84	**0.88**	0.63	**0.78**	0.73	**0.83**	0.72	**0.89**	0.91	**0.92**

3.3 Results

To evaluate SynLog+, we answer the following research questions (RQs).

RQ2.1. How much performance boost is achieved by SynLog+ when coupled with benchmark log parsers?

RQ2.2. How efficient is SynLog+ compared to the benchmark log parsers?

RQ2.3. What impact does the number of representative logs chosen have on the performance?

RQ2.4. How effective is SynLog+ in identifying invariant variables?

3.3.1 Experimental Setup

For the evaluation, we use the Loghub-2 benchmark [15] as described in Section 2.3.2, the same set of evaluation metrics discussed in Section 2.3.2, and the same 12 state-of-the-art log parsers from Section 2.3.2 as baselines.

Table 3.3: Improvement in PA. The columns with ✗ and ✓ indicate the results without and with SynLog+, respectively.

	LFA		LogCluster		SHISO		LogMine		AEL		Drain		UniParser		LogPPT		LLMParser		LILAC	
	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓
Proxifier	0.00	**0.42**	0.00	**0.96**	0.50	**1.00**	0.00	**0.98**	0.68	**0.90**	0.69	**1.00**	0.63	**0.88**	**1.00**	0.79	**0.96**	0.79	**1.00**	**1.00**
Linux	0.03	**0.63**	0.02	**0.64**	0.02	**0.58**	0.04	**0.61**	0.10	**0.61**	0.11	**0.63**	0.14	**0.66**	0.52	**0.64**	0.43	**0.59**	0.68	**0.69**
Apache	0.64	**0.80**	0.03	**0.99**	0.03	**0.99**	0.26	**0.99**	0.73	**0.99**	0.73	**0.99**	0.28	**0.99**	0.97	**0.99**	**1.00**	0.99	0.98	**0.99**
Zookeeper	0.35	**0.81**	0.46	**0.82**	0.66	**0.80**	0.46	**0.82**	**0.84**	0.83	**0.84**	0.83	**0.82**	0.82	**0.85**	0.82	**0.91**	0.82	0.80	**0.82**
Hadoop	0.43	**0.80**	0.05	**0.78**	0.02	**0.67**	0.53	**0.83**	0.54	**0.83**	0.54	**0.83**	0.61	**0.79**	0.73	**0.79**	**0.83**	0.79	**0.87**	0.78
HealthApp	0.31	**0.91**	0.02	**0.93**	0.01	**0.37**	0.31	**0.92**	0.31	**0.95**	0.31	**0.96**	0.46	**0.97**	**1.00**	0.97	**0.99**	0.97	0.57	**0.96**
OpenStack	0.01	**0.65**	0.01	**0.84**	0.02	**0.87**	0.01	**0.97**	0.03	**0.73**	0.03	**0.81**	0.14	**0.98**	0.87	**0.98**	0.95	**0.98**	0.95	**0.98**
HPC	0.69	**0.86**	0.64	**0.89**	0.00	**0.13**	—	—	0.74	**0.96**	0.72	**0.97**	0.75	**0.87**	**1.00**	0.92	0.41	**0.90**	0.99	**1.00**
Mac	0.23	**0.49**	0.12	**0.56**	0.11	**0.54**	0.28	**0.59**	0.24	**0.56**	0.36	**0.60**	0.31	**0.56**	0.48	**0.56**	**0.59**	0.58	**0.63**	0.60
OpenSSH	0.05	**0.52**	0.03	**0.66**	0.14	**0.72**	—	—	0.36	**0.43**	0.59	**0.65**	0.53	**0.68**	0.71	**0.71**	**0.88**	0.72	**1.00**	0.72
HDFS	0.19	**0.76**	0.00	**0.92**	0.03	**0.94**	—	—	0.46	**0.94**	0.46	**0.94**	0.81	**0.94**	0.90	**0.94**	0.86	**0.94**	**1.00**	**1.00**
Spark	0.32	**0.79**	0.01	**0.82**	0.00	**0.53**	—	—	0.33	**0.81**	0.33	**0.81**	0.36	**0.93**	**0.96**	0.94	**0.96**	0.94	0.76	**0.90**
Thunderbird	0.06	**0.45**	0.08	**0.45**	0.06	**0.38**	—	—	0.25	**0.52**	0.28	**0.53**	0.25	**0.50**	0.49	**0.51**	**0.72**	0.54	0.55	**0.60**
BGL	0.02	**0.16**	0.35	**0.60**	0.25	**0.37**	—	—	0.43	**0.61**	0.46	**0.62**	0.47	**0.67**	**0.69**	**0.69**	0.13	**0.69**	**0.98**	**0.98**
Average	0.24	**0.65**	0.13	**0.78**	0.13	**0.64**	0.24	**0.75**	0.43	**0.76**	0.46	**0.80**	0.47	**0.80**	0.80	**0.82**	0.76	**0.80**	0.84	**0.86**

Table 3.4: Improvement in FGA. The columns with ✗ and ✓ indicate the results without and with SynLog+, respectively.

	LFA		LogCluster		SHISO		LogMine		AEL		Drain		UniParser		LogPPT		LLMParser		LILAC	
	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓
Proxifier	**0.40**	**0.40**	0.00	**0.21**	0.36	**0.69**	0.01	**0.47**	0.52	**0.55**	0.21	**0.54**	0.05	**0.43**	**0.87**	**0.87**	0.15	**0.46**	**1.00**	**1.00**
Linux	0.74	**0.82**	0.29	**0.73**	0.49	**0.61**	0.75	**0.85**	0.75	**0.86**	0.78	**0.86**	0.13	**0.76**	0.72	**0.77**	0.69	**0.81**	0.90	**0.90**
Apache	**0.87**	**0.87**	0.00	**0.84**	0.78	**0.83**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.26	**0.77**	0.68	**0.84**	0.73	**1.00**	**1.00**	**1.00**
Zookeeper	0.63	**0.71**	0.09	**0.84**	0.44	**0.62**	0.02	**0.85**	0.79	**0.90**	**0.90**	**0.90**	0.71	**0.88**	0.93	**0.98**	0.41	**0.98**	0.93	**0.96**
Hadoop	0.65	**0.82**	0.00	**0.77**	0.55	**0.70**	0.12	**0.91**	0.12	**0.92**	0.79	**0.95**	0.63	**0.91**	0.60	**0.88**	0.41	**0.89**	**0.93**	**0.93**
HealthApp	0.01	**0.81**	0.01	**0.46**	0.40	**0.55**	0.01	**0.73**	0.01	**0.86**	0.01	**0.97**	0.59	**0.96**	0.93	**0.96**	0.61	**0.94**	0.97	**0.99**
OpenStack	0.60	**0.62**	0.00	**0.84**	0.53	**0.84**	0.00	**0.96**	**0.68**	**0.68**	0.01	**0.71**	0.83	**1.00**	**1.00**	**1.00**	0.15	**1.00**	**1.00**	**1.00**
HPC	0.35	**0.44**	0.05	**0.10**	0.12	**0.15**	—	—	0.20	**0.27**	0.31	**0.41**	0.66	**0.71**	0.69	**0.86**	0.01	**0.02**	**0.96**	**0.96**
Mac	0.65	**0.70**	0.07	**0.66**	0.44	**0.67**	0.45	**0.84**	0.79	**0.81**	0.23	**0.68**	0.70	**0.79**	0.48	**0.70**	0.20	**0.60**	0.87	**0.89**
OpenSSH	0.51	**0.67**	0.01	**0.33**	0.61	**0.74**	—	—	0.69	**0.75**	**0.87**	**0.87**	0.02	**0.03**	0.14	**0.18**	0.37	**0.60**	**0.88**	**0.88**
HDFS	0.86	**0.91**	0.00	**0.88**	0.95	**1.00**	—	—	0.86	**0.95**	0.95	**1.00**	**1.00**	**1.00**	0.74	**0.95**	0.57	**0.90**	**1.00**	**1.00**
Spark	0.64	**0.79**	0.00	**0.23**	0.42	**0.54**	—	—	0.02	**0.85**	0.89	**0.91**	0.25	**0.62**	0.48	**0.83**	0.18	**0.72**	0.86	**0.89**
Thunderbird	0.71	**0.77**	0.01	**0.47**	0.56	**0.68**	—	—	0.59	**0.83**	0.85	**0.89**	0.78	**0.82**	0.73	**0.84**	0.32	**0.55**	0.64	**0.90**
BGL	0.56	**0.60**	0.06	**0.86**	0.43	**0.53**	—	—	0.74	**0.78**	0.79	**0.82**	0.87	**0.92**	0.90	**0.95**	0.01	**0.02**	**0.94**	**0.94**
Average	0.58	**0.71**	0.04	**0.59**	0.51	**0.65**	0.30	**0.78**	0.55	**0.79**	0.61	**0.82**	0.53	**0.76**	0.71	**0.83**	0.34	**0.68**	0.92	**0.95**

Table 3.5: Improvement in FTA. The columns with ✗ and ✓ indicate the results without and with SynLog+, respectively.

	LFA		LogCluster		SHISO		LogMine		AEL		Drain		UniParser		LogPPT		LLMParser		LILAC	
	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓
Proxifier	0.00	**0.40**	0.00	**0.21**	0.24	**0.69**	0.00	**0.47**	0.44	**0.55**	0.15	**0.54**	0.05	**0.43**	**0.87**	0.70	0.13	**0.31**	**1.00**	**1.00**
Linux	0.07	**0.71**	0.06	**0.61**	0.13	**0.49**	0.18	**0.70**	0.20	**0.71**	0.26	**0.73**	0.02	**0.64**	0.43	**0.65**	0.25	**0.67**	0.62	**0.74**
Apache	0.26	**0.66**	0.00	**0.69**	0.25	**0.67**	0.41	**0.76**	0.52	**0.76**	0.52	**0.76**	0.10	**0.62**	0.31	**0.69**	0.62	**0.76**	**0.83**	0.76
Zookeeper	0.18	**0.58**	0.05	**0.75**	0.17	**0.51**	0.01	**0.69**	0.47	**0.71**	0.61	**0.78**	0.48	**0.71**	0.78	**0.79**	0.38	**0.79**	**0.82**	0.79
Hadoop	0.10	**0.56**	0.00	**0.55**	0.06	**0.45**	0.06	**0.66**	0.06	**0.66**	0.38	**0.67**	0.47	**0.64**	0.46	**0.63**	0.28	**0.64**	**0.72**	0.66
HealthApp	0.00	**0.71**	0.00	**0.41**	0.04	**0.46**	0.00	**0.63**	0.00	**0.75**	0.00	**0.86**	0.27	**0.85**	0.83	**0.85**	0.52	**0.83**	0.82	**0.88**
OpenStack	0.08	**0.46**	0.00	**0.71**	0.09	**0.70**	0.00	**0.83**	0.17	**0.54**	0.00	**0.57**	0.26	**0.88**	0.83	**0.88**	0.14	**0.88**	**0.90**	0.88
HPC	0.20	**0.42**	0.03	**0.09**	0.00	**0.14**	—	—	0.13	**0.26**	0.14	**0.40**	0.33	**0.66**	0.53	**0.81**	0.00	**0.02**	0.89	**0.92**
Mac	0.08	**0.44**	0.02	**0.45**	0.08	**0.44**	0.11	**0.56**	0.20	**0.53**	0.07	**0.45**	0.26	**0.52**	0.29	**0.47**	0.10	**0.40**	0.56	**0.59**
OpenSSH	0.05	**0.58**	0.00	**0.31**	0.17	**0.69**	—	—	0.31	**0.71**	0.44	**0.80**	0.01	**0.03**	0.13	**0.17**	0.31	**0.54**	**0.85**	0.80
HDFS	0.03	**0.41**	0.00	**0.44**	0.40	**0.53**	—	—	0.52	**0.54**	**0.59**	0.53	**0.59**	0.53	0.40	**0.52**	0.30	**0.55**	**1.00**	**1.00**
Spark	0.09	**0.55**	0.00	**0.17**	0.12	**0.37**	—	—	0.01	**0.61**	0.46	**0.65**	0.14	**0.41**	0.35	**0.55**	0.15	**0.51**	**0.75**	0.63
Thunderbird	0.06	**0.49**	0.00	**0.30**	0.11	**0.40**	—	—	0.18	**0.52**	0.28	**0.56**	0.31	**0.51**	0.44	**0.53**	0.22	**0.35**	0.44	**0.58**
BGL	0.03	**0.50**	0.01	**0.72**	0.07	**0.40**	—	—	0.14	**0.68**	0.19	**0.68**	0.20	**0.75**	**0.79**	0.78	0.00	**0.02**	**0.90**	**0.90**
Average	0.09	**0.53**	0.01	**0.46**	0.14	**0.50**	0.10	**0.60**	0.24	**0.61**	0.29	**0.64**	0.25	**0.58**	0.53	**0.64**	0.24	**0.52**	**0.79**	**0.79**

3.3.2 RQ2.1: Effectiveness evaluation of SynLog+

Motivation SynLog+ is designed as a parser-agnostic refinement module intended to enhance template accuracy without modifying the internal logic of existing log parsers. To validate its general applicability and practical value, it is necessary to evaluate whether SynLog+ consistently improves accuracy across diverse parsing architectures. This RQ therefore assesses the effectiveness and robustness of SynLog+ when integrated with a range of benchmark log parsers.

Approach We apply SynLog+ to all of the benchmark log parsers and evaluate the efficacy of SynLog+ in improving their accuracy. First, we integrate SynLog+ with each baseline parser by treating the parser as a grouper that clusters raw log messages. Second, SynLog+ is applied to each resulting log group to refine the extracted templates.

Result As shown in Table 3.3, SynLog+ boosts the parsing accuracy across all baseline syntax-based log parsers. On average, it yields a 236% improvement in message-level log parsing accuracy of syntax-based log parsers. For semantic-based log parsers, the average boost is lower at 20.6%. This is because SynLog+ aims to lift the parsing accuracy close to the grouping accuracy by operating on the log groups. Since the parsing accuracy is already close to the grouping accuracy for the semantic-based log parsers, SynLog+ does not provide as high a boost as it provides the syntax-based log parsers for which the parsing accuracy is much lower than the grouping accuracy. Table 3.2, 3.4, and 3.5 presents the performance boost in GA, FGA, and FTA, respectively. SynLog+ achieves 17% and 181.5% increase in GA and FGA, respectively. For FTA, the syntax-based log parsers attain an average boost of 831.5% and the semantic-based parsers attain that of 67%.

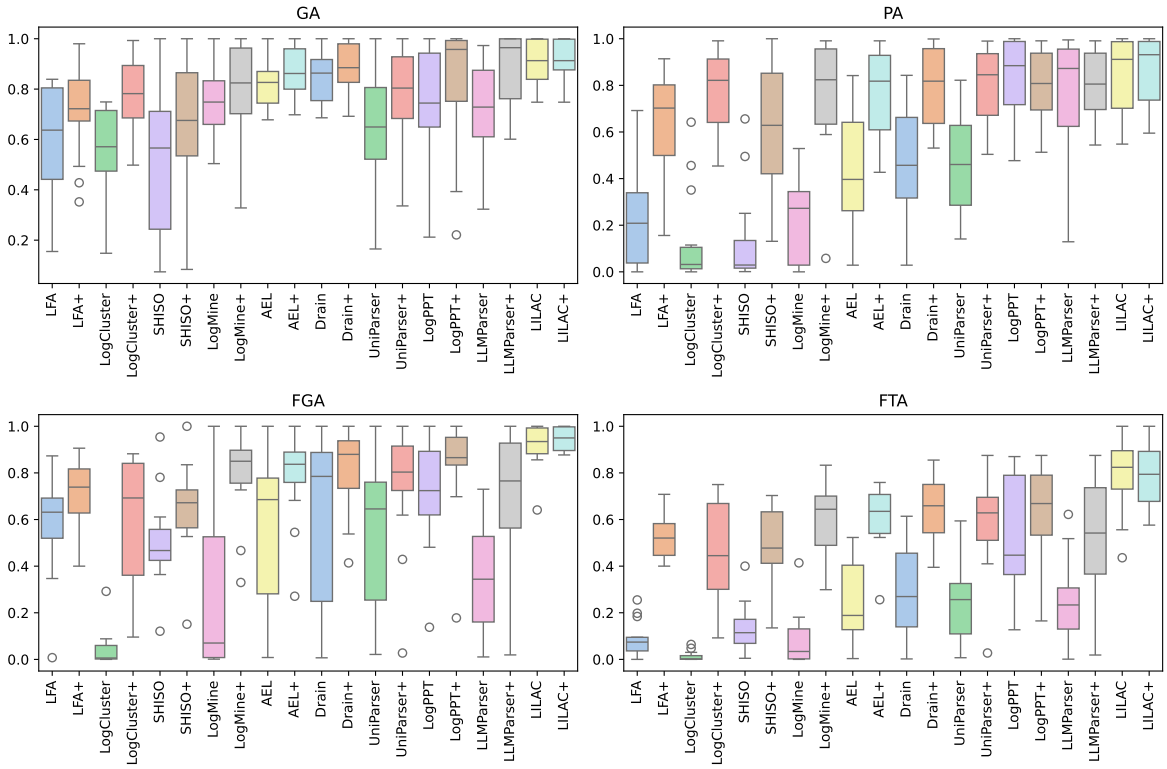


Figure 3.2: Robustness comparison of SynLog+ with benchmark syntax-based and semantic-based log parsers

Observation 2.1 On average, SynLog+ achieves 17%, 158%, 181%, and 554% improvement over benchmark log parsers in GA, PA, FGA, and FTA, respectively. The improvement is most noticeable in syntax-based log parsers where the gap between grouping accuracy and parsing accuracy are significant.

Along with the average accuracy, it is important to compare the robustness of the log parsers. If a log parser shows high variance across different log dataset, that is indicative of low robustness against different logging formats. Figure 3.2 shows that not only does SynLog+ improves accuracy across for all baselines across all four metrics, it also reduces the variance in the accuracy distribution, demonstrating the robustness of SynLog+ across different log types. Specifically, SynLog+ improves the median GA, PA, FGA, and FTA by 11.4%, 34.7%, 27.6%, and 35.7%, respectively, over the benchmark log parsers.

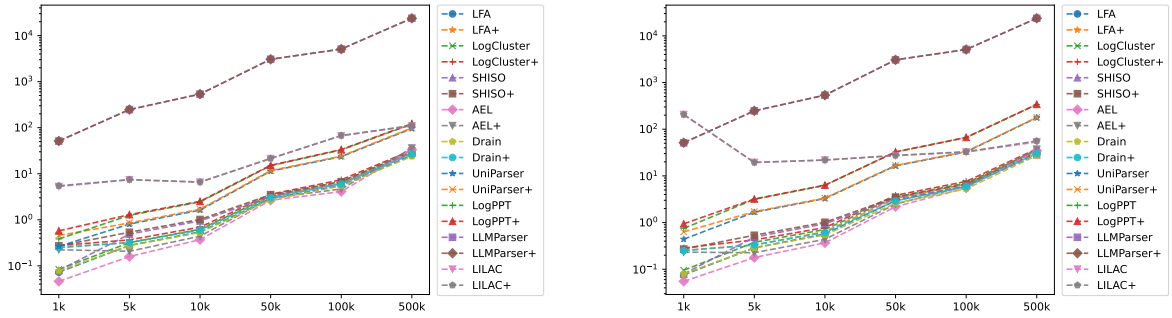
Observation 2.2 SynLog+ raises the median of parsing accuracies by 35% and that of grouping accuracies around 11% to 27% over the benchmark log parsers, demonstrating improvement in robustness across diverse log types.

3.3.3 RQ2.2: Efficiency evaluation of SynLog+

Motivation Besides accuracy, efficiency is another critical metric for log parsers to consider in order to handle large-scale log data. It is essential to evaluate the efficiency of SynLog+ to properly evaluate its applicability in large-scale log parsing.

Approach To evaluate the efficiency of SynLog+, we compare its execution time for all baseline log parsers except LogMine since it is unable to finish parsing these datasets within a reasonable time limit. Following RQ1.2, we evaluate the runtime efficiency using four datasets, namely HPC, OpenSSH, BGL, and HDFS.

Result Figure 3.3 shows the runtime of the log parsers on the four selected datasets on a logarithmic scale. The figure shows that syntax-based log parsers require significantly smaller execution time compared to the inference time of the semantic-based log parsers. Specifically, semantic-based parsers require on average 6,091s runtime and syntax-based parsers require 31.46s runtime on average for 500k log lines. Additionally, semantic-based log parsers require training costs, which are not considered in this efficiency evaluation. When SynLog+ is coupled with the syntax-based log parsers, it incurs an additional runtime cost ranging from 1.94s to 3.54s for 500k log lines, and 2.87s on average. Specifically, as shown by the Figure 3.3, with the added runtime requirement SynLog+, it is much more efficient than the benchmark semantic-based log parsers.



(a) HPC dataset

(b) OpenSSH dataset

(c) BGL dataset

(d) HDFS dataset

Figure 3.3: Runtime efficiency of SynLog+ when coupled with three of the best-performing syntax-based log parsers compared to benchmark syntax- and semantic-based log parsers

Observation 2.3 SynLog+ introduces an additional runtime cost of only 2.87s on average per 500k logs to the benchmark log parsers, which is a negligible addition to the 31.46s average runtime for syntax-based parsers and 6,091s average runtime for semantic-based parsers.

3.3.4 RQ2.3: Sensitivity analysis of SynLog+

Motivation To increase efficiency, SynLog+ samples a set of representative logs to identify the constants and variables instead of analyzing all the logs containing n logs in a log group. A higher value of n would increase runtime, but a lower value may decrease parsing accuracy. In this RQ, we conduct a sensitivity analysis to analyze the impact of different values of n in the log sampling module of SynLog+.

Table 3.6: Sensitivity Analysis

Dataset	k = 2				k = 3				k = 6			
	GA	PA	FGA	FTA	GA	PA	FGA	FTA	GA	PA	FGA	FTA
Proxifier	0.69	1.00	0.54	0.54	0.69	1.00	0.54	0.54	0.69	1.00	0.54	0.54
Linux	0.71	0.63	0.86	0.73	0.71	0.63	0.86	0.73	0.71	0.63	0.86	0.73
Apache	1.00	0.99	1.00	0.76	1.00	0.99	1.00	0.76	1.00	0.99	1.00	0.76
Zookeeper	0.99	0.83	0.90	0.78	0.99	0.83	0.90	0.78	0.99	0.83	0.90	0.78
Hadoop	0.94	0.83	0.95	0.67	0.94	0.83	0.95	0.67	0.94	0.83	0.95	0.67
HealthApp	0.99	0.96	0.97	0.86	0.99	0.96	0.97	0.86	0.99	0.96	0.97	0.86
OpenStack	0.82	0.81	0.71	0.57	0.82	0.81	0.71	0.57	0.82	0.81	0.71	0.57
HPC	0.84	0.97	0.41	0.40	0.84	0.97	0.41	0.40	0.84	0.97	0.41	0.40
Mac	0.90	0.60	0.68	0.45	0.90	0.60	0.68	0.45	0.90	0.60	0.68	0.45
OpenSSH	0.71	0.65	0.87	0.80	0.71	0.65	0.87	0.80	0.71	0.65	0.87	0.80
HDFS	1.00	0.94	1.00	0.53	1.00	0.94	1.00	0.53	1.00	0.94	1.00	0.53
Spark	0.87	0.81	0.91	0.65	0.87	0.81	0.91	0.65	0.87	0.81	0.91	0.65
Thunderbird	0.87	0.53	0.89	0.56	0.87	0.53	0.89	0.56	0.87	0.53	0.89	0.56
BGL	0.91	0.62	0.82	0.68	0.91	0.62	0.82	0.68	0.91	0.62	0.82	0.68
Average	0.86	0.82	0.79	0.65	0.86	0.82	0.79	0.65	0.86	0.82	0.79	0.65

Approach We evaluate the performance of SynLog+ combined with Drain with $k = 2$, $k = 3$, and $k = 6$, where k is the number of representative logs selected from each group.

Result As presented in Table 3.6, the experiments resulted in the same accuracy scores across all four metrics. This demonstrates that there is no gain in accuracy by increasing the number of representative logs.

Observation 2.4 Increasing the number of representative logs from 2 to 3 or 6 does not result in an increased accuracy.

3.3.5 RQ2.4: Effectiveness of SynLog+ in identifying invariant variables

Motivation In RQ1.7, we analyzed the impact of invariant variables on the parsing accuracy of the benchmark log parsers. In this RQ, we analyze the effectiveness of SynLog+ in identifying invariant variables as variables when combined with the benchmark log parsers

Table 3.7: Number of invariant variable positions misidentified as a constant by the log parser. The columns with ✗ and ✓ indicate the results without and with SynLog+, respectively.

	LFA		LogCluster		SHISO		LogMine		AEL		Drain		UniParser		LogPPT		LLMParser		LILAC	
	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓
Apache	5,941	45	12,994	72	12,864	72	12,994	72	12,864	72	12,864	72	6,204	72	45	45	2	2	0	0
BGL	22,845	12	178,628	2,163	178,628	8,410	29,194	11,097	111,169	12	75,936	16	6,260	14,449	2,281	2,281	178,483	14,450	2	2
Hadoop	20,630	2,041	45,871	6,247	45,602	6,523	40,499	7,294	45,068	6,246	45,080	6,246	19,387	6,312	1,226	1,226	2,120	2,120	904	904
HDFS	0	0	31	0	12	0	21	0	11	0	2	0	7	0	0	0	3	0	0	0
HealthApp	12,909	3,574	29,870	2,830	29,870	1,260	14,381	3,963	24,595	1,246	28,617	2,830	6,684	2,830	5	5	272	272	33	33
HPC	100	0	108	0	108	0	—	—	106	0	108	0	50	0	0	0	2	2	1	0
Linux	4,517	1,181	7,625	1,299	7,559	841	4,066	1,209	4,324	1,291	7,104	1,199	4,045	1,297	1,160	1,160	807	807	61	61
Mac	18,479	9,852	82,417	12,582	76,320	12,714	61,252	11,084	49,858	10,978	57,349	11,884	24,463	12,250	4,810	4,810	13,633	13,633	7,580	7,580
OpenSSH	636,445	177,667	976,454	177,850	834,459	177,667	—	—	573,204	177,667	573,204	177,667	723,742	177,851	37,362	37,362	0	0	0	0
OpenStack	6,788	1,747	143,481	5,877	2,073	1,747	143,481	1,747	8	1,747	8	1,747	5,225	1,747	0	0	15	15	0	0
Proxifier	206	0	11,266	0	362	0	11,266	0	113	0	329	0	0	0	0	0	0	0	0	0
Spark	13,334	5,634	28,835	6,827	27,975	5,636	—	—	24,328	5,634	23,488	5,634	3,599	5,644	63	63	1,833	1,833	333	333
Thunderbird	152,092	62,530	396,715	87,557	390,319	68,636	—	—	284,926	65,340	289,420	67,470	93,099	72,766	51,158	51,158	73,224	73,224	40,807	40,807
Zookeeper	10,687	10,443	11,768	10,421	11,688	10,505	11,758	10,637	11,574	10,623	11,250	10,443	626	10,637	10,486	10,486	0	0	10,424	10,424
Average	64,641	19,623	137,576	22,409	115,560	21,001	32,891	4,710	81,582	20,061	80,340	20,372	63,814	21,847	7,757	7,757	19,314	7,597	4,296	4,296

Approach First, we capture the invariable variables from the ground truth log groups and validate if the parsed templates mark those positions as variables. If they mark it as constant, we consider it as an inaccuracy due to invariant variables.

Result Table 3.7 presents the number of variable positions in the benchmark datasets with the same value of the variables that are incorrectly identified as constants by the respective log parsers, standalone and with SynLog+. In the cases of syntax-based log parsers, SynLog+ reduces the inaccuracy by 78.5% on average. For semantic-based parsers, the average improvement reduces to 30%. Specifically, for LogPPT and LILAC there is no improvement. These results suggest that semantic-based parsers are able to resolve the inaccuracies due to invariant variable that are seen in syntax-based parsers. SynLog+ reduces 59.7% on average and 78.5% of syntax-based parsing inaccuracies due to invariant variables.

Observation 2.5 On average, SynLog+ resolves 78.5% of variables misidentified by syntax-based log parsers due to their invariant values, indicating its effectiveness in identifying invariant variables in log groups.

3.4 Conclusion

Based on the findings of the empirical study presented in Chapter 2, this chapter presents SynLog+, a lightweight regex-based template refinement module which refines the log tem-

plate from the log groups clustered by an existing log parser. It operates as a post-group refinement stage, targeting cases where templates are incorrectly inferred despite correct grouping.

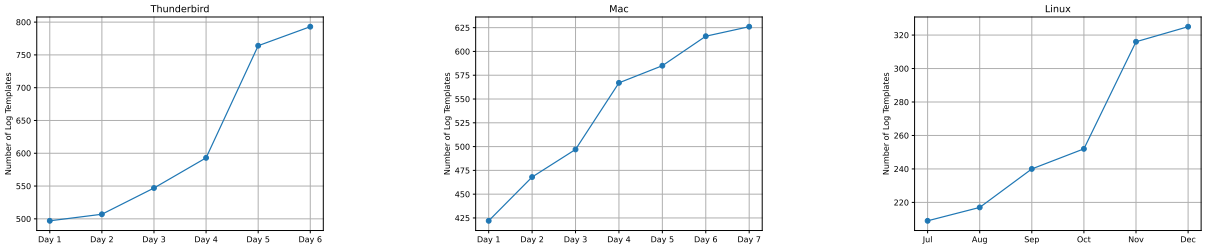
Chapter 4

GeLT: Diagnosing and Optimizing Generalizable Log Parsing with Test-time Training

4.1 Introduction

The aim of log parsing is to convert the raw log data into log templates by identifying the constant and variable tokens. For instance, in Figure 1.1, the log content "*Reading broadcast variable 11 took 15 ms*" is parsed into the log template "*Reading broadcast variable <*> took <*> ms*" with template parameters *11* and *15*, where the parameters are the values of the variables in the log template.

Syntax- vs Semantic-based Log Parsing. The existing log parsers roughly fall into two categories: a) syntax-based parsers [10, 17], which uses syntactic and global information to cluster logs of the same template, and b) semantic-based parser [11, 25], which uses a semantic model to classify the log tokens as constants or variables. The strengths of syntax-based parsers lie with their efficient usage of clustering techniques, but that becomes a weakness when logs in a clusters have the same variable values, resulting in inaccurate



(a) Thunderbird (b) Mac (c) Linux

Figure 4.1: Emergence of new log templates

template identification. Semantic parsers are designed to address this limitation, by learning of log representations automatically from training data. Indeed, semantic-based parsers have achieved remarkable accuracy in recent years and are now generally considered more effective than syntax-based parsers [15, 29].

Generalizability of Semantic-based Log Parsers. However, semantic-based parsers are not generalizable in parsing logs generated by real-world systems where the logging statements in the source code change over time and as a result the generated logs introduce new log templates [26]. Since the real-world software systems are constantly evolving and new log templates are introduced in the source code, as shown in Figure 4.1, the generalization of log parsing techniques over unseen or new log templates is a major concern [16, 21].

Test Time Training for Generalizable Semantic-based Log Parsing. Existing domain adaptation techniques for neural models fall under two types: domain generalization (DG) and unsupervised domain adaptation (UDA) [33]. DG aims to improve generalization across all domain by learning domain-independent representations [34], whereas UDA adapts the model trained on prior domains to new domain [35]. Since domains in log parsing generalization refers to log templates, we cannot eliminate domain-specific representations. Hence, recent paper by Yu et al. [26] adopted an UDA technique, test-time training (TTT) [36] to improve generalization in semantic-based log parsing. TTT, which updates the model parameters on the test data during test time, is uniquely suited to log parsing because it does not require the new log templates to share common features with historical log data [26].

The standard TTT, proposed by Sun et al. [36], is a method for improving the performance of predictive models when test data comes from a different distribution from the training data. Unlike conventional learning settings, where model parameters remain fixed during inference, TTT allows a model to adapt using unlabelled test data by optimizing a self-supervised auxiliary task at test time. Originally, TTT is applied in computer vision tasks such as object recognition. In Log3T, Yu et al. applied this technique in log parsing task to achieve generalization over new log types by updating the parameters of the underlying transformer encoder model with unseen logs. The auxiliary task in Log3T is designed as a token classification task, where a randomly chosen token of the original log is replaced with a variable from historical logs and the model learns to predict the label of the replaced token.

Contribution 1. An Empirical Study of the Effectiveness of TTT in Log3T. Although test-time training has shown promise in mitigating distribution shifts in vision and NLP tasks [36], its effectiveness and reliability for log parsing remain poorly understood. In particular, existing approaches like Log3T assume that self-supervised signals generated at test time are sufficiently accurate to improve model generalization—an assumption that has not been empirically validated in the log parsing domain. In this chapter, we conducted a large-scale empirical study of the effectiveness of existing TTT approach, Log3T, for generalizable log parsing. Our investigation finds two issues in the test-time training technique of Log3T as shown in Figure 4.2. Specifically, the auxiliary self-supervised task generates (a) unbalanced and (b) incorrectly-labelled mutated logs to train the model during test time, causing the performance of the model to deteriorate rather than improve. GeLT addresses these issues by (a) generating mutated logs from both historical variables and historical constants, and (b) optimizing only the loss for the replaced tokens whose labels are known.

Contribution 2. GeLT—An Optimized TTT-based Approach for Generalizable Log Parsing. To address the limitations in Log3T, we have designed GeLT, an optimized test-time training approach for generalizable log parsing, which addresses the aforementioned design flaws of Log3T and proposes an optimized mutated log generation strategy and auxil-

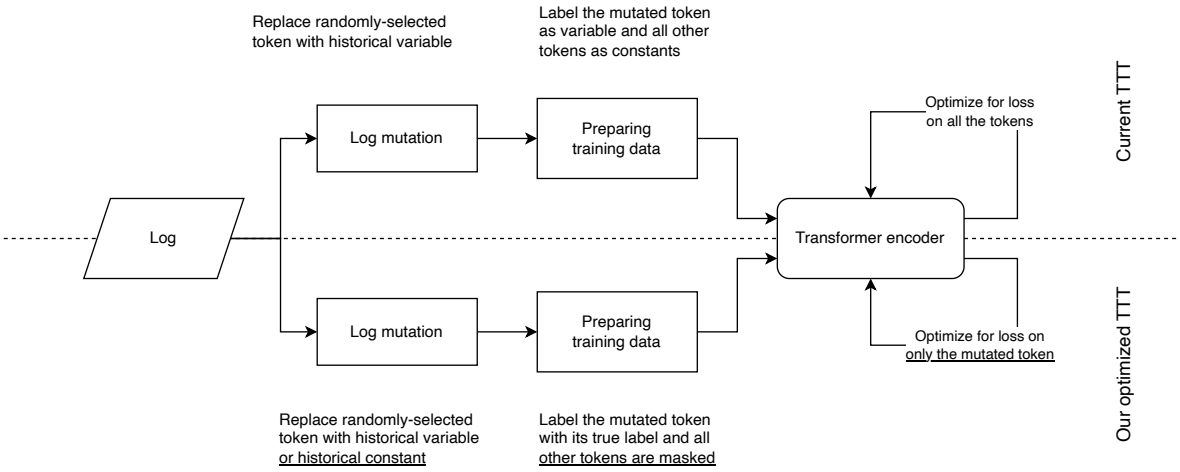


Figure 4.2: Generalizable log parsing with test-time training: unoptimized (top) and optimized (bottom)

iary self-supervised task. Specifically, GeLT uses both historical variables and constants for mutated log generation, and it finetunes the model only on the replaced token, so that no assumption regarding the labels of the other tokens is required GeLT improves F1 grouping accuracy (FGA) in 3 and F1 template accuracy (FTA) in 2 datasets out of 16 benchmark datasets, compared to Log3T's improvement of FGA in 2 datasets and FTA in 0 datasets. Moreover, Log3T shows a drop in FGA and FTA in 5 and 4 datasets respectively, whereas GeLT does not suffer a drop in performance in any dataset.

4.2 Background

4.2.1 Generalizable Log Parsing

As summarized in Table 4.1, studies have defined generalization in various ways. In this section, we briefly present the existing concepts.

Generalization across datasets. This type of generalizability is evaluated as robustness, i.e., the variance of accuracy achieved by the semantic-based parser across all the benchmark datasets. In evaluating syntax-based parsers, researchers noted their dependence of dataset-

Table 4.1: Generalization of log parsers according to the type of generalizability considered as a criteria in their evaluation.

	Generalization		
Log Parser	Across datasets	Across models	Over unseen log templates
NuLog	✓	✗	✗
UniParser	✓	✗	✗
LogBatcher	✓	✗	✗
LogPPT	✓	✓	✗
LLMParser	✓	✓	✗
AdaParser	✓	✓	✗
LILAC	✓	✓	✗
Log3T	✓	✓	✓

specific hyperparameters as a motivation for pursuing semantic-based parsers. The latter do not rely on handcrafted rules for each dataset but instead learn the dataset-specific representations automatically from a subset of training data. As such, it is important to evaluate its robustness across different datasets along with the average accuracies. Some studies have defined this quality as generalization ability [27, 19, 13].

Generalization across models. This type of generalizability is defined as the effectiveness of a semantic-based log parsing technique across a wide variety of machine learning (ML) or large language models (LLM). Specially for LLM-based log parsers, this is a major factor to consider since LLMs vary widely in terms of their parameter sizes, pretraining dataset, etc. Some studies proposing or evaluating LLM-based log parsers have defined this quality as generalizability [16, 13].

Generalization over new log templates. This type of generalizability is defined as the robustness of a semantic-based log parser in unseen log templates. It is a well-known phenomena in ML that neural models trained on a set of training data perform worse on test data if the test data comes from a different distribution or domain. In case of log parsing, unseen log templates constitute as different distribution. Many studies proposing semantic-

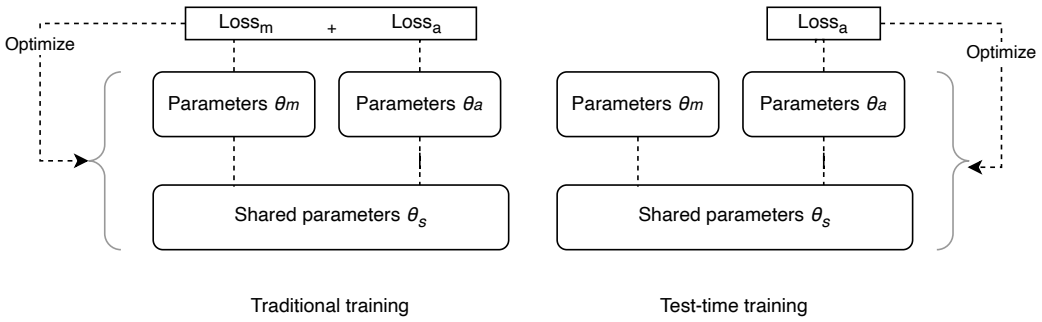


Figure 4.3: Test-time training

based parsers defined this quality as generalizability [16, 26]. In this study, we focus on generalization over new log templates.

4.2.2 Test-time Training for Generalizable Log Parsing

The challenge of generalization over unseen log templates is a specific instance of a larger challenge in supervised learning. When test data comes from a different distribution than the training data, the predictive accuracy of ML models drop. Prior works in this area has explored two approaches: domain generalization (DG) and unsupervised domain adaptation (UDA). DG aims to learn a classification model from multiple source domains and generalize it to unseen target domains, whereas UDA aims to adapt a pretrained classification model to a new domain.

Although DG appears to be more appealing in general, it is not applicable to log parsing, because it functions by removing domain-specific representations, which a log parsing model must learn. As such, UDA is the practical choice to achieve generalizability in log parsing. Several techniques have been proposed for UDA, such as CODA [35], CrossGrad [34], and TTT [36]. Since TTT does not require training data and test data to share common features, it is uniquely suited to log parsing. Hence, Yu et al. adopted TTT for generalizable log parsing [26].

In test-time training, a single test sample is transformed into a self-supervised learning problem, called the auxiliary problem, on which the model parameters are updated before

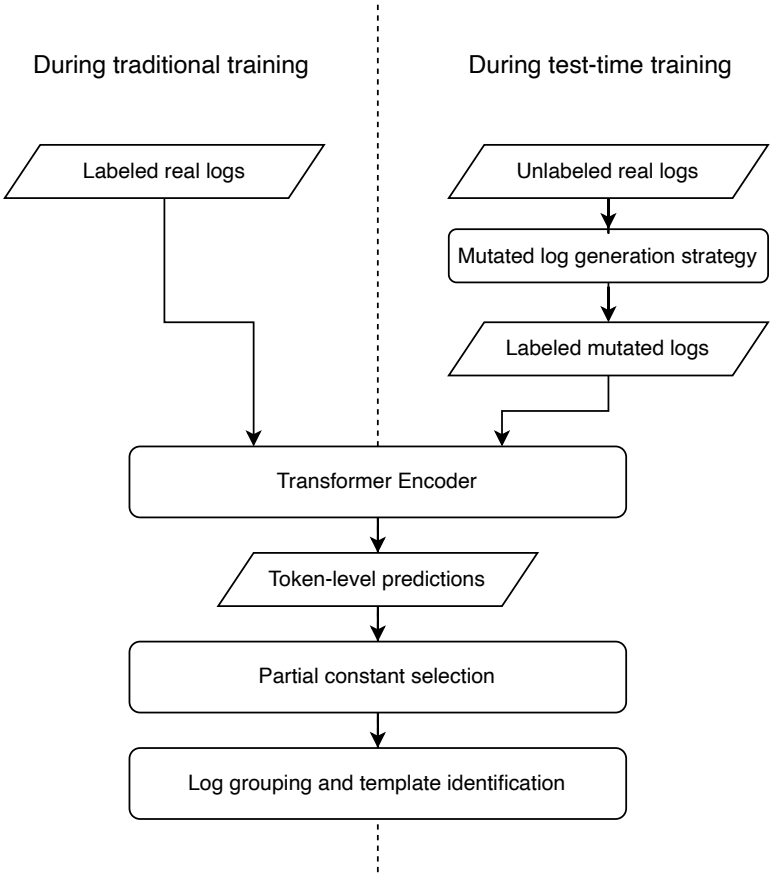


Figure 4.4: Workflow of Log3T

making prediction on the original test sample. The auxiliary task essentially generates labels for the unlabelled test samples. For example, for image classification benchmarks, Sun et al. use the auxiliary task of rotating each test image by a multiple of 90 degrees and predicting its angle [36].

Figure 4.3 presents traditional training and test-time training side-by-side. In standard test-time training, the auxiliary task shares some of the model parameters, denoted by θ_s, with the main task. Both the main task and the auxiliary tasks have their task-specific parameters, denoted by θ_m and θ_a respectively. During offline training, the model is trained on both tasks on the same training data. During inference, the model is trained only on the auxiliary task, i.e., only θ_a and θ_s are updated. The model then makes a prediction using

the updated parameters.

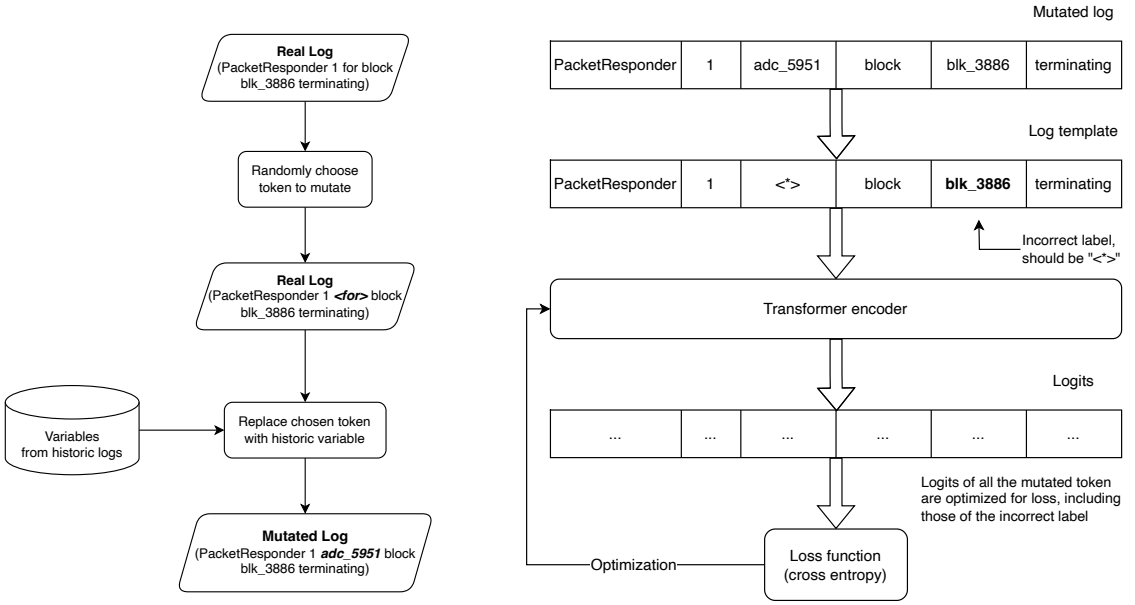
Figure 4.4 presents the log parsing workflow of Log3T. It uses a transformer encoder model to select n tokens least likely to be variables, where n is a dataset-specific hyperparameter. To do this, the model generates predictions for each token, where the predictions represent the model's confidence on the token being a variable. Hence, to select n tokens least likely to be variables, Log3T chooses the n tokens with the least confidence predicted by the model. These tokens are called *partial constants*. Logs with the same partial constants appearing in the same order are grouped together. Categorizing the common tokens among all the logs of a group to be constants, the log template of the groups are identified.

As shown in Figure 4.4, the model is first trained with traditional training methods on the available labelled training log data. During inference, to improve the generalization ability, Log3T employs test-time training to update the model parameters on the test log samples. The data for the auxiliary task is generated by mutated log generation strategy, where a randomly chosen token is replaced by a variable from historical logs. In Log3T, this mutated token is labelled as a variable and the other tokens are labelled as constants.

4.3 Case Study of Test-time Training for Generalizable Log Parsing

4.3.1 Motivation

A fundamental consideration in the evaluation of neural network models is their ability to generalize over unseen data. In the context of log parsing, this capability is particularly important, as real-world log streams frequently evolve over time and introduce previously unseen log templates [21, 11, 12, 16, 26]. Only recently has this gap begun to be addressed with the introduction of Log3T by Yu et al., which explores the use of test-time training to improve generalization in semantic-based log parsers. Their work highlights the potential of



(a) Mutated log generation strategy (b) Test-time training strategy with the mutated log

Figure 4.5: Auxiliary task for test-time training in Log3T

adapting models at inference time to improve parsing accuracy of unseen templates, making test-time training an important consideration for future syntax-based log parsing systems. However, there are some concerns regarding the evaluation criteria and the design choices of Log3T that calls for a re-evaluation of its effectiveness. In this subsection, we pose four research questions to address them.

Effectiveness of TTT in generalizable log parsing The evaluation of Log3T was restricted to only grouping accuracy (GA) and edit distance (ED), overlooking template-level metrics such as F1-score of group accuracy (FGA) and F1-score of template accuracy (FTA). When the objective of evaluation is to measure performance on unseen templates, template-level metrics are essential for providing a more faithful assessment of parsing quality. Message-level metrics such as GA and ED may underestimate the error in unseen log data if the frequency of the log templates are low. To address this limitation, we pose the following research question and re-evaluate the test-time training design of Log3T with appropriate

set of accuracy metrics.

RQ3.1. How effective is test-time training in generalizing over unseen log templates?

Reliability of the log mutation strategy The design choices of Log3T raises concerns about the potential of failure. Specifically, the log mutation strategy replaces a randomly-chosen token with a historical variable, while the other tokens are assumed to be constants. This assumption may not hold true. When false, finetuning on the mutated log data will cause the model to underperform because of mislabelling. To investigate the correctness of the mutated log generation strategy, we pose the following research question.

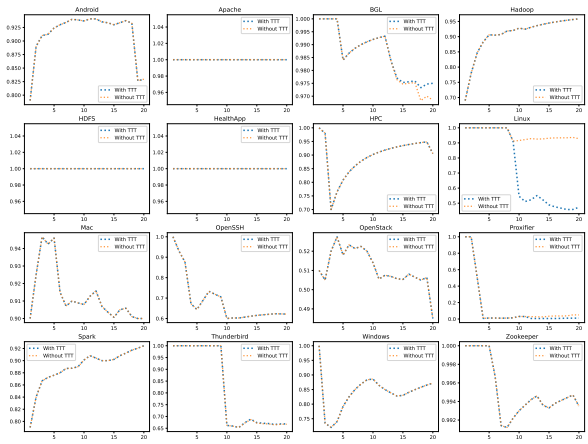
RQ3.2. How reliable is the mutated log generation strategy of Log3T?

Effectiveness of TTT in token-level classification The auxiliary self-supervised task in Log3T aims to improve log parsing generalization by improving the accuracy of the underlying model to classify tokens of unseen logs. Hence, it is important to evaluate the effectiveness of TTT in improving the token-level classification ability of the model along with the overall log parsing accuracy.

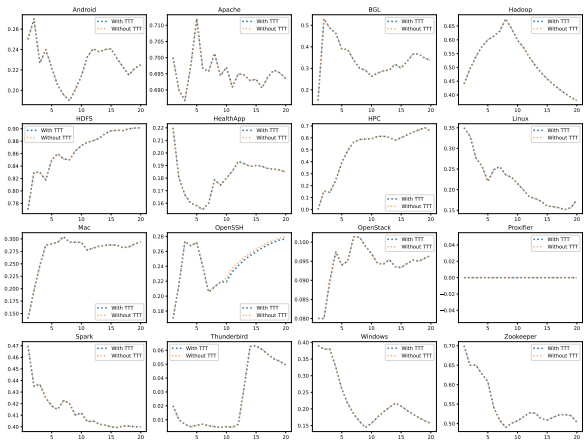
RQ3.3. How effective is TTT in improving token-level classification accuracy?

Effectiveness of TTT across other encoder architectures Lastly, the effectiveness of test-time training in Log3T is evaluated only with a BERT encoder model. To address external validity, evaluation of its effectiveness with other encoder architectures is also necessary. In this study, we evaluate the effectiveness of test-time training with T5 architecture to answer the following research question.

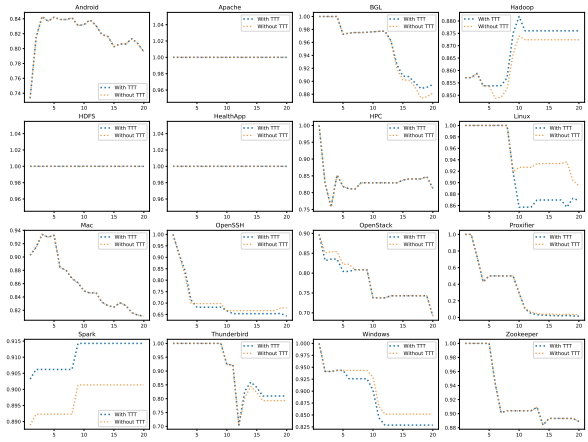
RQ3.4. Is the effectiveness of TTT in generalizable log parsing demonstrable across other encoder architectures?



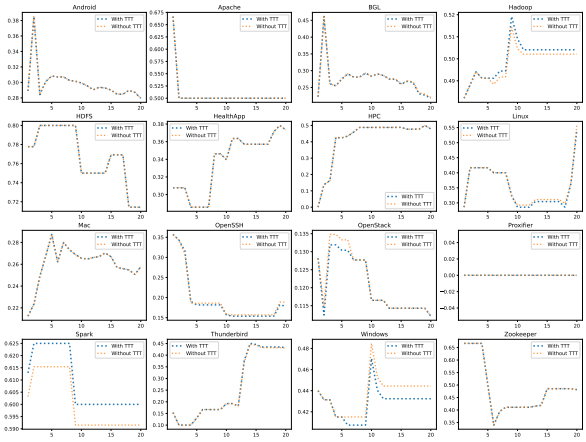
(a) Grouping accuracy (GA)



(b) Parsing accuracy (PA)



(c) F1-score of grouping accuracy (FGA)



(d) F1-score of template accuracy (FTA)

Figure 4.6: Accuracy of Log3T with and without test-time training

4.3.2 Study Setup

Dataset We conduct our experiments on the Loghub-2k [14] benchmark published by He et al. It contains 16 datasets, each containing 2,000 logs from various domains such as distributed systems, operating systems, mobile systems, supercomputers, and server applications. However, recent studies have identified incorrectly labelled logs in Loghub-2k [21, 28]. Following recent studies [11], we use the corrected benchmark published by Khan et al. [28].

4.3.3 RQ3.1. Effectiveness of test-time training in log parsing

In this RQ, we examine the effectiveness of the test-time training employed by Log3T by evaluating the log parsing accuracy.

Approach To illustrate the effectiveness of test-time training in log parsing, Log3T compared the grouping accuracy (GA) with and without test-time training. There are two issues with that: 1) GA only accounts for grouping abilities of a log parser, whereas it is also necessary to evaluate the correctness of the identified templates; and 2) GA is a message-level metric, and since we are evaluating generalization over new log types, we should use template-level metrics, e.g., FGA and FTA. Otherwise, if the new log types account for fewer logs, they would not have enough impact on the GA and PA to adequately reflect the effectiveness of test-time training.

To address the aforementioned issues, we compare the accuracy of Log3T with and without test-time training across all four metrics described in Section 2.3.2. The baseline is the same Log3T model trained on the real annotated logs of the first batch where no test-time training is applied. Following Log3T [26], we divide the datasets into equal batches each of size 100 to simulate the introduction of new log types over time. The first batch serves as the system's historical logs and the later batches are newly arrived logs.

Results Figure 4.6 presents the log parsing accuracy over the 20 batches ($= 2,000 \div 100$). across four metrics with and without applying test-time training (TTT). Among 16 datasets, test-time training improves GA in 1 dataset, FGA and FTA in 2 datasets. On the other hand, it reduces GA in 2, FGA and FTA in 4 datasets. It is clear that the test-time training strategy employed in Log3T does not consistently and robustly improve the baseline of Log3T with traditional offline training.

The improvement in GA and FGA in BGL dataset is not seen in PA or FTA. This means that TTT improves the grouping ability of the parser, but not the parsing accuracy. Although grouping is done based on the partial constants selected by the transformer encoder model,

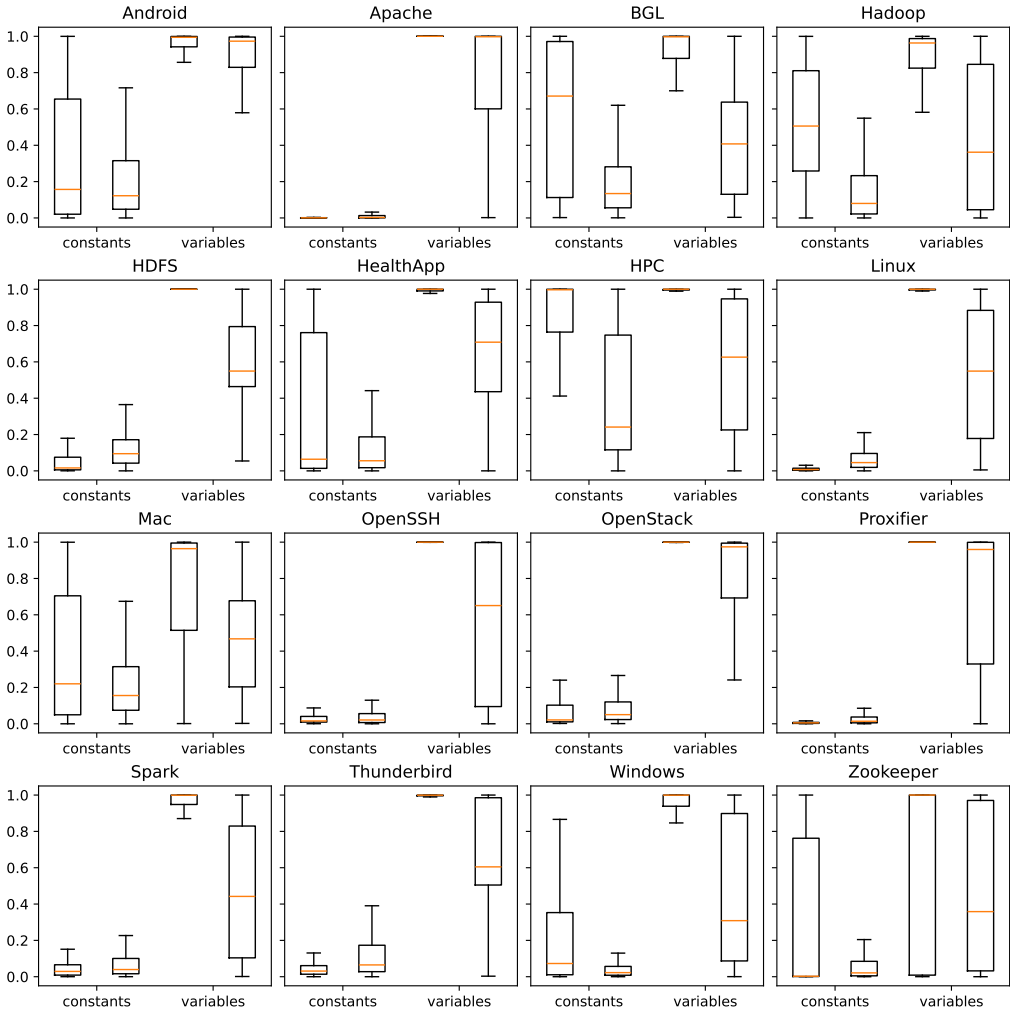


Figure 4.7: Distribution of token prediction probabilities from the encoder model with and without test-time training of Log3T. For both constant and variable tokens, two boxplots are shown side by side comparing predictions without test-time training (left) and with test-time training (right).

Observation 1. Test-time training strategy employed by Log3T increases log parsing accuracy for 2 datasets and decreases it for 4 out of 16 benchmark datasets.

4.3.4 RQ3.2. Accuracy of mutated log generation strategy

For the auxiliary self-supervised task, Log3T generates mutated log data for training during test time. As shown in Figure 4.5, Log3T replaces a randomly chosen token from a test sample with a variable from historical log data. This replaced token is identified as a variable. The other tokens are assumed to be constants and are labelled so. During test-time training, the transformer encoder model trains on this mutated log data with potential incorrect labels.

Table 4.2: Percentage of incorrectly-labelled tokens by Log3T

Dataset	Incorrect labels (%)	Dataset	Incorrect labels (%)
HDFS	67.24	Windows	42.99
Hadoop	81.42	Linux	57.42
Spark	48.52	Android	38.37
Zookeeper	34.97	HealthApp	65.23
BGL	35.14	Apache	22.67
HPC	45.90	OpenSSH	38.38
Thunderbird	31.57	OpenStack	49.01
Proxifier	48.64	Mac	39.58

In this RQ, we examine the accuracy of the pseudo labels generated by Log3T for test-time training. Specifically, for all 16 datasets in the Loghub-2k benchmark, we determine the correctness of the labels of all the unmutated tokens in the mutated log data.

Approach For each dataset, we collect the mutated log data generated by Log3T. For each mutated log, we collect all the unmutated tokens, i.e., the tokens that have not been replaced. These tokens are assumed to be constants by Log3T. Comparing with the groundtruth data, we determine the correctness of this assumption.

Results Table 4.2 shows that on average 46.7% of the tokens that are assumed to be constant by Log3T are incorrectly labelled. Specifically, for HDFS and Hadoop datasets, 81.42% and 67.24% of the tokens are mislabelled. The lowest percentage of mislabelling is in Apache dataset at 22.67%. This high percentage of mislabelled data, when used for test-time training, will degrade the model's accuracy rather than improving it.

Observation 2. The assumption made by Log3T regarding the unlabelled tokens while generating mutated log data does not hold by the benchmark datasets. On average, 46.7% of the tokens assumed to be constants are mislabelled.

Investigating the cause of it, we found that the datasets containing more numbers in their tokens than non-numeric tokens tend to have high percentage of incorrect labels. For

example, In HDFS and Hadoop datasets, the percentage of tokens that are numbers are 25.7% and 22.6%, the two largest ratios of numeric tokens among the 16 benchmark datasets. On other hand, for Apache, the percentage is 5.3%. Intuitively, numeric values are likely to be variables, and Log3T's assumption that all unreplaced tokens are constants do not hold for them and thus may drive the percentage of incorrect labels.

Observation 3. The ratio of incorrectly-labelled tokens can be explained by the ratio of numeric tokens in the dataset. Higher percentage of numeric tokens (25.7% in HDFS) lead to higher percentage of incorrect labels (81.4%) and vice versa.

4.3.5 RQ3.3. Effectiveness of test-time training in token-level classification

In this RQ, we investigate how effective test-time training is in token-level classification. From RQ3.2 we learned that the mutated log generation strategy of Log3T generates incorrectly-labelled data. In this RQ, we investigate the effect of that in token-level classification accuracy.

Approach For this RQ, we conduct the experiment in two settings: with and without test-time training. While running Log3T, we collect the probabilities of the tokens from the transformer encoder model. First, we create a boxplot of the probabilities of the constants and variables separately. The probabilities of the constants should be close to zero and those of the variables should be close to one. The larger the difference between the probabilities of constants and variables, the better the token-level classification ability of the model.

Besides visualization, to quantify the classification ability, we calculate the difference the of the average probabilities of constant and variable tokens. The higher the difference, the better the clusters, and the better the classification ability.

Results Figure 4.7 show that in none of the 16 datasets does test-time training enlarges the gap between the probabilities of constants and variables. For 8 datasets, the probabilities

Table 4.3: Difference between averages of constants and variables

Dataset	With TTT	Without TTT
Android	**0.62**	0.4785
Apache	0.75	**0.96**
BGL	0.21	**0.27**
HDFS	0.50	**0.84**
HPC	**0.19**	0.12
Hadoop	0.28	**0.31**
HealthApp	0.51	**0.59**
Linux	0.44	**0.76**
Mac	0.23	**0.37**
OpenSSH	0.51	**0.77**
OpenStack	0.71	**0.82**
Proxifier	0.65	**0.72**
Spark	0.39	**0.67**
Thunderbird	0.53	**0.80**
Windows	0.38	**0.72**
Zookeeper	0.37	**0.39**
Average	0.45	**0.60**

of of constants and variables move further away from 0 and 1 respectively, signifying poorer clustering or classification ability. For 6 datasets, the probabilities of constants move closer to 0, but so do those of variables, signifying roughly the same or poorer clustering. Interestingly, Android and HPC datasets also show this behaviour, but their probability distribution is better clustered with test-time training.

Observation 4. Test-time training exacerbates token-level classification ability of the transformer encoder model for 14 out of 16 benchmark datasets.

Moreover, for all datasets, the probabilities move closer to zero in varying degrees, which signals that, with test-time training, the likelihood of a token being predicted to be a constant increases, be it accurately (the boxplots of constants moving closer to 0) or inaccurately (the boxplots of variables movie further away from 1).

This observation can be explained by the mutated log data on which the model is trained

during test time each contains only one mutated variable token. All the other tokens of each mutated log is assumed by Log3T to be constant. The skew in the size of two categories may cause the token-level probabilities predicted by the model to be skewed as well.

Observation 5. Test-time training increases the likelihood of the model predicting a token to be constant, potentially due to the imbalanced dataset of mutated log.

Table 4.3 shows the the difference between the average predicted probabilities of variables and constants. Other than Android and HPC datasets, for the other 14 datasets, applying test-time training technique decreases the difference, signifying a deterioration in the model's classification ability.

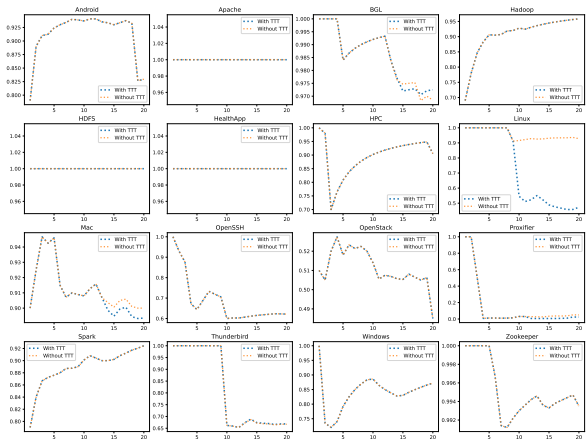
Observation 6. The difference between average probabilities of constants and variables predicted by the transformer encoder model drops for 14 out of 16 benchmark datasets.

4.3.6 RQ3.4. Effect of test-time training across encoder architectures

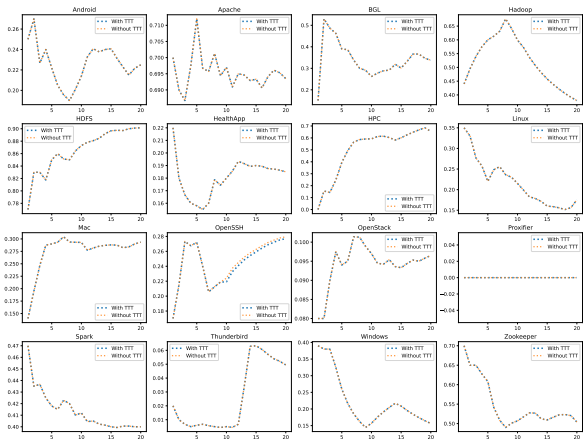
In this RQ, we examine if the results and observations collected from RQ3.1, RQ3.2, and RQ3.3 are the same across different encoder architectures other than the BERT-based transformer encoder used by Log3T.

Approach To investigate the applicability of the previous observations across different encoders, we intend to replicate the results using Lo3T with other models. We choose the T5 encoder model from Flan-t5 architecture [37]. Other than replacing the BERT-based transformer encoder model used in Log3T, we modify no other component of Log3T to verify the existence of no factor other than the encoder model.

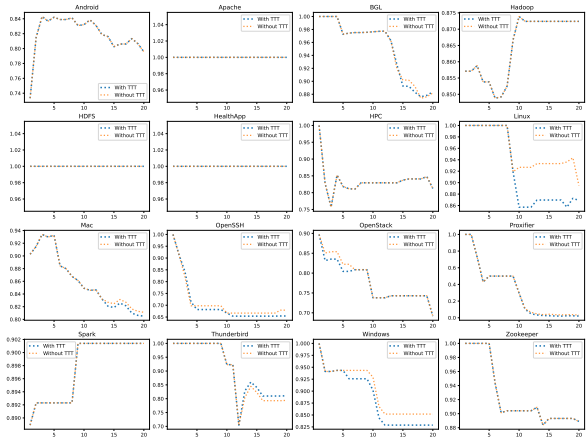
We run the same experiments as in RQ3.1, but with T5 encoder model instead of the BERT-based encoder model. Same as in RQ3.1, we split each of the benchmark datasets into 20 batches of equal size (100 logs per batch), where the first batch is treated as the historical log data. The encoder model is first trained on this historical log data with traditional offline



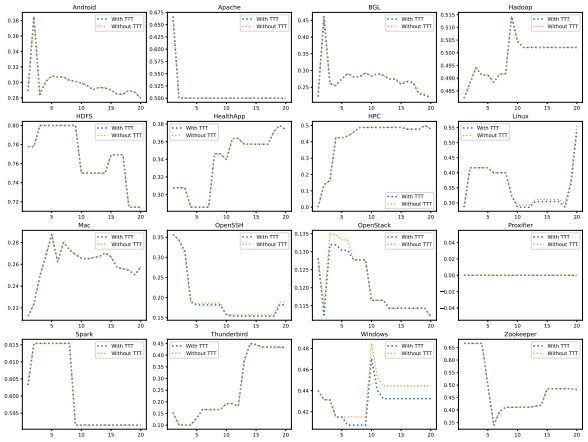
(a) Grouping accuracy (GA)



(b) Parsing accuracy (PA)



(c) F1-score of grouping accuracy (FGA)



(d) F1-score of template accuracy (FTA)

Figure 4.8: Accuracy with and without test-time training of Log3T using Flan-T5 as transformer encoder

training methods. During inference, test-time training is applied on the model and the final log templates are collected.

Results Figure 4.8 presents the log parsing accuracies when using Log3T with Flan-T5 as an encoder model. With the exception of BGL in GA and Thunderbird in FGA, all the other datasets show either no improvement or worse performance across all metrics. This demonstrates that the results and observations from RQ3.1-RQ3.3 are not exclusive to the underlying transformer encoder model.

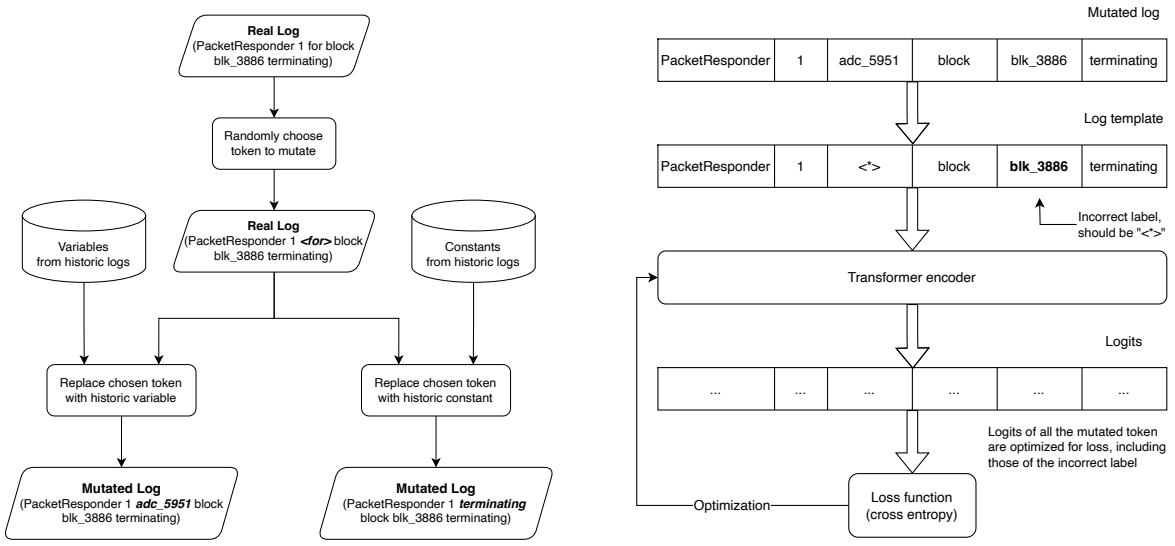
Table 4.4: Takeaways from the empirical study

ID	Observation from Log3T	Design Rationale of GeLT
	Category 1: Overview	
1	Test-time training strategy employed by Log3T overwhelmingly decreases the accuracy of log parsing instead of improving it	We limit our scope to optimizing the test-time training technique, specifically the auxiliary self-supervised task and the mutated log generation strategy. No design choices is proposed for the transformer encoder model.
4	Test-time training exacerbates token-level classification ability of the transformer encoder model for 14 out of 16 benchmark datasets	
7	The results and observations from the previous RQs regarding the effectiveness of test-time training are applicable across different encoder models	
	Category 2: Auxiliary self-supervised task	
2	The assumption of mutated log generation strategy results in 46.7% of tokens to be mislabelled on average	We only optimize the loss on the replaced token whose label is known, requiring no assumptions about unreplaced tokens.
3	Higher percentage of numeric tokens, which are mostly variables, lead to higher percentage of incorrect labels	
	Category 3: Log mutation strategy	
5	Test-time training increases the likelihood of the model predicting a token to be constant, potentially due to the imbalanced dataset of mutated log	We replace a randomly chosen token with historical variables *and* historical constants in equal quantities to eliminate imbalance. Each log is mutated once with constants and once with variables.
6	The difference between average probabilities of constants and variables predicted by the transformer encoder model drops for 14 out of 16 benchmark datasets	

Observation 7. The results and observations from the previous RQs regarding the effectiveness of test-time training are applicable across other encoder architectures, e.g., Flan-t5.

4.4 Methodology of GeLT

In this section, we present GeLT, our novel tool that is designed to address the shortcomings in the Log3T adoption TTT. In Table 4.4, we present the observations from our experiments on Log3T and the corresponding design choices we propose for improving the test-time training for log parsing. Specifically, we introduce two techniques in GeLT: a mutated log generation



(a) Mutated log generation strategy (b) Test-time training strategy with the mutated log

Figure 4.9: Optimized auxiliary task for test-time training

strategy and an auxiliary task as presented in Figure 4.9. We discuss the techniques below.

4.4.1 Mutated Log Generation Strategy

Log3T generates mutated log data by replacing a randomly chosen token in the test log sample with a historical variable. From RQ3, we have found that this strategy creates an imbalanced dataset of mutated logs which, after applying test-time training, results in the model's predictions to be skewed.

To address this, we propose a modified log generation strategy as shown in Figure 4.9a. We replace the randomly chosen token with both historical variables *and* historical constants. The number of mutated logs generated by replacing tokens with constants and with variables are equal in order to eliminate imbalance in the mutated log data.

4.4.2 Auxiliary Self-supervised Task

For test-time training, Log3T generates mutated log data from the real test sample and trains the the underlying transformer encoder model on the mutated logs. The auxiliary

self-supervised task for in Log3T is to predict the label for each token. However, from RQ2, we have found that the labels of the mutated logs generated by Log3T are inaccurate on average 46.7% of the time (O2).

Presented in Figure 4.9b, we modify the auxiliary task. Here, the model trains only on the token that is replaced by a historical variable token. Specifically, while training, the model only optimizes the loss of that replaced token, while the other tokens are ignored by masking.

4.5 Results

In this section, we evaluate the effectiveness of our proposed log mutation strategy and auxiliary self-supervised task. We present the following research questions.

RQ5. Does the optimized auxiliary task strategy improve token-level classification?

RQ6. Does the optimized auxiliary task strategy improve log parsing accuracy?

4.5.1 RQ5. Effectiveness of the optimized test-time training in token classification

In this RQ, we examine the effectiveness of GeLT to improve the token-level classification abilities of the underlying transformer encoder model of Log3T.

Approach Similar to RQ3, we collect the probabilities of the tokens predicted by the transformer encoder model during inference time. We do this once with and once without test-time training. The probabilities of constants should be close to zero and those of variables should be close to one. The larger the difference between the probabilities of the two categories, the better the token-level classification ability of the model.

We also calculate the difference the of the average probabilities of constant and variable tokens. The higher the difference, the better the classification ability.

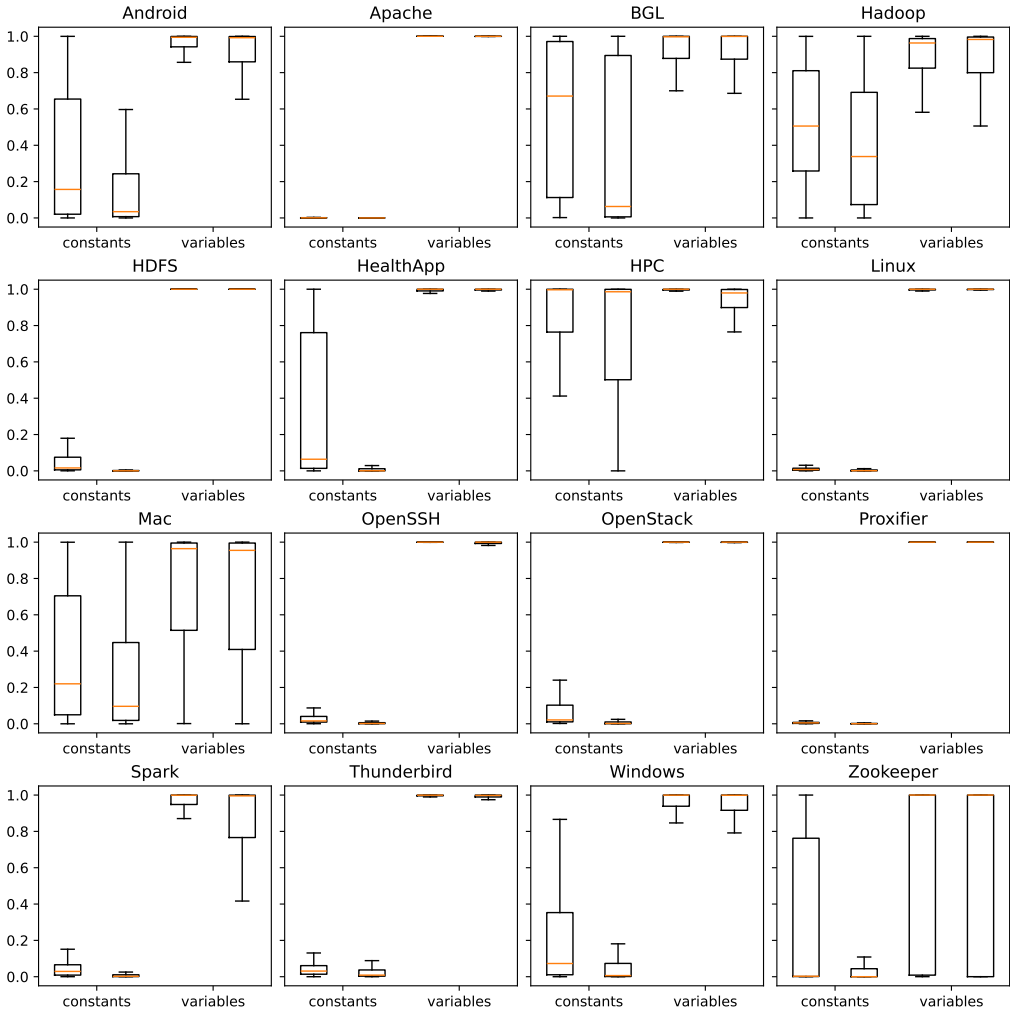


Figure 4.10: Distribution of token prediction probabilities from the encoder model with and without optimized test-time training of GeLT. For both constant and variable tokens, two boxplots are shown side by side comparing predictions without optimized test-time training (left) and with optimized test-time training (right).

Results Figure 4.10 shows the boxplots of the probabilities for the constant and variable tokens. For each of the categories (constants and variables), two boxplots are shown side-by-side. The left boxplot is without applying test-time training and the right boxplot is with test-time training. For 11 datasets, the probabilities of constants move closer to 0 while those of variables either a) remain same (BGL, HDFS, HealthApp, Linux, OpenStack, Proxifier, Zookeeper) or b) they move away from 1 but visibly less than the shift of the probabilities of constants (Android, Apache, Hadoop, HPC, Mac, OpenSSH, Spark, Thunderbird, Windows).

Table 4.5: Differences of average probabilities of constants and variables by optimized test-time training

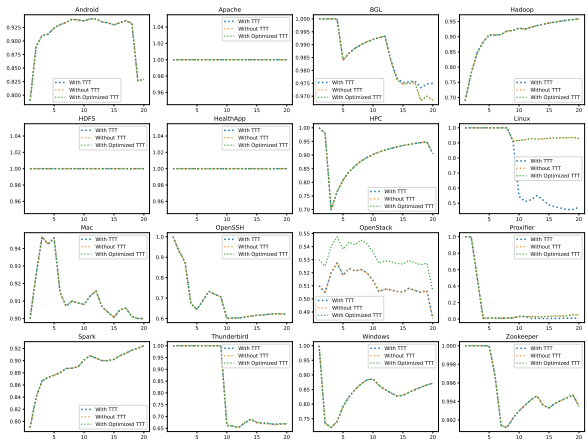
Dataset	With Log3T	Without TTT	With GeLT
Android	**0.62**	0.47	0.59
Apache	0.75	0.96	**0.95**
BGL	0.21	0.27	**0.46**
HDFS	0.50	0.84	**0.94**
HPC	**0.19**	0.12	0.18
Hadoop	0.28	0.31	**0.39**
HealthApp	0.51	0.59	**0.80**
Linux	0.44	0.76	**0.91**
Mac	0.23	0.37	**0.44**
OpenSSH	0.51	0.77	**0.81**
OpenStack	0.71	0.82	**0.91**
Proxifier	0.65	0.72	**0.89**
Spark	0.39	0.67	**0.70**
Thunderbird	0.53	0.80	**0.83**
Windows	0.38	0.72	**0.77**
Zookeeper	0.37	0.39	**0.46**
Average	0.45	0.60	**0.68**

Observation 8. For all benchmark datasets, GeLT increases the inter-cluster distance between the probabilities of constants and variables, demonstrating a consistent and robust improvement in classification ability of the transformer encoder model.

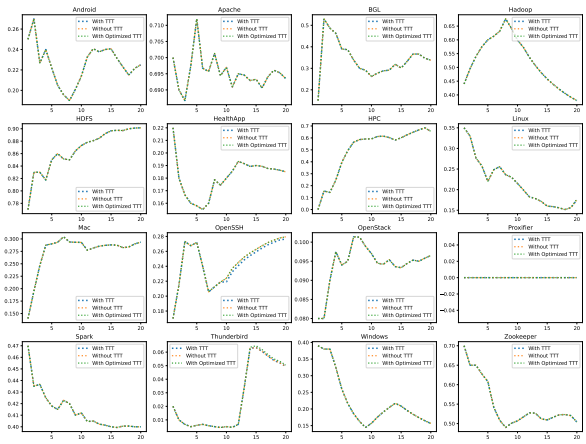
In Table 4.5, we present the difference between the average predicted probabilities of variables and constants. For all datasets, GeLT increases the difference, signifying a consistent and robust improvement in the model's classification ability.

Observation 9. GeLT improves the token-level classification ability of the underlying transformer encoder model. It increases the difference between the average probability of constant tokens and that of variable tokens for all benchmark datasets.

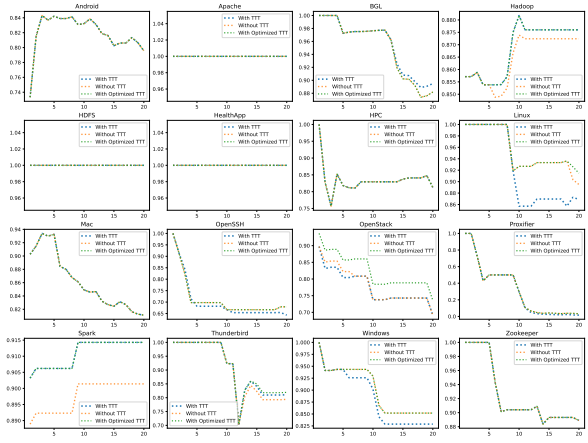
Moreover, other than Android and HPC datasets, for the other 14 datasets among the 16 benchmark datasets, the difference in average probabilities of constants and variables by GeLT is greater than those by Log3T, demonstrating that GeLT is more capable of improving the generalization ability of the transformer encoder model in terms of token-level classification.



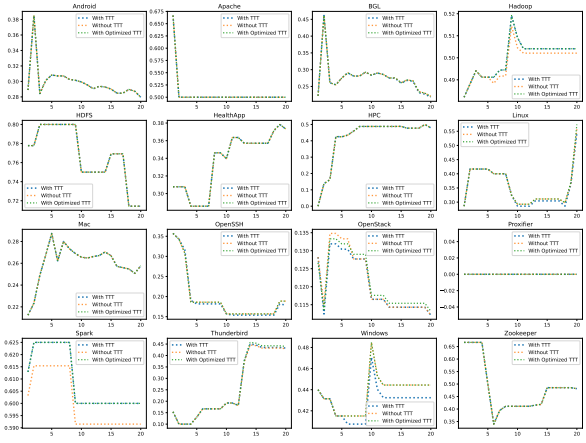
(a) Grouping accuracy (GA)



(b) Parsing accuracy (PA)



(c) F1-score of grouping accuracy (FGA)



(d) F1-score of template accuracy (FTA)

Figure 4.11: Accuracy of modified Log3T with and without test-time training

Observation 10. The improvement in classification ability of the transformer encoder model by GeLT is greater than that by Log3T for 14 out of 16 benchmark datasets.

4.5.2 RQ6. Effectiveness of the optimized test-time training in log parsing

In this RQ, we examine the effectiveness of the optimized log mutation strategy for test-time training by evaluating the log parsing accuracy.

Approach Similar to RQ1, we compare the log parsing accuracy of GeLT across all four accuracy metrics. The datasets are similarly split into 20 batches of 100 logs in sequential order, the first batch acting as the historical log data and the subsequent batches acting as new incoming logs. The baselines are: a) the model trained with only the first batch of 100 logs, and b) the model trained by Log3T's test-time training technique.

Results Figure 4.11 presents the log parsing accuracy across four metrics with and without applying test-time training. Among 16 datasets, the optimized test-time training improves GA and PA in 1 dataset (Thunderbird), FGA and FTA in 3 datasets (Linux, OpenStack, Thunderbird), while attaining equal accuracy as traditional offline training in the other datasets. Moreover, aside from BGL dataset, GeLT achieves better or equal accuracy in all the other datasets compared to Log3T.

Observation 11. GeLT improves log parsing accuracy over new log types in 3 out of 16 benchmark datasets in terms of template-level accuracies. Other than BGL dataset, GeLT outperforms Log3T in all other datasets.

4.6 Discussion

4.6.1 Threats to Validity

Our evaluation of Log3T is based on the implementation of the authors (**Internal validity.**). While we follow their hyperparameters in all respects, there may be differences in the hyperparameter values in their public replication package and the values originally used. The generalizable log parsing approach explored and proposed in this study is applicable to Transformer encoder models (**External validity.**). LLM-based log parsers adopt Seq2Seq decoder models for generating the log template from the input log message [16, 13]. The optimized test-time training approach of GeLT is not applicable to decoder models. To adopt test-time training for decoder models, a different auxiliary self-supervised task needs to be designed. The experiments in this study are conducted on Loghub-2k benchmark datasets, which may

not capture the diversity of the real-world large-scale log data (**Construct validity.**). As a result, the generalization capabilities reported may not reflect the results when applied to real-world scenarios.

4.6.2 Future Work

Although showing promising results, the test-time training approach, specifically the auxiliary self-supervised task, for log parsing in GeLT is applicable only to transformer encoder models. For decoder models, test-time training would call for a different strategy. We aim to continue our work of generalizable log parsing toward LLM-based decoder models, e.g., GPT [38], LLaMA [39].

4.7 Conclusion

In this chapter, we conducted an empirical analysis of the effectiveness of test-time training as designed by Log3T to increase the generalization ability of semantic-based log parsing. Our experiments showed that the log mutation strategy of Log3T and the auxiliary self-supervised task degrades the accuracy of the log parser instead of improving it. To address these issues, we propose an optimized log mutation strategy and a masking-based auxiliary task which improves GA and PA in 1 dataset and FGA and FTA 3 datasets among the 16 benchmark datasets, respectively.

Chapter 5

Conclusion

5.1 Thesis Findings and Contributions

This thesis investigates the effectiveness and limitations of existing log parsing approaches through an empirical study and proposes improvements to both syntax-based and semantic-based log parsing techniques. The key findings of this thesis are summarized as follows.

1. **Limitations of existing log parsing approaches.** The empirical study reveals several limitations in existing log parsing techniques. In particular, many syntax-based parsers exhibit a significant discrepancy between grouping accuracy (GA) and parsing accuracy (PA). Specifically, on average, syntax-based log parsers show a 51% drop in PA compared to GA. This indicates that correctly grouped log messages do not always lead to accurate template extraction. The study shows that this discrepancy is often caused by incorrect identification of variables, where tokens that remain constant within a group of log messages are incorrectly treated as template tokens.

2. **Characteristics of variables in benchmark log datasets.** The empirical analysis also provides insights into the characteristics of variables present in the benchmark datasets. A substantial proportion of variables (50% in Loghub-2k datasets [14]) follow common syntactic patterns such as IP addresses, file paths, or numerical identifiers.

These observations suggest that lightweight syntactic methods, such as regular expression matching, can be effectively used to identify many variable tokens and improve template inference without incurring the computational cost of semantic-based methods.

3. **Effectiveness of template refinement for improving parsing accuracy.** Based on these insights, this thesis proposes SynLog+, a template refinement pipeline designed to improve the accuracy of template inference in existing log parsers. The experimental results demonstrate that refining templates after log grouping can substantially improve parsing performance. Overall, SynLog+ improves the accuracy of the benchmark log parsers by 157% in parsing accuracy (PA) and 553% in F1 template accuracy (FTA), highlighting the effectiveness of lightweight syntactic refinements for improving template extraction.

4. **Challenges in generalizable semantic-based log parsing.** This thesis also examines the design of existing generalizable semantic-based log parsing through a case study of Log3T. The analysis identifies several design limitations. In particular, the results show that Log3T can sometimes degrade performance on certain datasets while offering inconsistent improvement in some datasets. Spcifically, out of 16 benchmark datasets, Log3T improves FGA in 2 datasets and FTA in 0 datasets while dropping in FGA and FTA in 5 and 4 datasets, respectively.

5. **Improved generalization through optimized test-time training.** To address these design flaws, this thesis proposes GeLT, a semantic-based log parsing approach that employs test-time training to improve generalization. Experimental evaluation demonstrates that GeLT improves generalization in several benchmark datasets compared to the existing approach. Specifically, GeLT improves the F1 GeLT improves F1 grouping accuracy (FGA) in 3 and F1 template accuracy (FTA) in 2 datasets out of 16 benchmark datasets. These results suggest that adapting model parameters during

inference can be an effective strategy for improving cross-dataset log parsing performance.

Overall, the findings of this thesis highlight that both syntactic refinement and improved semantic generalization strategies play important roles in advancing the state of the art in log parsing.

5.2 Thesis Limitations

1. **Reliance on Predefined Variable Patterns.** SynLog+ improves template inference partly through the use of regular expressions that capture common variable formats, e.g., IP addresses and file paths. While these patterns cover many common variable types, they may not capture all possible variable formats present in diverse log sources. Consequently, some variables with uncommon or domain-specific formats may remain incorrectly identified.

2. **Evaluation Scope of Semantic Models.** The semantic-based component explored in this thesis (GeLT) focuses on transformer encoder models with a test-time training strategy to improve generalization. However, recent advances in log parsing increasingly leverage large decoder-only LLMs. Since the generalization strategy proposed in GeLT is tailored to encoder architectures, its applicability to decoder-only models remains unexplored.

3. **Offline Evaluation Setting.** The experiments conducted in this thesis focus on offline log parsing, where the entire dataset is available prior to analysis. In real-world monitoring systems, log messages are often generated continuously and require online or streaming parsing. The proposed approaches would need additional adaptations to operate effectively in such environments.

5.3 Future Work

While this thesis have made progress in bridging the gap between high grouping accuracy and low parsing accuracy of syntax-based methods (with SynLog+) and improving the generalization ability of semantic-based methods (with GeLT), several promising areas remain for future exploration as follows.

1. **Extending SynLog+ toward online log parsing.** SynLog+ is currently designed as a post-group template refinement pipeline that operates on grouped log messages produced by existing parsers. An interesting direction for future work is to extend SynLog+ to support online log parsing scenarios, where log messages arrive continuously and templates must be inferred incrementally. In such environments, template refinement would need to be performed dynamically as new log messages are processed. Developing mechanisms for incremental template updates and efficient pattern detection could enable SynLog+ to operate effectively in real-time monitoring systems while preserving its lightweight design.

2. **Generalization of decoder-only semantic models.** This thesis proposes a generalizable semantic-based log parser, GeLT, which leverages a transformer-encoder model and employs a test-time training strategy to improve generalization over unseen log data. Recently, LLM-based log parsing approaches, which uses decoder-only models, have demonstrated superior parsing accuracy but suffer in generalization like other PLM-based log parsers (Obs. 1.4). However, the generalization strategy developed in GeLT is designed specifically for encoder-based architectures and cannot be directly applied to decoder-only models. A promising direction for future work is therefore to design effective generalization strategies tailored to decoder-only models.

Bibliography

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, (New York, NY, USA), p. 1285–1298, Association for Computing Machinery, 2017.

[2] H. Zheng, G. Chu, H. Sun, J. Wang, S. Tao, and H. Yang, "Logdapt: Log data anomaly detection with domain-adaptive pretraining (industry track)," in Proceedings of the 24th International Middleware Conference: Industrial Track, Middleware '23, (New York, NY, USA), p. 15–21, Association for Computing Machinery, 2023.

[3] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "Logfit: Log anomaly detection using fine-tuned language models," IEEE Transactions on Network and Service Management, vol. 21, no. 2, pp. 1715–1723, 2024.

[4] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," J. Syst. Softw., vol. 86, p. 2–11, Jan. 2013.

[5] D. Das, M. Schiewe, E. Brighton, M. Fuller, T. Cerny, M. Bures, K. Frajtak, D. Shin, and P. Tisnovsky, "Failure prediction by utilizing log analysis: A systematic mapping study," in Proceedings of the International Conference on Research in Adaptive and Convergent Systems, RACS '20, (New York, NY, USA), p. 188–195, Association for Computing Machinery, 2020.

[6] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "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, ASPLOS XV, (New York, NY, USA), p. 143–154, Association for Computing Machinery, 2010.

[7] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based abnormal task detection and root cause analysis for spark," in 2017 IEEE International Conference on Web Services (ICWS), pp. 389–396, 2017.

[8] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp. 114–117, 2010.

[9] M. Mizutani, "Incremental mining of system log format," in 2013 IEEE International Conference on Services Computing, pp. 595–602, 2013.

[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), pp. 33–40, 2017.

[11] V.-H. Le and H. Zhang, "Log parsing with prompt-based few-shot learning," in Proceedings of the 45th International Conference on Software Engineering, ICSE '23, p. 2438–2449, IEEE Press, 2023.

[12] Y. Liu, S. Tao, W. Meng, J. Wang, W. Ma, Y. Zhao, Y. Chen, H. Yang, Y. Jiang, and X. Chen, "Interpretable online log analysis using large language models with prompt strategies," 2024.

[13] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Lilac: Log parsing using llms with adaptive parsing cache," Proc. ACM Softw. Eng., vol. 1, July 2024.

[14] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-driven log analytics," in 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), pp. 355–366, IEEE, 2023.

[15] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu, and M. R. Lyu, "A large-scale evaluation for log parsing techniques: How far are we?," 2024.

[16] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," in Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, vol. 48 of ICSE '24, p. 1–13, ACM, Apr. 2024.

[17] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," J. Softw. Maint. Evol., vol. 20, p. 249–267, July 2008.

[18] Y. Wu, S. Yu, and Y. Li, "Log parsing with self-generated in-context learning and self-correction," 2024.

[19] Y. Xiao, V.-H. Le, and H. Zhang, "Stronger, cheaper and demonstration-free log parsing with llms," 2024.

[20] R. Vaarandi and M. Pihelgas, "Logcluster - a data clustering and pattern mining algorithm for event logs," in 2015 11th International Conference on Network and Service Management (CNSM), pp. 1–7, 2015.

[21] H. Dai, H. Li, W. Shang, T.-H. Chen, and C.-S. Chen, "Logram: Efficient log parsing using n-gram dictionaries," 2020.

[22] K. Shima, "Length matters: Clustering system log messages using length of words," 2016.

[23] 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, CIKM '16, (New York, NY, USA), p. 1573–1582, Association for Computing Machinery, 2016.

[24] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 859–864, 2016.

[25] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang, S. Rajmohan, and D. Zhang, "Uniparser: A unified log parser for heterogeneous log data," in Proceedings of the ACM Web Conference 2022, WWW '22, (New York, NY, USA), p. 1893–1901, Association for Computing Machinery, 2022.

[26] S. Yu, Y. Wu, Z. Li, P. He, N. Chen, and C. Liu, "Log parsing with generalization ability under new log types," in Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, (New York, NY, USA), p. 425–437, Association for Computing Machinery, 2023.

[27] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "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, (Berlin, Heidelberg), p. 122–138, Springer-Verlag, 2020.

[28] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), pp. 1095–1106, 2022.

[29] V.-H. Le, Y. Xiao, and H. Zhang, "Unleashing the True Potential of Semantic-Based Log Parsing with Pre-Trained Language Models," in 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), pp. 975–987, Apr. 2025.

[30] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '19, p. 121–130, IEEE Press, 2019.

[31] Z. Li, C. Luo, T.-H. P. Chen, W. Shang, S. He, Q. Lin, and D. Zhang, "Did we miss something important? studying and exploring variable-aware log abstraction," in Proceedings of the 45th International Conference on Software Engineering, ICSE '23, p. 830–842, IEEE Press, 2023.

[32] X. Wang, X. Zhang, L. Li, S. He, H. Zhang, Y. Liu, L. Zheng, Y. Kang, Q. Lin, Y. Dang, S. Rajmohan, and D. Zhang, "Spine: a scalable log parser with feedback guidance," in Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, (New York, NY, USA), p. 1198–1208, Association for Computing Machinery, 2022.

[33] H. Wang and X. Bi, "Domain generalization and adaptation based on second-order style information," Pattern Recognition, vol. 127, p. 108595, July 2022.

[34] S. Shankar, V. Piratla, S. Chakrabarti, S. Chaudhuri, P. Jyothi, and S. Sarawagi, "Generalizing across domains via cross-gradient training," 2018.

[35] M. Chen, K. Q. Weinberger, and J. C. Blitzer, "Co-training for domain adaptation," in Proceedings of the 25th International Conference on Neural Information Processing Systems, NIPS'11, (Red Hook, NY, USA), p. 2456–2464, Curran Associates Inc., 2011.

[36] Y. Sun, X. Wang, Z. Liu, J. Miller, A. A. Efros, and M. Hardt, "Test-time training with self-supervision for generalization under distribution shifts," 2020.

[37] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," 2022.

[38] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[39] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.