PhD-FSTM-2023-111
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Presented on 13/11/2023 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Zanis Ali Khan
Born on 09 July 1992 in Attock (Pakistan)

ON LOG PARSING AND LOG-BASED ANOMALY DETECTION: AN EMPIRICAL EVALUATION

Acknowledgments

I want to express my gratitude and extend sincere thanks to my supervisor, Dr. Domenico Bianculli, for his substantial contributions to the success of my research. His dedication, insightful guidance, and scientific approach have kept me actively involved in my research endeavors.

I extend my heartfelt appreciation to Prof. Lionel C. Briand for his invaluable support and guidance in completing this thesis. I feel privileged to have benefited from his expertise and academic excellence.

I am profoundly thankful to my co-supervisor, Dr. Donghwan Shin, for significantly enriching this thesis. His dynamic guidance and unwavering generosity played a pivotal role in the successful completion of my tasks.

I express heartfelt gratitude to my family and friends for their unwavering support during my doctoral journey.

Abstract

A software log is a sequence of log messages generated by log printing statements in the source code. Logs are essential for various software engineering tasks, such as model inference and anomaly detection, since they are often the only data available that records the run-time behavior of a software system. However, they cannot be directly processed by log based analysis techniques that require structured input logs instead of free-formed log messages. *Log parsing* aims to address the issue by decomposing log messages into fixed parts called message templates, characterizing the event types, and variable parts containing the parameter values of the events, which are determined at run time.

Although, many log parsing techniques have been presented, they have not been systematically compared and ranked using different criteria. Additionally, logs have been used widely in log-based anomaly detection and might affect anomaly detection accuracy; yet, the relationship between log parsing and anomaly detection has not been thoroughly investigated. With the emergence of non-log-parsing-based anomaly detection techniques that would rule out the impact of log parsing, a comprehensive evaluation to assess which approach is more suitable for anomaly detection is required.

In this thesis we have made the following contributions:

1. We assessed and compared different log parsing techniques and provided guidelines for evaluating the accuracy of log parsing techniques considering different use cases.
2. We proposed a theoretical framework for understanding the relationship between log parsing and anomaly detection, formally defining the concepts of *distinguishability* and *minimality* of *ideal* log parsing results.
3. We performed a comprehensive empirical study investigating the impact of log parsing on anomaly detection accuracy.
4. We performed a comprehensive empirical study comparing the accuracy and efficiency of log-parsed-based and non-log-parsing-based anomaly detection techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Software logs are essential for various software engineering tasks, such as model inference [WTD16, MPS17] and anomaly detection [NMA$^+$16, DLZS17], since logs are often the only data available that records the run-time behavior of a software system. In general, a log is a sequence of log messages generated by log printing statements in the source code. For example, the execution of the log printing statement `log("retry " + counter)`, when the program variable `counter` evaluates to 1, will generate the log message "`retry 1`". While such log messages contain valuable run-time information, they cannot be directly processed by log-based analysis techniques that require structured input logs instead of free-formed log messages.

Log parsing (a.k.a. log message template identification) aims to address the issue by decomposing log messages into fixed parts called message templates (templates, in short), characterizing the event types, and variable parts containing the parameter values of the events, which are determined at run time. For the above example message, the event template would be "`retry <*>`", where symbol "`<*>`" indicates the position of the parameter value ("`1`") in the variable part. Log template identification is straightforward when one has access to the source code because one can derive message templates from the log printing statements. However, often the source code is not available, for example, when the system is composed of 3rd-party components, and, as a result, many automated log template identification techniques have been proposed in the literature (e.g., LogPPT [LZ23], AEL [JHFH08], Drain [HZZL17], IPLoM [MZHM09], LenMa [Shi16], LFA [NV10], LKE [FLWL09], LogCluster [VP15], LogMine [HDX$^+$16], LogSig [TLP11], MoLFI [MPB$^+$18], SHISO [Miz13], SLCT [Vaa03], Spell [DL16]) to identify templates using only log messages [ZHL$^+$19]. Although many log parsing techniques have been presented, they have not been systematically compared and ranked using different criteria, making it challenge to decide which one is the best with respect to certain criteria (e.g., accuracy, execution time). Therefore, an essential research question is: **How can we properly assess and compare different log parsing techniques?**.

Logs have been widely used in log-based anomaly detection techniques to automatically detect if the logs contain any anomalous patterns that do not conform to the expected behavior of the system [HHC$^+$21]. Many techniques have been proposed for log-based anomaly detection based on Deep Learning (DL) models, such as Long Short-Term Memory (LSTM) [DLZS17, MLZ$^+$19, ZXL$^+$19] and Convolutional Neural Networks (CNNs) [LWLW18]. Most of these techniques have in common an essential pre-processing step. This step is required because anomaly detection techniques require structured logs to automatically process them, whereas logs are generally often free-formed or semi-structured. Therefore, log parsing might significantly affect anomaly detection, and therefore, it is essential to thoroughly investigate the relationship between log parsing and anomaly detection, to determine whether using a certain log parsing technique that yields a high accuracy score (according to a certain metric) leads to a higher accuracy score for anomaly detection. Therefore, an essential research question is **What are the ideal log parsing results for better anomaly detection, and is there a correlation between log parsing accuracy and anomaly detection accuracy?**

As mentioned above, the impact of log parsing techniques on log-parsing-based anomaly detection techniques might not be neglected. However, the existence of an anomaly detection technique that does not rely on log parsing might be another solution that would eventually rule out the impact of log parsing on anomaly detection accuracy. Recently, an anomaly detection technique namely NeuralLog [LZ21] has been proposed; it does not use log-parsing as a pre-processing step. Although non-log-parsing-based and log-parsing-based techniques utilize the same logs, they differ in the way they process the logs. NeuralLog processes the raw log messages to represent semantic information in the form of semantic vectors. These semantic vectors are used by a Transformer-based classification model to capture the contextual information from the log sequences, and to detect anomalies. Le and Zhang [LZ21] have shown that the performance of anomaly detection models is highly influenced by log parsers; the reason is that log-parsing errors can lead to extra log events and wrong log templates that are utilized by many anomaly detection models. Due to this very reason, one might speculate that, without the pre-processing step (log parsing) involved, the non-log-parsing-based anomaly detection technique might be more accurate than log-parsing-based anomaly detection techniques. This leads to our last question: **Is a non-log-parsing-based approach more suitable for anomaly detection than a log-parsing-based approach?**.

1.2 Research Contributions

The overall goal of this dissertation is to investigate the answer to the aforementioned questions through theoretical and empirical studies.

To answer the first question *How can we properly assess and compare different log parsing techniques?*, we provide guidelines for assessing the accuracy of log message template identification techniques and assessed the application of such guidelines through a comprehensive evaluation of 14 existing template identification techniques. More specifically, to address the issue related to the choice of accuracy metrics, we first describe new metrics, called *Template Accuracy* (TA) metrics, that, unlike GA and PA, are not sensitive to the number of log messages. The accuracy metric, sensitive to the number of log messages, might lead to misleading results when there are many repetitive but non-essential messages (e.g., heartbeat messages). We then discuss which metrics (GA, PA, or TA) are more appropriate depending on the nature of the log analysis task in which log message templates are used. Additionally, we propose a set of heuristic rules for correcting oracle templates in order to minimize the negative bias in experimental results introduced by manually generated oracle templates. Furthermore, to provide additional information about incorrectly identified templates for a detailed analysis, we propose a technique for analyzing incorrect templates.

To answer the second question *What are the ideal log parsing results for better anomaly detection, and is there a correlation between log parsing accuracy and anomaly detection accuracy?*, we propose a theoretical framework for defining and discussing what are ideal log parsing results for anomaly detection. As mentioned, log-based anomaly detection techniques require log-parsing as a pre-processing step and thus acts as an information abstraction process, as it converts the log messages (specific form) to a more generic form (i.e., event templates). Since automated anomaly detection relies on structured logs, its best operating conditions are when the minimum amount of information that is necessary to distinguish normal from abnormal behaviors is present in such logs. To this end, we formally define the concepts of *distinguishability* and *minimality*, which further lead to the definition of *ideal* log parsing results. Additionally, we performed a comprehensive empirical study using 13 log parsing techniques and five deep learning-based anomaly detection techniques on three publicly available log datasets. In the study, we provided (1) a systematic and comprehensive evaluation of the impact of log parsing on anomaly detection, (2) an investigation of the impact of the distinguishability of log parsing results on anomaly detection, and (3) a discussion of the practical implications for the application of log parsing in the context of anomaly detection.

To answer the third question *Is a non-log-parsing-based approach more suitable for anomaly detection than a log-parsing-based approach?*, we compared log-parsing-based anomaly detection techniques with non-log-parsing-based technique using 8 datasets and evaluated them in terms of accuracy, and efficiency, aiming to guide practitioners in selecting the most suitable anomaly detection technique for their specific use cases.

1.3 Dissemination

The publications resulted from our research work are listed below. These publications are arranged in chronological order based on their publication date.

Published papers

- Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)*, page 1095–1106, New York, NY, USA, 2022. ACM, ACM. `doi:10.1145/3510003.3510101`. This paper is the basis for Chapter 3.

- Donghwan Shin, Zanis Ali Khan, Domenico Bianculli, and Lionel Briand. A theoretical framework for understanding the relationship between log parsing and anomaly detection. In *International Conference on Runtime Verification (RV'21)*, pages 277–287, Cham, 2021. Springer, Springer. `doi:10.1007/978-3-030-88494-9_16`. This paper is the basis for Chapter 4.

Unpublished reports

- Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. Impact of log parsing on log-based anomaly detection, 2023. `arXiv:`

`2305.15897`. This paper is the basis for Chapter 5. It is under review for publication in the EMSE journal.

- KHAN, Zanis Ali, SHIN, Donghwan, BIANCULLI, Domenico, and BRIAND, Lionel. Deep Learning-Based Anomaly Detection: A Comparison of Non-Log-Parsing-Based and Log-Parsing-Based Approaches. This paper is the basis for Chapter 6.

1.4 Organization of the Thesis

Chapter 2 provides fundamental concepts for the thesis. We define logs, messages, and templates, explain log parsing, delve into log-based anomaly detection, and distinguish between log-parsing-based and non-log-parsing-based anomaly detection techniques.

Chapter 3 presents guidelines for assessing the accuracy of log message template identification techniques.

Chapter 4 presents a theoretical framework for understanding the relationship between log parsing and anomaly detection.

Chapter 5 presents an empirical study on the impact of log-parsing on deep learning-based anomaly detection.

Chapter 6 presents an empirical study on the comparison of non-log-parsing-based and log-parsing-based approaches for anomaly detection.

Chapter 7 summarizes the thesis contributions and discusses perspectives on future works.

Chapter 2

Background

In this section, we provide an overview of the main concepts that will be used throughout the thesis. We first introduce the definitions of logs, messages, and log templates (§ 2.1.1). We then explain the concept of log parsing (also known as log template identification) (§ 2.1.2). We discuss log-based anomaly detection and the corresponding accuracy metrics in § 2.2. Additionally, we discuss the differences between log-parsing-based anomaly detection and non-log-parsing-based anomaly detection techniques.

2.1 Logs and Log Parsing

2.1.1 Logs, Messages, and Templates

A *log* is a sequence of log entries[1]. A *log entry* contains various information about the event being logged, including a timestamp, a logging level (e.g., `INFO`, `DEBUG`), and a log message. A *log message* can be further decomposed into fixed and variable parts since it is generated by executing a logging statement that can have both fixed (hard-coded) strings and program variables in the source code. For example, the execution of the logging statement "`logger.info("Deleting block " + blkID + " file " + fileName)`"

[1]Note that a log is different from a log file. In practice, *one* log file may contain *many* logs representing the execution flows of different components/sessions. For example, an HDFS (Hadoop Distributed File System) log file contains many logs, distinguished by file block IDs, each representing an independent execution for a specific block.

when the program variables `blkID` and `fileName` evaluate to `blk-1781` and `/hadoop/dfs`, respectively, will generate a log entry "`11:22:33 INFO Deleting block blk-1718 file /hadoop/dfs`" where the log message "`Deleting block blk-1718 file /hadoop/dfs`" can be decomposed into the fixed parts (i.e., "`Deleting block`" and "`file`") and the variable parts (i.e., "`blk-1718`" and "`/hadoop/dfs`"). A *(log message) template* masks the various elements of each variable part with a special character "`<*>`"; this representation is widely used in log-based analyses (e.g., log parsing [HZZL17, JHFH08], anomaly detection [ZXL+19, DLZS17], and log-based testing [Ely12, JJSL20]) when it is important to focus on the event types captured by a log message. For instance, the template corresponding to the example log message "`Deleting block blk-1178 file /hadoop/dfs`" is "`Deleting block <*> file <*>`".

2.1.2 Log Parsing (Log Template Identification)

Although software execution logs contain valuable information about the run-time behavior of the software system under analysis, they cannot be directly processed by log-based downstream analysis techniques that require structured input logs (containing templates) instead of free-formed log messages. Extracting log templates from log messages is straightforward when the source code with the corresponding logging statements is available. However, often the source code is unavailable, for example, due to the usage of 3rd-party, proprietary components. This leads to the problem of log parsing (log template identification): *How can we identify the log templates of log messages without accessing the source code?*

To address this problem, many automated log-parsing approaches, which take as input log messages and identify their log templates have been proposed in the literature. Such log parsing techniques are based on different approaches, such as:

1. *Frequent Pattern Mining:* A group of items that regularly appear in a data collection is known as a frequent pattern. The same may be said of event templates, which are a collection of consistent tokens that are regularly seen in logs. Therefore, frequent pattern mining is a simple method for automating log parsing. SLCT [Vaa03], LFA [NV10], and LogCluster [VP15] are examples of log parsing techniques using frequent pattern mining.

2. *Clustering:* Log parsing can be considered a clustering problem, as log templates form a natural pattern of a group of log messages. LogMine [HDX+16], LogSig [TLP11], SHISO [Miz13], and LenMa [Shi16] are few techniques using clustering.

3. *Heuristics:* Log messages have unique characteristics that can be used by log parsing techniques to extract log templates, for example, AEL [JHFH08] separates log messages into multiple groups by comparing the occurrences between constant tokens and variable tokens. Drain [HZZL17] is another technique using heuristics, as it applies a fixed-depth tree structure to represent log messages and extract common log templates efficiently.

2.2 Anomaly Detection

(Log-based) anomaly detection is a technique that aims to identify anomalous patterns, recorded in input logs, that do not conform to the expected behaviors of the system under analysis [HHC$^+$21]. It takes as input a sequence of log templates and determines whether the given sequence represents a normal behavior of the system or not.

With the recent advances in Deep Learning (DL), many anomaly detection approaches, which leverage DL models to learn various aspects of log template sequences of normal and abnormal behaviors and classify them, have been proposed in the literature; for example, DeepLog [DLZS17], LogAnomaly [MLZ$^+$19], and LogRobust [ZXL$^+$19] are based on Long Short-Term Memory based (LSTM), CNN [LWLW18] is based on Convolutional Neural Network, and PLELog [YCW$^+$21] is based on Gated recurrent units (GRUs).

To assess the accuracy of anomaly detection approaches, it is common practice to use standard metrics from the information retrieval domain, such as *Precision*, *Recall*, and *F1-Score*. These metrics are defined as follows: $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, and $F1\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall}$ where *TP* (True Positive) is the number of abnormal logs correctly identified by the model, *FP* (False Positive) is the number of normal logs incorrectly identified as anomalies by the model, and *FN* (False Negative) is the number of abnormal logs incorrectly identified as normal.

2.2.1 Log-parsing-based Anomaly Detection

For the sake of understanding the difference between non-log-parsing-based and log-parsing-based anomaly detection techniques, we can look at three different anomaly detection techniques proposed in the literature:

- DeepLog [DLZS17] detect log anomalies by employing LSTM. DeepLog captures log patterns by analyzing the sequential relations between log events, wherein each log message is represented by its respective log event index. Furthermore, it also utilizes a limited portion of normal event subsequences to train the LSTM model, and consequently, this enables the model to subsequently identify and classify normal event subsequences accurately.

- LogRobust [ZXL+19] incorporates an attention mechanism into a BiLSTM model. This mechanism assigns different weights to log events, known as attentional BiLSTM. Specifically, LogRobust adds a fully connected layer as the attention layer to the combined hidden state (h_t). It calculates an attention weight (a_t) to indicate the importance of the log event at time step t. This weight is then determined by applying a tangent function ($tanh$) to the product of (W_t^a) (the weight of the attention layer) and (h_t). To generate the classification result (whether it is an anomaly or not), LogRobust sums the hidden states at different time steps based on their respective attention weights. It then uses a softmax layer, using the weight W, to calculate the maximum value among the summed values ($\sum_{t=1}^{T} a_t \cdot h_t$). Here, T corresponds to the length of the log sequence.

- LogCluster [LZL+16] uses a clustering-based approach method for identifying online system problems. The methodology comprises of two training phases: knowledge base initialization and online learning. In the initialization phase, log sequences are vectorized using event count vectors, which are then revised with Inverse Document Frequency (IDF) and normalized accordingly. Agglomerative hierarchical clustering is used to separately cluster normal and abnormal event count vectors, creating distinct knowledge base sets. Representative vectors for each cluster are computed as centroids by assigning a score to each log sequence in a cluster by measuring the distance of each log sequence to other log sequences, and selecting a log sequence as centroid with minimal score. In the online learning phase, clusters are refined by incrementally adding event count vectors and updating representative vectors based on distances. LogCluster is subsequently used for anomaly detection by computing distances to representative vectors. If the smallest distance surpasses a predefined threshold, the log sequence is identified as an anomaly; otherwise, it is categorized as normal or abnormal based on proximity to the nearest cluster.

Note that all three aforementioned anomaly detection techniques require log-parsing as a preliminary step.

2.2.2 Non-log-parsing-based Anomaly Detection

Compared to the log-parsing-based anomaly detection as described with examples in Section 2.2.1, Le and Zhang [LZ21] proposed an anomaly detection technique called NeuralLog that does not use log-parsing as a pre-processing step. It consists of three main steps.

1. *Preprocessing:* A log message is first tokenized into a set of word tokens by using common delimiters in the logging system (i.e., white space, colon, comma, etc.) to split a log message. All non-character tokens (operators, punctuation marks, and number digits) are removed from the word set since they usually represent variables in the log message and are not informative.

2. *Neural Representation:* NeuralLog focuses on extracting meaningful information from log messages, including both the message header and content. Unlike existing methods that typically use the message content, while NeuralLog uses all textual information available, such as component, verbosity, and content, to capture the semantic meaning of log messages. Furthermore, it consists of: *(a) Subword Tokenization:* WordPiece tokenization is utilized to handle out-of-vocabulary (OOV) words. This incorporates all characters and symbols into a base vocabulary and iteratively selects the most likely pair based on training data likelihood. This technique reduces the number of OOV words and captures their meanings effectively. *(b) Log Message Representation:* NeuralLog uses the BERT base model, which consists of 12 transformer encoder layers and 768 hidden units per transformer. Each layer generates embeddings for the subwords in a log message. The word embeddings from the last encoder layer are averaged to compute the embedding of the log message. This approach enables BERT to handle out-of-vocabulary (OOV) words by learning their representation vectors based on subword meanings. BERT's positional embedding layer captures word representations in the context of the log message, while its self-attention mechanisms effectively measure word importance.

3. *Transformer-based Classification:* Transformer-based classification consists of two phases: (a) positional encoding, and (b) transformer encoder.
(a). *Positional encoding* [LYDH20, SQ19, CTZ$^+$21]: The order of log messages in a log sequence is crucial for anomaly detection. BERT encoder represents log messages with similar meanings closer together but lacks relative position information. To address this, a sinusoidal encoder generates embeddings using *sin* and *cos* functions for each position in the log sequence. These embeddings are added to the semantic vectors, allowing the transformer-based model to learn the relative position information and distinguish log messages at different positions.
(b). *Transformer encoder*: This model is based on the transformer architec-

ture [VSP+17], featuring self-attention and position-wise feed-forward layers. Positional embeddings are added to the input before entering the transformer. Multi-head attention layers calculate attention score matrices for each log message, employing different attention patterns. The model combines these scores through a feed-forward network. The output undergoes pooling, dropout, and a fully connected layer. Class probabilities for identifying normal/abnormal log sequences are obtained using a softmax classifier.

Chapter 3

Guidelines for Assessing the Accuracy of Log Message Template Identification Techniques

3.1 Overview

Given the number of available techniques for log message template identification, it is important to evaluate and rank them using different criteria (e.g., accuracy, execution time). However, most of the proposed techniques have been evaluated, in terms of accuracy, in isolation or with respect to a few alternatives, often using ad-hoc benchmarks. A notable exception is the study by Zhu et al. [ZHL$^+$19], which provides the first comprehensive evaluation of the 13 aforementioned techniques using a benchmark composed of 16 log datasets collected from real-world systems. Dai et al. [DLC$^+$20] recently used the same benchmark to compare their new technique (Logram) with the five techniques that achieved the highest accuracy scores as reported by Zhu et al. [ZHL$^+$19]. Nevertheless, in both studies, we observed three important issues regarding the accuracy evaluation of template identification techniques.

First, both studies used different accuracy metrics, namely Grouping Accuracy (GA)[1] and Parsing Accuracy (PA), leading to different rankings for several techniques. However, these evaluation results have not been compared with each other yet, making it hard to clearly understand how and why the rankings of template identification techniques vary when using different accuracy metrics. Furthermore, some accuracy metrics can return misleading results since they are sensitive to the number of repeated log messages. This can be a significant issue for systems whose logs contain repeated log messages (e.g., heartbeat messages) that are not important in terms of the system's business logic, a common situation.

Second, both studies determined the oracle templates (which represent the ground truth of template identification results) by manually inspecting log messages, which is an error-prone process. Though such a process is unavoidable when the source code is not accessible, the impact of using incorrect oracle templates on the assessment of the accuracy of template identification techniques is unclear for the various accuracy metrics.

Third, both studies did not provide any information regarding the templates incorrectly identified by template identification techniques. Due to such lack of information, we cannot understand the weaknesses and limitations of the various techniques for log template identification, making it difficult to select the best technique for a given application context.

Contributions. In this chapter, we address the above issues by (1) providing guidelines for assessing the accuracy of log message template identification techniques and (2) assessing the application of such guidelines through a comprehensive evaluation of 14 existing template identification techniques.

More specifically, to address the issue related to the choice of accuracy metrics, we first describe new metrics, called *Template Accuracy* (TA) metrics, that, unlike GA and PA, are not sensitive to the number of log messages. We then discuss which metrics (GA, PA, or TA) are more appropriate depending on the nature of the software engineering task in which log message templates are used. The idea behind TA metrics is that template identification should be regarded as an information retrieval process in which message templates are identified from a collection of log messages. Therefore, based on standard information retrieval metrics, i.e., precision and recall, we define *Precision-TA* (PTA) and *Recall-TA* (RTA) metrics for template identification. As for determining oracle templates, we propose a set of heuristic rules for correcting oracle templates in order to minimize the negative bias in experimental results introduced by manually generated oracle templates.

[1]The metric is called Parsing Accuracy in the original paper [ZHL+19]; in this chapter, we follow the naming convention proposed by Dai et al. [DLC+20].

Furthermore, to provide additional information about incorrectly identified templates for a detailed analysis, we propose a technique for analyzing incorrect templates.

From an empirical software engineering point of view, we assess the application of the proposed guidelines by investigating, using a benchmark composed of 14 existing template identification techniques, the following research questions:

RQ1: How does the ranking of techniques vary when using different accuracy metrics?

RQ2: What is the impact of oracle template correction on different accuracy metrics?

RQ3: Can the analysis of incorrect templates provide any insight to improve template identification techniques?

Significance. Log message template identification is an essential pre-processing step for automated log analysis research and practice. Therefore, adopting an adequate evaluation and comparison methodology for log message template identification is of great importance for both researchers and practitioners. The contributions of this chapter can impact the field of log analysis by providing practical guidelines for accurately selecting the most adequate template identification techniques for a given software engineering application. Specifically, our empirical evaluation results show that following the guidelines is indeed critical in assessing and comparing the accuracy of log template identification techniques. The results also shed very different insights than existing studies and in particular a much less optimistic outlook on existing techniques. The insights will be useful for improving log template identification techniques in future research.

Chapter structure. The rest of the chapter is organized as follows. Section 3.2 reviews the state of the art and motivates our work with a running example. Section 3.3 describes the guidelines for assessing the accuracy of log template identification techniques. Section 3.4 reports on the evaluation results and Section 3.5 concludes the chapter.

3.2 Motivation

Given the number of available log template identification techniques, it is important to evaluate and rank them using all relevant criteria (e.g., accuracy, execution time); in this chapter, we focus on the evaluation of the *accuracy*, informally defined as a measure of the ability of a technique to correctly identify log templates. In particular, our starting point are the two notable studies of Zhu et al. [ZHL+19] and Dai et al. [DLC+20], which represent the state of art in empirical software engineering in terms of assessment of the accuracy of template identification techniques.

Zhu et al. [ZHL+19] provided the first comprehensive evaluation of 13 log template identification techniques using 16 real-world log datasets. They made the replication package publicly available, including the implementation of the log template identification techniques, the real-world log datasets, and the oracle templates for the logs. Using these artifacts, Zhu et al. [DLC+20] compared their new technique (called Logram) with the five techniques that achieved the highest accuracy scores reported by Zhu et al. [ZHL+19]. Dai et al. [DLC+20] also proposed a new accuracy metric after raising an issue about the accuracy metric previously used by Zhu et al. [ZHL+19]. Though both studies made essential steps towards a comprehensive assessment of the accuracy of log template identification techniques, we observed three important issues: 1. the choice of accuracy metrics, 2. determining oracle templates, and 3. incorrectly identified templates. The three issues are discussed in detail in the following subsections.

3.2.1 The Choice of Accuracy Metrics

The choice of accuracy metrics is naturally of great importance in evaluating the accuracy of log template identification techniques. Initially, Zhu et al. [ZHL+19] used the *Grouping Accuracy* (GA) metric to assess the accuracy of log template identification. The idea behind the GA metric is that template identification can be regarded as a clustering process in which log messages with different log events are clustered into different groups. Therefore, this metric checks if log messages that are grouped together by having the same identified template indeed form the same group as in the ground truth. Specifically, the GA metric is defined as the ratio of "correctly parsed" log messages (thanks to the identified templates) over the total number of log messages, where a log message is considered "correctly parsed" if and only if it is grouped with other log messages in a way consistent with the ground truth. However, Dai et al. [DLC+20] used another metric, called *Parsing Accuracy* (PA), since the GA metric does not consider whether the identified templates are identical to the oracle ones but only accounts for how the identified templates support the log message grouping activity. The PA metric is defined as the ratio of "correctly parsed" log messages over the total number of log messages (the same as for the GA metric), where a log message is con-

Table 3.1: Log messages, oracle templates, and templates identified by two techniques *A* and *B* in our running example

	Template			GA		PA	
Message (M_{ex})	Technique *A* (T_A)	Technique *B* (T_B)	Oracle (O)	Technique *A*	Technique *B*	Technique *A*	Technique *B*
(m_1) `retry 1`	(t_1^A) `retry <*>`	(t_1^B) `retry 1`	(o_1) `retry <*>`	✓	✗	✓	✗
(m_2) `retry 2`	(t_1^A) `retry <*>`	(t_2^B) `retry 2`	(o_1) `retry <*>`	✓	✗	✓	✗
(m_3) `x is 3.5`	(t_2^A) `x is <*>.<*>`	(t_3^B) `x is <*>`	(o_2) `x is <*>`	✓	✓	✗	✓
(m_4) `tot 4 MB`	(t_3^A) `<*> <*> MB`	(t_4^B) `<*> <*> MB`	(o_3) `tot <*> MB`	✗	✗	✗	✗
(m_5) `usd 1 MB`	(t_3^A) `<*> <*> MB`	(t_4^B) `<*> <*> MB`	(o_4) `usd <*> MB`	✗	✗	✗	✗
(m_6) `adr=0xff`	(t_4^A) `<*>=0xff`	(t_5^B) `adr=<*>`	(o_5) `adr=<*>`	✓	✓	✗	✓

sidered to be "correctly parsed" if and only if "all its static text and dynamic variables (i.e., fixed and variables parts) are correctly identified" [DLC+20, p. 7]. As a result, the GA and PA metrics return different results in assessing the accuracy of log template identification techniques.

To better understand the GA and PA metrics, let us consider the scenario in which a software engineer (or a researcher) wants to assess the accuracy of two template identification techniques (called A and B), using the set of example log messages $M_{ex} = \{m_1, \ldots, m_6\}$ and the corresponding set of oracle (i.e., ground truth) templates $O = \{o_1, \ldots, o_5\}$ in Table 3.1; the example is based on a simplified version of real log messages and templates extracted from the log datasets provided by Zhu et al. [ZHL+19]. Running the implementation of each of the two techniques on M_{ex} yields the corresponding set of identified templates $T_A = \{t_1^A, \ldots, T_4^A\}$ and $T_B = \{t_1^B, \ldots, T_5^B\}$, as shown Table 3.1. Notice that one template matches one or more messages, e.g., template t_1^A matches both m_1 and m_2.

According to the definition of correctly parsed log messages adopted for computing the GA metric, messages m_1 and m_2 can be correctly parsed by technique *A* because , according to t_1^A, they are exactly grouped together as they would be by o_1 (i.e., the oracle template of m_1 and m_2). However, message m_5 cannot be correctly parsed using t_3^A since the latter groups m_5 with another message m_4 while o_4 (i.e., the oracle template of m_5) does not. For readability, the correctly parsed log messages are marked under column *GA* in Table 3.1. Overall, only four messages (i.e., m_1, m_2, m_3, and m_6) out of six can be correctly parsed using the templates in the set $T_A = \{t_1^A, \ldots, t_4^A\}$, resulting in a GA score for technique *A* equal to $\frac{4}{6} \approx 0.67$. Similarly, the GA score for technique *B* is $\frac{2}{6} \approx 0.33$ since only two messages (i.e., m_3 and m_6) out of six can be correctly parsed using the templates in the set $T_B = \{t_1^B, \ldots, t_5^B\}$. Just by looking at the GA score, one could think that technique A is better than technique *B*.

However, even if template t_4^A can be used to correctly parse message m_6 in terms of the GA metric, it is quite different from oracle template o_5. More specifically, template o_5 contains string `adr` in the fixed part, informally suggesting that this template matches log messages that record different values for the memory location `adr`. On the other hand, the fixed part of template t_4^A is represented by string `0xff`, meaning that this template will match log messages that record the value `0xff` at any memory location. Such a limitation of the GA metric is addressed in the PA metric, which considers both the fixed and variable parts of identified templates. In the same running example, the PA score for technique *A* is $\frac{2}{6} \approx 0.33$ since two messages (i.e., m_1 and m_2) out of six can be correctly parsed by the template t_1^A (since t_1^A is identical to its oracle template o_1 in both fixed and variable parts). Similarly, the PA score for technique *B* is $\frac{2}{6} \approx 0.33$ since two messages (i.e., m_3 and m_6) out of six can be correctly parsed by the templates t_3^B and t_5^B. For readability, the log messages correctly parsed in terms of PA are marked under column *PA* in Table 3.1. By looking at the PA score, one could think that both of the techniques *A* and *B* are equally accurate in log template identification, which is clearly different from the result obtained when using the GA metric.

While the GA and PA metrics are clearly different, both metrics share the same issue: they are sensitive to the number of messages contained in the log (and not to the number of templates). This can be problematic for systems whose logs contain repeated log messages that are not important in terms of the system's business logic. For example, if a log contains many heartbeat messages that are repeated every second, then the GA and PA scores of a template identification technique can appear sufficiently high even if the technique is only able to correctly identify one template for the heartbeats. In our running example, we can already see that, though technique *A* correctly identifies only one template (i.e., t_1^A), the PA score is $\frac{2}{6} \approx 0.33$ since two messages (i.e., m_1 and m_2) out of six can be correctly parsed by the template t_1^A. If we focus on the number of correctly identified templates, technique *B* is "better" than technique *A* since technique *B* correctly identifies two templates (t_3^B and t_5^B).

We want to note that choosing a template identification technique based on the GA metric is not an issue per se when a template identification technique is only used for grouping log messages based on the identified templates. For example, for performance anomaly detection [NMA+16, JYC+17], it is sufficient to monitor occurrence patterns (e.g., a repetition of the same event in a short period of time) of the (events corresponding to) log messages, without considering the message parameter values. However, in other scenarios in which the parameter values of messages (e.g., `1` in m_1) matter—such as model inference with guard conditions [WTD16] and advanced anomaly detection using parameter values [DLZS17]—it is important to choose a log template identification technique based on another accuracy metric that accounts for the correctness of both fixed and variable parts in identified templates. Section 3.3.1.3 further describes how to choose appropriate metrics in detail.

To address the limitations of the GA and PA metrics, in Section 3.3.1 we propose to use alternative metrics for assessing the accuracy of template identification techniques. We then replicate the comprehensive evaluations of Zhu et al. [ZHL+19] and Dai et al. [DLC+20] in Section 3.4.2, additionally using different accuracy metrics, including the proposed alternative ones, to better understand how and why the ranking of log template identification techniques vary when using different techniques.

3.2.2 Determining Oracle Templates

Oracle templates are essential to evaluate the accuracy of template identification techniques. Both of the existing studies used the same oracle templates generated by Zhu et al. [ZHL+19]. The oracle templates were determined by manually inspecting log messages. However, when we compare them with the corresponding log printing statements in the source code, the oracle templates are not always correct. For example, for a log message "`status is false`" generated by a log printing statement `log.info("status is " + var)`, an engineer with no access to the source code has mistakenly defined an oracle template "`status is false`" instead of "`status is <*>`", by incorrectly assuming that the token "`false`" was hard-coded in the log printing statement.

Nevertheless, manually determining oracle templates (in the context of the study by Zhu et al. [ZHL+19]) was unavoidable since some of the benchmark log datasets were collected from real-world systems that do not provide access to the source code. Furthermore, from a more pragmatic perspective, this strategy is acceptable considering the fact that manually generated oracle templates are, in most cases, very similar to the actual templates extracted from the source code. However, the impact of using slightly incorrect oracle templates on the assessment of the accuracy of template identification techniques may vary depending on the accuracy metrics used. For example, the GA metric only accounts for how log messages are grouped and does not consider whether the Technique Under Evaluation (TUE) correctly identifies templates by accurately decomposing fixed and variable parts; therefore, one could speculate that the impact of using slightly incorrect oracle templates could be limited. Proving such assumptions is an open problem.

To correct such potentially-incorrect oracle templates without accessing the source code, in Section 3.3.2, we propose a set of heuristic rules that can automatically correct many oracle templates. Furthermore, we empirically investigate the impact of using the oracle template correction on different accuracy metrics in Section 3.4.3.

3.2.3 Incorrectly Identified Templates

Regardless of the accuracy metric used, simply ranking the log template identification techniques by their accuracy scores may not be sufficient. It is also important to analyze why the techniques incorrectly identify some templates. For example, let us consider a technique that has a low accuracy score only because it does not correctly recognize IP addresses as variable parts. In practice, such an issue can be easily addressed by modifying the TUE to support user-defined regular expressions for pre-processing structured text strings like IP addresses. However, without additional information about incorrectly identified templates, the TUE could be perceived as being severely inaccurate, even if this is not really the case.

However, both existing studies did not provide any information regarding the templates incorrectly identified by template identification techniques. Such missing information hinders the possibility to improve existing techniques by addressing the limitations discovered by experimental assessment.

To address this issue, in Section 3.3.3, we propose a technique for analyzing incorrect templates that can help software engineers and researchers understand in which way templates are incorrectly identified. We also discuss the insights one can get by applying the proposed analysis technique to the empirical evaluation results presented in Section 3.4.4.

3.3 Guidelines

In this section, to address the issues discussed in Section 3.2, we present three guidelines for assessing the accuracy of log template identification techniques:

1. Do use appropriate accuracy metrics (§ 3.3.1),
2. Do perform oracle template correction (§ 3.3.2), and
3. Do perform analysis of incorrect templates (§ 3.3.3).

3.3.1 Do Use Appropriate Metrics

As discussed in Section 3.2.1, since existing accuracy metrics rely on different definitions for “correctly parsed” log messages, it is essential to use a metric that adequately assesses the accuracy of template identification techniques in the context of the targeted use cases. Before we present our guidelines for choosing an appropriate accuracy metric considering a use case, we first introduce new metrics to address the common limitations of the existing metrics.

3.3.1.1 Template Accuracy (TA) Metrics

Recall that both GA and PA are sensitive to the number of messages contained in the log, which is possibly misleading when there are many repetitive yet useless messages, commonly found in real-world system logs. To address this, we propose alternative accuracy metrics that take into account how many *templates* are "correctly" identified by the TUE.

First, we define the concept of *correctly identified* templates. A template (determined by a template identification technique) is *correctly identified* from log messages if and only if it is identical (token-by-token) to the oracle template(s) of the log messages. In other words, the correctness of an identified template must be determined by comparison to its "corresponding" oracle template(s).

More precisely, let M be a set of log messages, O be the set of oracle templates for M, and T_v be a set of templates for M identified by a TUE v. We introduce two auxiliary functions: $\mathit{msg}\colon T_v \to 2^M$, which, given an identified template t, returns the set of messages from M whose template is t; $\mathit{ot}\colon M \to O$, which, given a message m, returns the oracle template of m included in O. We define the *set of corresponding oracle templates in O for a template t*, denoted and defined as $\mathit{corrOT}(t) = \{\mathit{ot}(m) \mid m \in \mathit{msg}(t)\}$. For instance, in our running example M_{ex}, O_{ex}, and T_A shown in Figure 3.1, $\mathit{corrOT}(t^A_1)$ is $\{o_1\}$ because $\mathit{msg}(t^A_1) = \{m_1, m_2\}$ and $\mathit{ot}(m_1) = \mathit{ot}(m_2) = o_1$.

Based on the definition of corresponding oracle templates, we say that a template $t \in T_v$ is *correctly identified* (*correct*, for simplicity) by TUE v if and only if $\mathit{corrOT}(t) = \{t\}$; otherwise, we say t is *incorrectly identified* (*incorrect*, for simplicity) by v. For the running example above, t^A_1 is correct since $\mathit{corrOT}(t^A_1) = \{o_1\} = \{t^A_1\}$; on the other hand, t^A_3 is incorrect since $\mathit{corrOT}(t^A_3) = \{o_3, o_4\} \neq \{t^A_3\}$.

Notice that our definition of correctly identified template relies on the set of oracle templates corresponding to a certain message. It is not enough that the identified template is included in the set of oracle templates O; to be correct, the identified template has to be identical to the one and only oracle template *corresponding* to the messages for which it was identified. For instance, in our running example, even if there were an additional oracle template $o_6 \in O'_{ex}$ (where $O'_{ex} = O_{ex} \cup \{o_6\}$) that is identical to t^A_4 such that

$t_4^A \in T_A \cap O'_{ex}$, t_4^A could not be considered correct because $corrOT(t_4^A) = \{o_5\} \neq \{t_4^A\}$.

Using the definition of correctly identified template, we introduce two TA metrics: *Precision-TA* (PTA) and *Recall-TA* (RTA) — collectively denoted by PTA‡RTA — which are based on the standard metrics *precision* and *recall* used in the information retrieval domain. PTA is defined as $\frac{|\{t \in T_v | corrOT(t)=\{t\}\}|}{|T_v|}$, the *ratio of correctly identified templates over the total number of identified templates*, indicating the *precision* of the TUE at the template level. RTA is defined as $\frac{|\{t \in T_v | corrOT(t)=\{t\}\}|}{|O|}$, the *ratio of correctly identified templates over the total number of oracle templates*, indicating the *recall* of the TUE at the template level.

These two metrics range between 0 and 1. For our running example, in the case of technique *A*, its PTA score is $\frac{1}{4} = 0.25$ because only one template (i.e., t_1^A) out of the four in T_A is correct; the RTA score is $\frac{1}{5} = 0.2$ since the total number of oracle templates in O_{ex} is five. In the case of technique *B*, its PTA score is $\frac{2}{5} = 0.4$ because two templates (i.e., t_3^B and t_5^B) out of the five in T_B are correct; the RTA score is $\frac{2}{5} = 0.4$.

3.3.1.2 Relationship among the Metrics

There are interesting relationships among GA, PA, and PTA‡RTA. Specifically, when PTA = RTA = 1 we have GA = 1 because log message groups determined by oracle templates must be the same as the groups determined by identified templates, given that all identified templates are correct and the total number of identified templates is the same as the number of oracle templates. For the same reason, when PA = 1 we have GA = 1. On the other hand, having GA = 1 does not imply PTA = 1, RTA = 1, or PA = 1 since completely incorrect templates can group messages in the same way as oracle templates do (like t_4^A in our running example). Meanwhile, since PTA‡RTA and PA consider the correctness of fixed and variable parts of identified templates (though PA counts correctly parsed log messages while PTA‡RTA count correctly identified templates), having PTA = RTA = 1 imply PA = 1, and vice versa.

3.3.1.3 How to Choose Appropriate Metrics

Given a target use case, one should choose an appropriate metric to evaluate the accuracy of the TUE. The two important criteria to consider when performing this choice are 1. whether the variable parts of log messages are used and 2. whether the importance of a message is proportional to its frequency.

When the variable parts of log messages are not used in the target use case, then one should use the GA metric. For example, the log key anomaly detection model of DeepLog [DLZS17] aims to detect behavioral anomalies using the system's event sequences recorded in the log. It uses log template identification to convert a sequence of log messages into a sequence of log keys (i.e., template ids). Therefore, the GA metric — that only considers message grouping — is the best to assess the accuracy of template identification in this use case.

When the variable parts of log messages matter in the target use case, one should additionally consider the second criterion above: if the importance of a log message is proportional to its frequency, then PA should be used; otherwise, PTA‡RTA should be used. This is because PA is sensitive to the proportion of log messages whereas PTA‡RTA are not. For example, DeepLog [DLZS17] has another model for detecting irregular parameter values, which is not detected by the above log key anomaly detection model. Since the irregular parameter detection model needs different parameter values (i.e., variable parts) of log messages accurately extracted by log template identification, GA cannot be used to adequately assess the accuracy of template identification in this case. Furthermore, if the importance of system events recorded in the log does not depend on the frequency of the event messages, then using PA can yield misleading accuracy evaluation results, as PA counts the number of log messages parsed by correctly identified templates. In this case, PTA‡RTA should be used to assess the accuracy of the TUE.

To summarize, by evaluating the two aforementioned criteria in the context of the target use cases, one can choose an appropriate metric among GA, PA, and PTA‡RTA to assess the accuracy of log template identification techniques.

3.3.2 Do Perform Oracle Template Correction

As discussed in Section 3.2.2, when oracle templates are manually determined (e.g., because there is no access to the source code), they may be incorrect. To address this issue, we propose to perform oracle template correction using heuristic rules.

3.3.2.1 Heuristic-based Oracle Template Correction

The idea of correcting oracle templates is based on our analysis of manually determined oracle templates provided by Zhu et al. [ZHL^{+}19], a publicly available, widely-used template identification benchmark. After a de-

Table 3.2: Heuristic rules for oracle template correction

Rule	Description	Example (before → after)
Double Spaces (DS)	Replace double spaces with a single space	`Input:␣␣<*>` → `Input:␣<*>`
Digit (DG)	Replace digit tokens with variables	`euid=0` → `euid=<*>`
Boolean (BL)	Replace True/False with a variable	`cancel=false` → `cancel=<*>`
Path String (PS)	Replace a path-like token with a variable	`/lib/tmp started` → `<*> started`
User-defined String (US)	Replace user-defined strings with variables	`status=idle` → `status=<*>`
Mixed Token (MT)	Replace a token containing both fixed and variable parts with a variable	`python v<*>` → `python <*>`
Consecutive Variables (CV)	Replace consecutive variables as a single variable	`value=<*><*>` → `value=<*>`
Dot-separated Variables (DV)	Replace dot separated variables as a single variable	`<*>.<*>` seconds → `<*>` seconds

tailed analysis of the oracle templates, we noticed that some of them contain fixed parts that are unlikely to be hard-coded in log printing statements. For example, one of the oracle templates for the Hadoop system in LogHub is "`nodeBlacklistingEnabled:true`"; hard-coding the value "`true`" in the log printing statement seems wrong. Indeed, we checked the source code of Hadoop and found that the actual log printing statement is "`LOG.info ("nodeBlacklistingEnabled:" + nodeBlacklistingEnabled)`", meaning that the correct oracle template is "`nodeBlacklistingEnabled:<*>`". Note that such an incorrect oracle template is not an issue for the GA metric, as long as the incorrect oracle template can group log messages in the same way as the correct oracle template does. However, the presence of such an incorrect oracle template may introduce a bias in the computation of the PA and TA metrics, which directly compare identified and oracle templates. Therefore, we propose a set of heuristic rules, based on the manual investigation of the oracle templates in LogHub, that converts fixed parts into variable parts using regular expressions. The regular expressions are provided in our replication package (§ 3.4.7).

Table 3.2 shows the rules, their descriptions, and simple application examples. The DS (Double Space) rule replaces any double (or more) whitespace with a single whitespace to eliminate trivial whitespace differences between oracles and identified templates. The DG (DiGit), BL (BooLean), PS (Path String), and US (User-defined String) rules replace a fixed token, such as digits, Boolean values, paths, and user-specific strings, with the placeholder symbol "`<*>`"; the rationale behind these rules is that the type of token they try to represent is rarely hard-coded in log printing statements. Unlike the others, rule US is made for users who can indicate specific strings that should be replaced with "`<*>`". For example, a string "`idle`" could be hard-coded in a log printing statement in a system, such as `log.info("system status is idle")`; however, for another system, "`idle`" could be a value for variable "`ret`" in a log printing statement like `log.info("status=" + ret)`. If all log messages containing "`idle`" have the form "`...=idle`", one could be drawn to consider token "`idle`" as a variable part in the template.

Table 3.3: Number of oracle templates corrected using our heuristic rules for the benchmark provided by Zhu et al. [ZHL^{+}19]

Dataset	—O—	DS	DG	BL	PS	US	MT	CV	DV
HDFS	14	1	0	0	7	0	0	0	0
Hadoop	114	6	4	2	3	1	3	0	5
Spark	36	0	0	0	0	0	0	0	0
Zookeeper	50	1	0	0	10	0	1	0	0
OpenStack	43	0	0	0	0	0	0	0	22
BGL	120	0	2	0	6	0	2	0	6
HPC	46	7	0	0	0	0	3	0	0
Thunderbird	149	0	4	0	9	10	5	3	37
Windows	50	0	0	2	0	1	2	0	10
Linux	118	12	13	0	3	3	6	2	41
Mac	341	20	0	5	9	13	6	4	167
Android	166	10	0	63	0	12	0	4	26
HealthApp	75	3	0	10	0	0	0	0	2
Apache	6	0	0	0	0	0	0	0	0
OpenSSH	27	1	0	0	0	4	0	0	0
Proxifier	8	0	0	0	0	0	0	0	0
Total	1363	61	23	82	47	44	28	13	316

In this case, “`idle`” can be indicated as one of the user-defined strings for the application of rule US. The MT (Mixed Token), CV (Consecutive Variables), and DV (Dot-separated Variables) rules replace a token containing variable parts (such as “`v<⋆>`”, “`<⋆><⋆>`”, and “`<⋆>.<⋆>`”) with “`<⋆>`” because the latter catches all the strings caught by the former, without loss of information. For instance, “`v<⋆>`” is used in templates to indicate version information (e.g., “`v2`”), “`<⋆><⋆>`” usually indicates a number with a unary operator (e.g., “`-1`”), and “`<⋆>.<⋆>`” usually indicates a floating-point number (e.g., “`0.5`”). In all cases, these tokens can be simply replaced by “`<⋆>`”, which captures all the strings captured by the original tokens.

Table 3.3 shows the number of corrected oracle templates for each dataset in the benchmark provided by Zhu et al. [ZHL^{+}19], when each of our heuristic rules is applied to the manually determined oracle templates (columns DS, DG, BL, PS, US, MT, CV, and DV); column $|O|$ shows the number of manually determined oracle templates for each dataset in the benchmark. In summary, for a total of 1363 oracle templates in the benchmark, each rule corrected from a minimum of 13 (CV) to a maximum of 316 (DV) oracle templates. We manually checked the soundness of all the corrected oracle templates. Nevertheless, we cannot guarantee that they are truly consistent with the corresponding log printing statements in the source code, which is unknown to us. Section 3.4.6 will further discuss this issue.

3.3.2.2 Using Oracle Template Correction

The proposed oracle template correction is meant to be used when oracle templates are manually determined (e.g., when the source code is inaccessible). In such a case, the oracle template correction should be used to make possibly error-prone oracle templates more consistent with the general form of log printing statements in the source code.

When manually determining oracle templates, the heuristic rules for oracle template correction can be already taken into account. For example, given a log message "`status is false`", one can identify an oracle template "`status is <*>`" by considering the rationale behind the BL rule.

3.3.3 Do Perform Analysis of Incorrect Templates

As discussed in Section 3.2.3, simply ranking different log template identification techniques by their accuracy scores may not be sufficient, since such a ranking does not provide any insight about the reason for which a TUE got a certain accuracy score. We propose to perform an *analysis of incorrect templates* to further analyze the incorrectly identified templates to understand in which way they are incorrect; this analysis may provide insights to engineers and researchers on how to improve the TUE.

3.3.3.1 Analysis of Incorrect Templates

This analysis is based on the observation that template identification can also be seen as the process of generalizing log messages by converting fixed parts into variable parts. In our running example, shown in Figure 3.1, technique A generalizes the two messages m_1 ("`retry 1`") and m_2 ("`retry 2`") to the template t_1^A ("`retry <*>`") by converting the second token of both messages into a variable. Thus, incorrectly identified templates can be seen as incorrectly generalized templates. Based on this idea, we classify incorrect templates into three types: *Over-Generalized* (OG), *Under-Generalized* (UG), and *MiXed* (MX). By doing this, we can understand whether the TUE suffers from over-generalization, under-generalization, or both.

The core of this three-way classification is the *generalization* relationship between templates, which is defined in terms of the formal language defined by the templates. Since a template can be seen as a regular expression[2], we

[2]Recall that "<*>" represents any characters except whitespace.

denote by $lang(t)$ the regular language defined by a template t. We say that a template t_x is *more general* than a template t_y if and only if $lang(t_y) \subset lang(t_x)$. In our running example, o_2 is more general than t_2^A because all potential log messages that match t_2^A will also match o_2 but the opposite does not hold (e.g., message "`x is 1`" is in $lang(o_2)$ but not in $lang(t_2^A)$).

Based on the generalization relationship between templates, we can classify an incorrectly identified template as OG, UG, or MX. Specifically, an incorrectly identified template t is classified as OG if t is more general than all oracle templates in $corrOT(t)$, i.e., t is always more general than its corresponding oracle templates. Similarly, t is classified as UG if all oracle templates in $corrOT(t)$ are more general than t, i.e., t is always less general than its corresponding oracle templates. If t is classified neither as OG nor UG, it is classified as MX, meaning t is over-generalized in some cases while under-generalized in others.

In our running example, template t_2^A is UG because $corrOT(t_2^A) = \{o_2\}$ and o_2 is more general than t_2^A. On the other hand, template t_3^A is OG because $corrOT(t_3^A) = \{o_3, o_4\}$ and t_3^A is more general than both o_3 and o_4. Template t_4^A is MX because $corrOT(t_4^A) = \{o_5\}$ and neither t_4^A nor o_5 is more general than the other. Note that, with respect to o_5, t_4^A is over-generalized because of the position of `<*>` but is also under-generalized because of the presence of token `0xff`. Based on this analysis, the three incorrect templates of technique A cover all the three types, meaning that technique A needs to be improved in all aspects.

3.3.3.2 Using Incorrect Template Analysis

The proposed incorrect template analysis can be performed without requiring additional inputs other than oracle templates and identified templates, which are already used to compute accuracy scores. Furthermore, as it is based on the formal definition of the generalization relationship between templates, it can be easily automated. We remark that the proposed incorrect template analysis is independent from the choice of accuracy metrics.

3.4 Evaluation

In this section, we assess the application of the guidelines proposed in the previous section, using 14 existing template identification techniques. Specifically, we answer the following research questions, already introduced in Section 3.1:

RQ1: How does the ranking of techniques vary when using different accuracy metrics?

RQ2: What is the impact of oracle template correction on different accuracy metrics?

RQ3: Can the analysis of incorrect templates provide any insight to improve template identification techniques?

3.4.1 Benchmark and Settings

To answer the research questions, we used the publicly available benchmark proposed by Zhu et al. [ZHL$^+$19] to assess different template identification techniques. This benchmark is based on the LogHub benchmark [HZHL20], which contains a large collection of log messages from 16 real-world systems including distributed systems, operating systems, mobile systems, and standalone programs. To determine the ground truth in terms of templates, Zhu et al. [ZHL$^+$19] randomly sampled 2000 messages for each system and manually labeled them with oracle templates. Table 3.4 provides an overview of the 16 datasets (each of them containing 2000 log messages and the corresponding manually generated templates) in the benchmark; column $|O|$ shows the number of oracle templates for each dataset. To support reproducibility, Zhu et al. [ZHL$^+$19]'s benchmark also includes the implementation of 13 log template identification techniques (i.e., AEL [JHFH08], Drain [HZZL17], IPLoM [MZHM09], LenMa [Shi16], LFA [NV10], LKE [FLWL09], LogCluster [VP15], LogMine [HDX$^+$16], LogSig [TLP11], MoLFI [MPB$^+$18], SHISO [Miz13], SLCT [Vaa03], Spell [DL16]).

In our evaluation, we used the same logs, oracle templates, implementations of template identification techniques, and parameter settings (for each template identification technique) used by Zhu et al. [ZHL$^+$19]. We also added the implementation of Logram [DLC$^+$20] to our benchmark and used the parameter values suggested by the authors. In total, we considered 14 log template identification techniques. Additionally, as discussed in Section 3.3.2, the use of the US (User String) rule requires user-defined strings as input. Based on our analysis of the templates in the Zhu et al. [ZHL$^+$19]'s benchmark, we used the following tokens as user-defined strings: `null`, `root`, and `admin`.

Table 3.4: Log datasets (each dataset has 2000 messages, —O— indicates the number of oracle templates)

Dataset	Description	$\|O\|$
HDFS	Hadoop distributed file system log	14
Hadoop	Hadoop mapreduce job log	114
Spark	Spark job log	36
ZooKeeper	ZooKeeper service log	50
OpenStack	OpenStack software log	43
BGL	Blue Gene/L supercomputer log	120
HPC	High performance cluster log	46
Thunderbird	Thunderbird supercomputer log	149
Windows	Windows event log	50
Linux	Linux system log	118
Mac	Mac OS log	341
Android	Android framework log	166
HealthApp	Health app log	75
Apache	Apache server error log	6
OpenSSH	OpenSSH server log	27
Proxifier	Proxifier software log	8

3.4.2 RQ1: Ranking of Techniques

3.4.2.1 Methodology

To answer RQ1, we executed the 14 log template identification techniques on each dataset in the benchmark and used their output to compute the accuracy in terms of the GA, PA, and TA (PTA and RTA) metrics. For each execution (of a technique on a dataset), we set a timeout of 24 hours. To account for the randomness of LKE and MoLFI, we repeated the corresponding experiments 30 times and computed the average results. Notice that we did not use oracle template correction because it will be investigated in RQ2.

3.4.2.2 Results

All techniques completed their execution on all datasets, except in one case: the execution of IPLoM on the Android dataset timed out. Figure 3.1 shows the GA, PA, and PTA‡RTA score distributions of the individual techniques across all datasets for which the techniques completed their execution. Overall, it is clear that GA scores tend to be higher than PA and PTA‡RTA scores for all techniques. This means that, for all techniques, there are many identified templates that are incorrect but happen to be able to group log messages in a way consistent with the ground truth. This implies that the GA metric

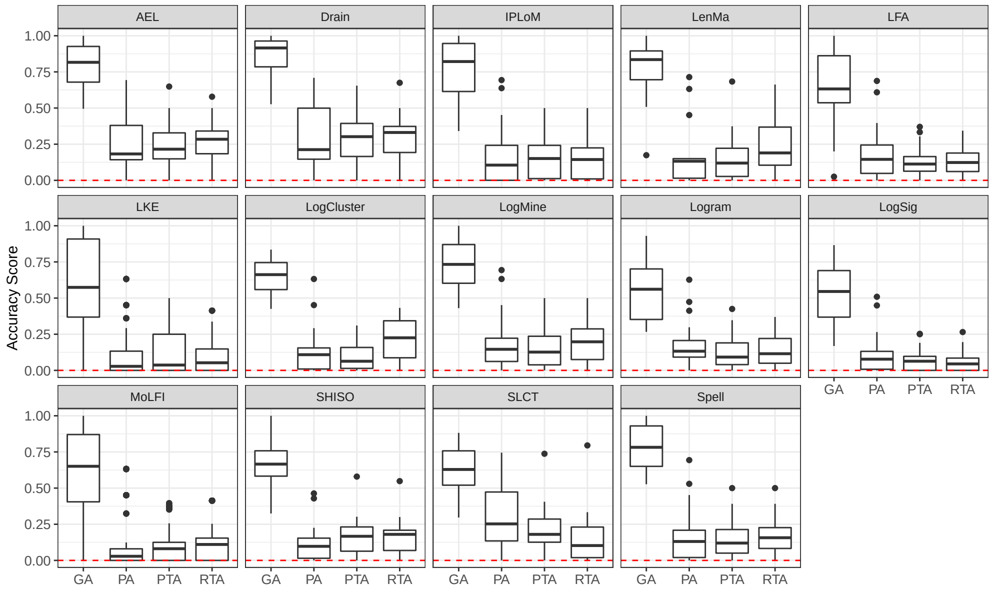

Figure 3.1: Accuracy results using the different metrics

is inadequate to measure the accuracy of a template identification technique when identifying correct templates (both fixed and variable parts) is important.

Table 3.5 and Table 3.6 shows the ranking of the different techniques in terms of the GA, PA, and PTA‡RTA scores. The techniques are sorted, in descending order, by average accuracy score (computed over the 16 datasets). We can see that Drain is ranked 1st for GA and PTA‡RTA, but not for PA. In other words, Drain outperforms all other techniques, both at correctly grouping messages and at identifying correct templates. However, SLCT outperforms Drain in terms of PA, meaning that SLCT is better than Drain at identifying correct templates when the number of log messages corresponding to the correctly identified templates is also considered. Notice that SLCT is ranked 10th for GA but 1st for PA. This is not the only case where the ranking of the techniques differs depending on the metric used; for example, LenMa is better than AEL in terms of GA, but AEL is better than LenMa in terms of PA and PTA‡RTA. This happens because LenMa correctly groups more log messages than AEL, even though AEL correctly identifies more templates than LenMa. Such results strongly imply that the choice of the accuracy metric matters a lot when comparing log template identification techniques.

To assess how the ranking of the techniques varies when using the different accuracy metrics, we measured Spearman's rank correlation between

Table 3.5: Ranking of log identification techniques based on the GA, PA, and PTA‡RTA metrics (without Oracle Template Correction)

Ranking without Oracle Template Correction							
GA		PA		PTA		RTA	
Technique	Score	Technique	Score	Technique	Score	Technique	Score
Drain	0.87	SLCT	0.30	Drain	0.27	Drain	0.29
Spell	0.79	Drain	0.29	AEL	0.25	AEL	0.27
AEL	0.79	AEL	0.26	SLCT	0.22	LenMa	0.23
LenMa	0.77	LogMine	0.20	SHISO	0.16	LogCluster	0.22
IPLoM	0.76	LFA	0.20	LenMa	0.16	LogMine	0.20
LogMine	0.74	Logram	0.19	IPLoM	0.16	Spell	0.17
SHISO	0.68	IPLoM	0.19	LogMine	0.16	SHISO	0.17
LogCluster	0.65	Spell	0.18	Spell	0.15	SLCT	0.16
LFA	0.64	LenMa	0.18	LFA	0.14	IPLoM	0.15
SLCT	0.63	LogCluster	0.15	Logram	0.13	Logram	0.14
MoLFI	0.62	SHISO	0.13	LKE	0.12	LFA	0.14
LKE	0.56	LogSig	0.13	LogCluster	0.10	MoLFI	0.11
Logram	0.55	MoLFI	0.09	MoLFI	0.09	LKE	0.09
LogSig	0.53	LKE	0.08	LogSig	0.08	LogSig	0.07

Table 3.6: Ranking of log identification techniques based on the GA, PA, and PTA‡RTA metrics (with Oracle Template Correction)

Ranking with Oracle Template Correction							
GA		PA		PTA		RTA	
Technique	Score	Technique	Score	Technique	Score	Technique	Score
Drain	0.86	Drain	0.34	Drain	0.29	Drain	0.31
AEL	0.80	AEL	0.28	AEL	0.24	AEL	0.27
Spell	0.79	SLCT	0.27	SLCT	0.19	LenMa	0.22
LenMa	0.76	LFA	0.24	Spell	0.16	LogCluster	0.19
IPLoM	0.76	LogMine	0.23	IPLoM	0.16	LogMine	0.19
LogMine	0.74	Spell	0.20	LenMa	0.15	Spell	0.19
SHISO	0.68	IPLoM	0.19	SHISO	0.15	SHISO	0.16
LogCluster	0.65	Logram	0.17	LogMine	0.15	IPLoM	0.15
LFA	0.65	LenMa	0.16	LFA	0.15	LFA	0.15
LKE	0.63	LogCluster	0.13	Logram	0.13	Logram	0.14
SLCT	0.63	LogSig	0.11	LKE	0.10	SLCT	0.14
MoLFI	0.63	SHISO	0.11	LogCluster	0.09	MoLFI	0.11
Logram	0.55	LKE	0.09	MoLFI	0.08	LKE	0.10
LogSig	0.53	MoLFI	0.09	LogSig	0.07	LogSig	0.06

the rankings of each pair of accuracy metrics. A rank correlation coefficient ρ ranges between 0 and 1; the more similar the compared rankings, the higher the value of ρ. We found that, though the resulting rank correlation varies depending on the accuracy metrics compared, ranging from 0.455 (between PA and GA) to 0.846 (between GA and RTA), the average rank correlation remains moderate (0.666), implying that the ranking of different log template identification techniques can significantly vary depending on the accuracy metrics used.

To see whether the difference between Drain and the other individual techniques is statistically significant in terms of the GA and PTA‡RTA metrics, we additionally performed the Wilcoxon signed-rank test [Wil92], which is a non-parametric statistical hypothesis test for paired samples (i.e., in our context, pairs of techniques' accuracy scores across log datasets). Given a level of significance $\alpha = 0.05$, Drain is significantly better than the others only in terms of GA; the p-values comparing Drain and AEL in terms of PTA and RTA are 0.1519 and 0.0663, respectively. As a result, we can conclude that Drain is preferable only in terms of correctly grouping log messages as it is significantly better than the other techniques with respect to GA. However, we cannot say that Drain is significantly better than the others in terms of identifying correct templates.

To conclude, the answer to RQ1 is that the ranking of template identification techniques can vary a lot depending on the choice of accuracy metrics, since different metrics consider different aspects of template identification. When accuracy is measured using the PA and PTA‡RTA metrics, all template identification techniques achieve a low accuracy score (less than 50%), thus providing a much worse assessment than what was reported in existing studies [ZHL+19, DLC+20]. This implies that empirical studies using the GA metric were too optimistic due to the use of message-level grouping for calculating the accuracy scores of template identification techniques. This therefore calls for further research on accurate template identification techniques; some insights on how to improve existing techniques can be drawn from the results of our analysis of incorrect templates. In Section 3.4.4, we will show the analysis results and discuss the insights that can be drawn from them.

3.4.3 RQ2: Impact of Oracle Template Correction

3.4.3.1 Methodology

To answer RQ2, we assessed how the GA, PA, and PTA‡RTA scores change when using oracle template correction. To compute the accuracy scores, we followed the same methodology and settings used for RQ1, and *corrected the*

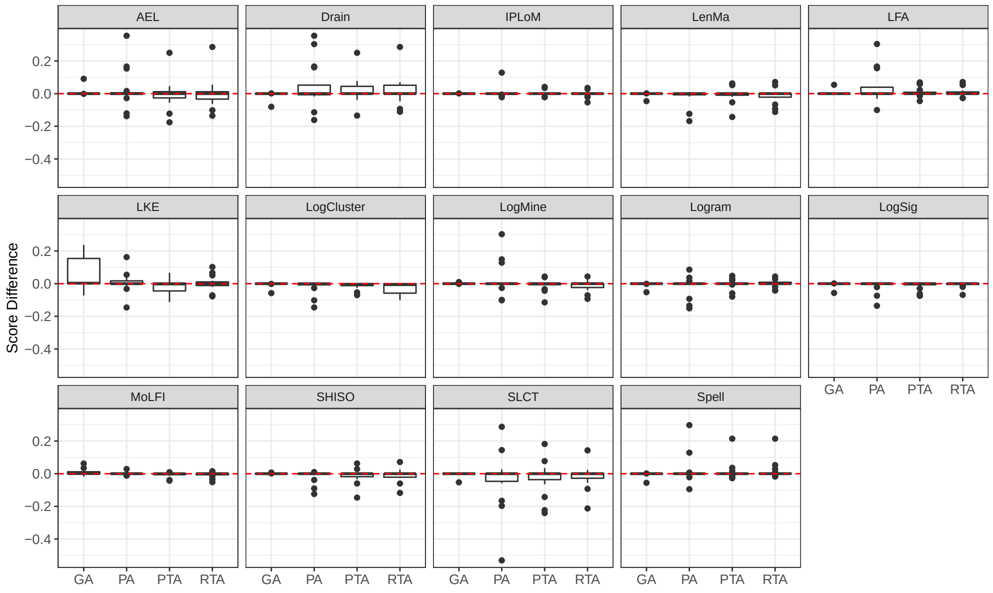

Figure 3.2: Impact of Oracle Template Correction

oracle templates applying the rules described in Section 3.3.2. Then, we calculated the difference between the accuracy scores obtained, hereafter denoted with the subscript X_{otc} (where $X \in \{GA, PA, PTA, RTA\}$), and those computed as part of answering RQ1, hereafter denoted with the subscript X_{org}. We computed the following differences: $\Delta^{GA}_{otc} = GA_{otc} - GA_{org}$, $\Delta^{PA}_{otc} = PA_{otc} - PA_{org}$, $\Delta^{PTA}_{otc} = PTA_{otc} - PTA_{org}$, $\Delta^{RTA}_{otc} = RTA_{otc} - RTA_{org}$.

3.4.3.2 Results

Figure 3.2 shows the distributions of the Δ^{GA}_{otc}, Δ^{PA}_{otc}, Δ^{PTA}_{otc}, and Δ^{RTA}_{otc} values for the individual techniques on all datasets, except IPLoM whose execution timed out on the Android dataset. For all $X \in \{GA, PA, PTA, RTA\}$, the difference between X_{otc} and X_{org} is statistically insignificant across log datasets based on the Mann–Whitney U test [MW47], with a level of significance $\alpha = 0.05$. This means that the impact of the oracle template correction on the GA, PA, and PTA‡RTA metrics is statistically insignificant, though there are some outliers as visible in the figure.

Given that many oracle templates (28.5% on average for all datasets) were incorrect and thus modified by applying oracle template corrections, it is surprising to see such a limited impact of the corrections. Through a more thorough analysis, we found that this was the case because many templates identified by the techniques are different enough from the corresponding oracle templates to such an extent that they are "incorrect" regardless of the corrections that can be applied to the oracle templates. This implies that the impact of oracle template corrections would be much larger if the techniques fared better at correctly identifying templates.

Notice that Δ_{otc}^{GA} is non-zero for many techniques, even though the GA metric ignores whether the identified templates are correct in terms of their fixed and variables parts. This is because the number of oracle templates (and therefore the number of log message groups generated by the oracle templates) can be changed by the oracle template correction step when two or more incorrect oracle templates become the same oracle template, after applying a correction.

Though the impact of the oracle template correction on the metrics is limited in terms of accuracy scores, the ranking of the techniques varies due to oracle template correction. Table 3.6 shows the ranking of the different techniques based on the accuracy scores with corrected oracle templates. We can see that Drain is now ranked 1st independently from the metric used to determine accuracy. Furthermore, based on the Wilcoxon signed-rank test [Wil92] with a significance level $\alpha = 0.05$, the difference between Drain and the other individual techniques is statistically significant for all the metrics. Compared to the results without the oracle template correction, the difference between Drain and the others becomes statistically significant for PA and PTA‡RTA, mainly because the score of Drain has increased more than that of the other techniques; for example, through the oracle template correction, the PA score of Drain has increased from 29% to 34% while that of SLCT has decreased from 30% to 27%. This happens because Drain identifies more templates that are identical to correct oracle templates, whereas SLCT identifies more templates that are identical to slightly incorrect oracle templates. As a result, we can conclude that Drain is preferable in all circumstances if the oracle template correction is used.

The answer to RQ2 is that, for all template identification techniques, enabling the oracle template correction has a limited impact on their accuracy scores in terms of GA, PA, and PTA‡RTA metrics. This is mainly because the templates identified by the techniques are, in general, quite different from oracle templates and therefore oracle template corrections have a limited effect. In other words, the limited impact of corrections are due to the relatively poor accuracy of the techniques, to various degrees. Nevertheless, the ranking of some template identification techniques, most particularly that of the best ones, changes when applying the oracle template correction. This implies that the oracle template correction may indeed be important for prop-

Table 3.7: Average Percentage of OG, UG, and MX Types

Technique	OG (%)	UG (%)	MX (%)
AEL	21.7	38.2	15.3
Drain	19.4	32.6	20.6
IPLoM	4.6	10.5	69.0
LFA	52.7	16.2	17.5
LKE	0.1	32.8	55.4
LenMa	5.5	44.5	33.8
LogCluster	0.0	72.9	17.0
LogMine	6.8	36.0	41.4
LogSig	0.1	14.2	77.7
Logram	27.3	26.4	33.2
MoLFI	0.0	8.4	82.5
SHISO	6.4	44.4	32.8
SLCT	17.5	28.9	31.7
Spell	7.7	13.1	64.2

erly ranking template identification techniques based on their accuracy, especially when the latter tends to be high.

3.4.4 RQ3: Insights from the Analysis of Incorrect Templates

3.4.4.1 Methodology

To answer RQ3, we followed the same methodology and settings used for RQ1; in addition, we classified incorrectly identified templates into OG (Over-Generalized), UG (Under-Generalized), and MiXed (MX) types, as described in Section 3.3.3.

3.4.4.2 Results

Table 3.7 shows the average percentage of OG, UG, and MX templates for the individual techniques on all datasets, except IPLoM whose execution timed out on the Android dataset. We can see that different techniques have different distributions of OG, UG, and MX types. The results provide insights on the main limitations of the techniques and how to improve them. For example, LFA generated over-generalized (OG) templates in 52.7% of the cases; in other words, LFA's main limitation is over-generalization. This information can then be used to improve the algorithm, e.g., by adjusting it to make it less likely to convert fixed tokens into variables. On the other hand, LogCluster — which generated under-generalized (UG) templates in 72.9% of the cases — could be improved by making it more likely to convert fixed tokens into variables. Techniques with many MX templates, such as MoLFI (82.5%), LogSig (77.7%), and IPLoM (69.0%), generated both under-generalized and over-generalized templates; they require a more thorough analysis to determine where to intervene for improving the underlying algorithms.

To conclude, the answer to RQ3 is that the analysis of incorrect templates can provide insights to improve template identification techniques by highlighting the limitations of individual techniques through the percentages of OG, UG, and MX types among identified templates.

3.4.5 Discussion

In RQ1, the results obtained for the GA metric are exactly the same as those computed by the replication package[3] of the original study [ZHL$^+$19]. However, the results obtained for the PA metric are quite different from those reported by Dai et al. [DLC$^+$20]. For example, on average for all datasets, Drain achieved a PA score of 29% (without oracle template correction, see Table 3.5), which is much lower than the PA score of 75% reported by Dai et al. [DLC$^+$20]. We could not reproduce their results since 1. they manually checked the parsing results to calculate PA values and 2. their replication package[4] contains an accuracy evaluation script only for calculating GA scores, not PA scores.

One of the most surprising results from our evaluation is the significant gap between GA and the other metrics in terms of accuracy scores for the same template identification techniques; for example, on average for all datasets, Drain achieved a GA score of 87%, but its PA and PTA‡RTA scores were only 29%, 27%, and 29%, respectively. In other words, by taking into account the correctness of fixed and variable parts of identified templates, we get a very different picture of the accuracy of the techniques in comparison to the existing study [ZHL$^+$19] relying on the GA metric. The results emphasize the importance of using adequate metrics to assess template identification results in the context of target use cases. For example, if the parameter values of log messages are used by a log-based analysis task, e.g., model inference with guard conditions [WTD16] and advanced anomaly detection using parameter values [DLZS17], accuracy should be assessed using PA or TA scores since they adequately measure a technique's ability to correctly identify the variable parts of log messages. However, one might argue that PA and TA metrics are too strict to fairly indicate the usefulness of template identification

[3]https://github.com/logpai/logparser (accessed: 2021-07-21)

[4]https://github.com/BlueLionLogram/Logram (accessed: 2021-07-21)

techniques as no degree of correctness is accounted for individual templates, even for the cases where the variable parts of log messages matter. More studies are therefore required to better understand the relationship between various accuracy scores of template identification techniques and their actual usefulness in specific applications.

For some of the techniques (e.g., Drain), the importance of oracle template corrections is confirmed by the evaluation results. However, overall, the impact of such corrections on accuracy scores is limited. Based on our analysis, we reckon that the more accurate the technique, the more likely its ranking is to be significantly affected by oracle template corrections. Since all techniques, to various degrees, fare relatively poorly, as discussed above, this result in limited impact on accuracy. In the future, as template identification techniques are expected to improve, oracle template corrections will increasingly become important.

We also show that the analysis of incorrectly identified templates can provide insights on how to improve individual techniques by understanding their limitations in terms of the over- and under-generalization of templates. Nevertheless, since decreasing the over-generalization may increase under-generalization and vice versa, how to practically improve the accuracy of the techniques remains an open problem for future work.

3.4.6 Threats to Validity

Using a specific set of log datasets is a potential threat to external validity. However, the benchmark contains 16 log datasets from real-world systems and the corresponding oracle templates, and has been used in previous studies focused on log template identification techniques [ZHL+19, DLC+20]. Nevertheless, further experiments with different benchmarks are required to improve the generalizability of our results.

The implementations of the template identification techniques used in the evaluation may introduce threats to internal validity. To mitigate such threats, we used the same set of implementations used in the previous studies (i.e., [ZHL+19, DLC+20]). Furthermore, as mentioned above, we checked that the GA scores obtained by our experiments are exactly the same as those computed by the replication package of Zhu et al. [ZHL+19].

The heuristic rules for oracle template correction can also be potential threats to internal validity. Ideally, correct oracle templates extracted from the source code are mandatory to validate the heuristic rules. At the same time, such extraction task is very challenging, even for open-source programs. This is because different versions of the same program may have different log printing statements while the logs provided in the benchmark [ZHL+19] do not have the corresponding program version information. Therefore, for each program, one should manually check many previous versions to find log printing statements that would generate the given log messages; moreover, older versions may no longer be available. Though this process is extremely time-consuming, we tried our best to find the matching versions for the open-source program logs. We could find them only for HDFS, Hadoop, and OpenStack logs, and confirmed that the oracle templates modified by our heuristic rules are indeed correct with respect to the logging statements in the source code.

3.4.7 Data Availability

The replication package of our empirical evaluation — including the Python implementations for computing TA metrics and applying oracle template correction and incorrect template analysis — is available on Figshare [KSBB22a].

3.5 Summary

In this chapter, we provide three guidelines for evaluating the accuracy of log message template identification techniques. We also assess the application of such guidelines through a comprehensive evaluation of 14 log template identification techniques using a benchmark composed of 16 real-world log datasets, previously used in the literature. The evaluation results show that the accuracy assessment results for different techniques may significantly vary depending on whether the guidelines are followed.

Chapter 4

A Theoretical Framework for Understanding the Relationship between Log Parsing and Anomaly Detection

4.1 Overview

Despite advances in log parsing and log-based anomaly detection, to the best of our knowledge, no existing work has thoroughly investigated the impact of the "quality" of log parsing results on anomaly detection. In particular, the concept of "ideal" log parsing results with respect to anomaly detection has not been defined and formalized yet. The lack of such a conceptual framework makes it difficult, when performing empirical studies on anomaly detection techniques, to determine if the root causes of inaccuracies in anomaly detection are due to the limitations of log parsing techniques.

In this chapter, we propose a theoretical framework for defining and discussing what are ideal log parsing results for anomaly detection. In particular, we consider log parsing as an information abstraction process that converts collected logs into structured logs. Since automated anomaly detection relies on structured logs, its best operating conditions are when the minimum amount of information that is necessary to distinguish normal from abnormal behaviors is present in such logs. To this end, we formally define the concepts of *distinguishability* and *minimality*, which further lead to the definition of *ideal* log parsing results. We also discuss practical implications related to log parsing and anomaly detection and present future research directions derived from our theoretical framework.

The rest of the chapter is organized as follows. Section 4.2 explains the notations and basic definitions that will be used throughout the chapter. Section 4.3 formalizes the key concepts regarding ideal log parsing results for anomaly detection. Section 4.4 discusses practical implications of the proposed theoretical framework. Section 4.5 discusses related work. Section 4.6 concludes the chapter.

4.2 Preliminaries

This section introduces basic notations and concepts that will be used throughout the chapter.

We use uppercase letters to denote *collections* (i.e., sets and sequences) and lowercase letters to denote *elements* in a collection. Specifically, $\{\dots\}$ denotes a set and $\langle\dots\rangle$ denotes a sequence. For simplicity, we use the same notation $|S|$ to denote the *cardinality* of a set S and the *length* of a sequence S.

Definition 1 (Log Messages and Logs)**.** *A* log message *m is a string printed by a logging statement in the source code. A* log *l is a finite sequence of log messages, denoted by $l = \langle m_1, m_2, \dots, m_n \rangle$.*

For instance, Figure 4.1 shows a simplified[1] example from the actual log produced by running HDFS [HZHL20]. In this case, the example log can be denoted by $l_{ex} = \langle m_1, m_2, \dots, m_{14} \rangle$ where m_{14} is the string "`Verification succeeded for blk_471078`".

Definition 2 (Normal and Abnormal Logs)**.** *For a set of logs L, a log $l \in L$ is said to be* normal *if and only if it represents a normal behavior of the system. Otherwise, l is said to be* abnormal*. Normal and abnormal logs are denoted by $L_n \subseteq L$ and $L_a \subseteq L$, respectively; for a given L and the corresponding L_n and L_a, we have $L_n \cap L_a = \emptyset$ and $L_n \cup L_a = L$.*

Notice that Definition 2 is independent from the nature of anomalies (e.g., point and collective [CBK09]). The definition only assumes that normal and

[1]In general, logs may contain extra information, such as timestamps and logging levels (e.g., info, debug) for individual log messages. However, we omit such information since log parsing deals with log messages characterizing the states or events of the system.

Index	Log Message
1	`Receiving block blk_471078 src: /1.2.3.4:56 dest: /1.2.3.4:78`
2	`Receiving block blk_471078 src: /4.3.2.1:65 dest: /4.3.2.1:78`
...	...
14	`Verification succeeded for blk_471078`

Figure 4.1: Example from HDFS Logs [HZHL20]

abnormal behaviors of the system are known and distinguishable in logs. This assumption can be easily satisfied when the accuracy of anomaly detection techniques, including log parsing techniques, are evaluated in controlled experiments using known benchmarks; furthermore, enhancing the quality of logs to distinguish normal and abnormal behaviors has been actively studied [LXL+21, YPH+12, YZP+12, ZRL+17]. In the rest of the chapter we adopt this definition and make its underlying assumption.

4.3 Ideal Log Parsing Results for Anomaly Detection

4.3.1 Log Parsing as Abstraction

Intuitively, log parsing is a process that converts original logs composed of free-formed messages into structured logs, by extracting key information from individual log messages. For example, some log parsing approaches, such as Drain [HZZL17] and MoLFI [MPB+18], may extract key information from m_1 in Figure 4.1 as an event template "`Receiving block <*> src: <*> dest <*>`" characterizing the event of receiving a block, where symbol `<*>` indicates the position of a parameter value determined at runtime. With respect to these approaches, all messages that match the template, such as m_2 in Figure 4.1, represent the same event of receiving a block. In this sense, we can consider log parsing as an *abstraction* process that generates "abstract" key information that represents multiple "specific" messages. Notice that different log parsing approaches yield different abstraction results (not even necessarily in the form of templates). For example, a log parsing approach that simply counts the number of tokens would abstract m_1 as the integer 7 (as m_1 contains seven tokens). To keep our presentation general, we introduce an abstraction function τ that represents a log parsing approach.

Definition 3 (Log Parsing as an Abstraction Function). *Given a set of log messages M and a generic set of parsing results A, a log parsing approach can be represented as an abstraction function $\tau\colon M \to A$.*

Notice that the definition of A is left generic, to accommodate different types of results yielded by log parsing techniques.

Using the concept of τ, the results obtained by a parsing approach can be seen as an abstraction of the original log itself.

Definition 4 (Abstraction of Log)**.** *Given an abstraction function τ representing a log parsing approach and a log $l = \langle m_1, m_2, \ldots, m_n \rangle$, the abstraction of l using τ, denoted by $\tau^*(l)$, is defined as $\tau^*(l) = \langle \tau(m_1), \tau(m_2), \ldots, \tau(m_n) \rangle$.*

In other words, τ^* can be considered as an abstraction function for a log, extended from τ. Similarly, we can further extend τ^* to consider a set of logs as follows.

Definition 5 (Abstraction of Set of Logs)**.** *Given an abstraction function τ representing a log parsing approach and a set of logs L, the abstraction of L using τ, denoted by $\tau^{**}(L)$, is defined as $\tau^{**}(L) = \{\tau^*(l) \mid l \in L\}$.*

Based on these definitions, $\tau^{**}(L)$ represents the results of using a log parsing approach (abstracted by τ) on a set of logs L; in our context, it represents the structured input logs provided as input to an anomaly detection approach.

4.3.1.1 Running Example

To better understand the above definitions, let us consider a set of logs $L_{ex} = \{l_1, l_2, l_3\}$ where $l_1 = \langle m_a, m_b, m_c \rangle$, $l_2 = \langle m_b, m_a, m_c \rangle$, and $l_3 = \langle m_a, m_b, m_d \rangle$ and each message in $\{m_a, m_b, m_c, m_d\}$ is different from the others. Let us assume to use a log parsing approach that yields an integer value, such that both m_a and m_b are mapped to 1 while m_c and m_d are mapped to 2 and 3, respectively. We can represent the log parsing approach as the abstraction function τ_{ex} defined such that $\tau_{ex}(m_a) = \tau_{ex}(m_b) = 1$, $\tau_{ex}(m_c) = 2$, and $\tau_{ex}(m_d) = 3$. Using τ_{ex}, we can see that the abstraction of l_1 is $\tau^*_{ex}(l_1) = \langle \tau_{ex}(m_a), \tau_{ex}(m_b), \tau_{ex}(m_c) \rangle = \langle 1, 1, 2 \rangle$. Similarly, $\tau^*_{ex}(l_2) = \langle 1, 1, 2 \rangle$ and $\tau^*_{ex}(l_3) = \langle 1, 1, 3 \rangle$. As a result, the abstraction of L_{ex} is $\tau^{**}_{ex}(L_{ex}) = \{\tau^*_{ex}(l_1), \tau^*_{ex}(l_2), \tau^*_{ex}(l_3)\} = \{\langle 1, 1, 2 \rangle, \langle 1, 1, 3 \rangle\}$.

This example shows that different logs (e.g., l_1 and l_2) can be *indistinguishable* when abstracted using a certain log parsing approach. This is directly related to one of the key concepts for defining the ideal log parsing results for anomaly detection, which will be detailed in the next section.

4.3.2 Ideal Log Parsing Results

As described above, log parsing abstracts L to $\tau^{**}(L)$, possibly resulting in different logs indistinguishable from each other as a result of abstraction. For the main anomaly detection step that takes $\tau^{**}(L)$ as input, if normal and abnormal logs are indistinguishable in $\tau^{**}(L)$, then it is impossible to correctly identify abnormal behaviors from it. It is thus important to formalize the concept of *distinguishability* of log parsing results:

Definition 6 (Distinguishability of Log Parsing Results)**.** *Given a non-empty set of normal logs $L_n \subset L$ and a non-empty set of abnormal logs $L_a \subset L$ (where $L_n \cup L_a = L$ and $L_n \cap L_a = \emptyset$), an abstraction function τ* distinguishes *L_n and L_a if and only if $\tau^{**}(L_n) \cap \tau^{**}(L_a) = \emptyset$. In this case, $\tau^{**}(L)$ is called* d-maintaining *(maintaining the distinguishability) between L_n and L_a.*

In other words, *d-maintaining* log parsing results maintain the distinction between L_n and L_a after the abstraction of log parsing. For our running example used in Section 4.3.1, let us additionally consider $L_n = \{l_1, l_2\}$ and $L_a = \{l_3\}$. Since $\tau_{ex}^{**}(L_n) = \{\langle 1, 1, 2\rangle\}$ and $\tau_{ex}^{**}(L_a) = \{\langle 1, 1, 3\rangle\}$, $\tau_{ex}^{**}(L_n) \cap \tau_{ex}^{**}(L_a) = \emptyset$, and therefore $\tau_{ex}^{**}(L_{ex})$ is *d-maintaining* between L_n and L_a.

However, distinguishability is only a necessary condition for log parsing results to be ideal. For example, if we consider an abstraction function $\tau_{=}$ such that $\tau_{=}(m) = m$ for every message m, $\tau_{=}^{**}(L)$ is *d-maintaining* between arbitrary L_n and L_a (since $\tau_{=}^{**}(L_n) = L_n$, $\tau_{=}^{**}(L_a) = L_a$, and $L_n \cap L_a = \emptyset$ by definition). However, $\tau_{=}^{**}(L)$ does not represent the ideal log parsing results because $\tau_{=}$ does not produce an actual abstraction (since it is just defined as the identity function). Indeed, as long as log parsing results maintain the distinguishability between L_n and L_a, a higher degree of abstraction (i.e., mapping more messages to the same parsing result) leads to better operating conditions for anomaly detection, as it minimizes the "information" contained in the structured input logs (i.e., the log parsing results) that must be analyzed by the main anomaly detection step. Furthermore, since anomaly detection is largely based on Machine Learning (ML) [HHC+21], including Clustering, Support Vector Machine (SVM), and Long Short-Term Memory (LSTM), the abstraction of log parsing can significantly improve the learning performance of anomaly detection by reducing dimensionality (i.e., the number of features)[2]. Therefore, we should additionally consider the concept of *minimality* of the information contained in log parsing results.

[2]This is because distinct $\tau(m)$ for each message m that appear in L can lead to one or more dimensions. In ML, dimensionality reduction is an essential topic to improve predictive power [MRT12].

To formalize the minimality concept, we first need to define the information contained in log parsing results. Since the core of log parsing is to abstract individual messages, we consider the information contained in $\tau^{**}(L)$ in terms of its unique entities (i.e., abstracted messages) as follows.

Definition 7 (Information in Log Parsing Results). *Given a set of logs L and an abstraction function τ, the information contained in L as abstracted by τ, denoted by $I(L, \tau)$, is defined as $I(L, \tau) = \bigcup_{l \in L}\{\tau(m) \mid m \in l\}$.*

For our running example, the information contained in L_{ex} abstracted by τ_{ex} is $I(L_{ex}, \tau_{ex}) = \{\tau_{ex}(m_a), \tau_{ex}(m_b), \tau_{ex}(m_c), \tau_{ex}(m_d)\} = \{1, 2, 3\}$, meaning that τ_{ex} reduces the information from $\{m_a, m_b, m_c, m_d\}$ to $\{1, 2, 3\}$ through abstraction.

Using $I(L, \tau)$, we can define the concept of ideal log parsing results by considering both distinguishability and minimality as follows.

Definition 8 (Minimal Distinguishable Log Parsing Results). *Given a non-empty set of normal logs $L_n \subset L$ and a non-empty set of abnormal logs $L_a \subset L$ (where $L_n \cup L_a = L$ and $L_n \cap L_a = \emptyset$), we say that $\tau^{**}(L)$ is* minimally d-maintaining *between L_n and L_a if and only if (1) $\tau^{**}(L)$ is* d-maintaining *between L_n and L_a and (2) there is no abstraction function τ' such that $\tau'^{**}(L)$ is* d-maintaining *and $|I(L, \tau')| < |I(L, \tau)|$.*

Taking the running example again, $\tau^{**}_{ex}(L_{ex})$ is *d-maintaining* (but not minimally) because there exists τ_{new} such that (1) $\tau^{**}_{new}(L_{ex})$ is *d-maintaining* and (2) $|I(L_{ex}, \tau_{new})| < |I(L_{ex}, \tau_{ex})|$. Specifically, if $\tau_{new}(m_a) = \tau_{new}(m_b) = \tau_{new}(m_c) = 12$ and $\tau_{new}(m_d) = 3$, then $\tau^{**}_{new}(L_n) = \{\langle 12, 12, 12\rangle\}$, $\tau^{**}_{new}(L_a) = \{\langle 12, 12, 3\rangle\}$, $I(L_{ex}, \tau_{new}) = \{12, 3\}$; therefore $\tau^{**}_{new}(L_n) \cap \tau^{**}_{new}(L_a) = \emptyset$ and $|I(L_{ex}, \tau_{new})| = |\{12, 3\}| = 2$ is less than $|I(L_{ex}, \tau_{ex})| = |\{1, 2, 3\}| = 3$. However, $\tau^{**}_{new}(L_{ex})$ is *minimally d-maintaining* because there is no τ' such that $|I(L, \tau')| = 1$ and $\tau'^{**}(L)$ is *d-maintaining*. As a result, $\tau^{**}_{new}(L_{ex})$ represents the ideal log parsing results for anomaly detection.

4.4 Applications

4.4.1 Localization of the Causes of Inaccurate Anomaly Detection

When anomaly detection accuracy is not 100% (i.e., some abnormal behaviors are not correctly detected or some normal behaviors are incorrectly detected as abnormal), it is important to know exactly where the problem lies (in the log parsing step, in the main anomaly detection step, or in both), in order to improve the results. Using our theoretical framework, we can localize the cause of inaccurate anomaly detection results. Specifically, for a set of normal logs L_n and a set of abnormal logs L_a, we can distinguish the following three cases.

Case 1. If the log parsing results is minimally d-maintaining between L_n and L_a, the main anomaly detection step must be the cause of the inaccuracy, because the log parsing results are ideal for anomaly detection.

Case 2. If the log parsing results is d-maintaining between L_n and L_a but not minimally so, a perfect anomaly detection approach could achieve pinpoint accuracy. However, as discussed in Section 4.3.2, making the log parsing results minimally d-maintaining could significantly increase anomaly detection accuracy.

Case 3. Otherwise, inaccurate anomaly detection results are inevitable due to the low-quality log parsing results. We can further investigate the issue of log parsing results by focusing on exactly what prevents the log parsing results from being d-maintaining between L_n and L_a.

The above characterization has important implications for researchers who want to assess the accuracy of their anomaly detection approaches. As non-ideal log parsing results decrease anomaly detection accuracy, *it is recommended to use ideal log parsing results in controlled experiments* to properly assess the performance of a technique, independently of log parsing. Also, if possible, using various log parsing results including minimally d-maintaining, d-maintaining but not minimal, and non-d-maintaining ones, would provide a better picture on how anomaly detection would work in practice, depending on the quality of the log parsing results.

4.4.2 Removal of Unnecessary Log Messages for Anomaly Detection

As discussed in Section 4.3.1, some messages become indistinguishable through the abstraction of log parsing. One may wonder whether simply removing some of the messages could contribute to further reduce the amount of information contained in the log parsing results. Indeed, in our running example, $\tau_y^{**}(L_{ex})$ remains minimally d-maintaining even if we remove m_a and m_b from L_{ex}. However, this is not always true. For example, consider a normal log $l_n = \langle m_x, m_y \rangle$ and an abnormal log $l_a = \langle m_y \rangle$. While we can consider an abstraction function τ such that $\tau(m_x) = \tau(m_y)$ and $\tau^*(l_n) \neq \tau^*(l_a)$, removing m_x from the logs makes l_n and l_a indistinguishable. This example shows that, though there are messages that can be abstracted to the same entity, it does not necessarily mean that one of them can be removed without affecting anomaly detection accuracy.

Notice that existing log parsing techniques do not reduce the length of individual logs[3]. However, we know, as discussed above, that having minimal information necessary to distinguish normal and abnormal logs is the best operating condition for anomaly detection. In this sense, further research is needed to develop an automated approach for "greedy" log parsing techniques that not only abstract but also remove log messages to achieve minimality while maintaining the distinguishability of the results.

4.5 Related Work

To the best of our knowledge, there is no existing work that provides a framework to formalize the concept of ideal log parsing results for anomaly detection. This is mainly because most of the existing log parsing approaches, including AEL [JHFH08], Drain [HZZL17], IPLoM [MZHM09], LenMa [Shi16], LFA [NV10], LogCluster [VP15], LogMine [HDX+16], LogSig [TLP11], MoLFI [MPB+18], SHISO [Miz13], SLCT [Vaa03], Spell [DL16], and Logram [DLC+20], have been proposed as general-purpose approaches rather than specialized for anomaly detection. The accuracy of all these approaches has been assessed with respect to the logging statements that produce individual messages. For example, the execution of the logging statement `printf("retry " + i)` in the source code, when the program variable `i` evaluates to 1, will generate the log message "`retry 1`". Then a log parsing approach is expected to reconstruct the form of the logging statement as a template "`retry <*>`" without accessing the source code, where symbol "`<*>`" indicates the position of the parameter value (i.e., "`1`"). In other words, the ground truth used to assess the accuracy of general-purpose log parsing is determined based on the logging statements that generated the input logs. On the other hand, there is no ground truth that guarantees the best operating conditions for anomaly detection. To address this challenge, we provide a theoretical foundation to precisely define key concepts, including the distinguishability and minimality of ideal log parsing results.

[3]Though the length of logs can be reduced in a pre-processing step by omitting certain messages or events based on domain knowledge, this is independent from log parsing, which just abstracts messages.

4.6 Summary

In this chapter, we proposed a theoretical framework that formalizes the concepts of distinguishability and minimality, showing that log parsing results that minimally maintain distinguishability between normal and abnormal logs provide the best operating conditions for anomaly detection. Using our theoretical framework, we also identified practical implications for researchers regarding the root causes for inaccuracy in anomaly detection and the removal of log messages that are unnecessary for anomaly detection.

Chapter 5

Impact of Log Parsing on Deep Learning-Based Anomaly Detection

5.1 Overview

The frequent combination of log parsing and anomaly detection clearly implies the importance of the former for the latter. Nevertheless, assessing in a systematic way the impact of log parsing on anomaly detection has received surprisingly little attention so far. In chapter 4, we investigated what *ideal* log parsing results are in terms of accurate anomaly detection, but purely from a theoretical standpoint. Le and Zhang [LZ22] empirically showed that different log parsing techniques, among other potential factors, can significantly affect anomaly detection accuracy, but the accuracy of log parsing results was not adequately measured, and the correlation between log parsing accuracy and anomaly detection accuracy was not reported. Although Fu et al. [FYX$^+$23] attempted to address the issue by measuring log parsing and anomaly detection accuracy, they used only one log parsing accuracy metric among those proposed in chapter 3; moreover, the log parsing results used to measure the accuracy of anomaly detection techniques represented only a small subset (less than 1%) of all logs used by the techniques considered in the study, thus limiting the validity of the results.

To systematically investigate the impact of log parsing on anomaly detection while addressing the issues of the aforementioned studies, this chapter reports on an empirical study, in which we performed a comprehensive evaluation using 13 log parsing techniques and five deep learning-based anomaly detection techniques on two publicly available log datasets. We considered all three log parsing accuracy metrics (i.e., grouping accuracy [ZHL+19], parsing accuracy [DLC+20], and template accuracy proposed in chapter 3.

Against all assumptions, our results show that there is no strong correlation between log parsing accuracy and anomaly detection accuracy, regardless of the metric used for measuring log parsing accuracy. In other words, accurate log parsing results do not necessarily increase anomaly detection accuracy. To better understand the phenomenon at play, we investigated another property of log parsing, *distinguishability*, a concept proposed in chapter 4, that was theoretically shown to relate to anomaly detection accuracy. Our empirical results confirm that, as far as anomaly detection is concerned, distinguishability in log parsing results is the property that really matters and should be the key target of log parsing.

In summary, the main contributions of this chapter are:

- the systematic and comprehensive evaluation of the impact of log parsing on anomaly detection;
- the investigation of the impact of the distinguishability of log parsing results on anomaly detection.

The rest of the chapter is organized as follows. Section 5.2 motivates our study and introduces the research questions. Section 5.3 describes the experimental design, including the log datasets, log parsing techniques, and anomaly detection techniques used in the experiments. Section 5.4 presents the experimental results. Section 5.5 discusses the practical implications, derived from the results, for the application of log parsing in the context of anomaly detection. Section 5.7 concludes the chapter.

5.2 Motivation

Log parsing converts unstructured logs into structured ones, which can then be processed by log-based analysis techniques like anomaly detection. It is quite natural to speculate that log parsing results can affect anomaly detection results. Intuitively, the research literature has assumed that inaccurate log parsing results leads to inaccurate anomaly detection results. However, this hypothesis has not been fully investigated in the literature, except for one empirical study [LZ22] and one analytical investigation, i.e., chapter 4.

Le and Zhang [LZ22] recently presented an empirical work investigating several aspects that can impact Deep Learning (DL)-based anomaly detection approaches, such as training data selection, data grouping, class distribution, data noise, and early detection ability. One of their experiments considering data noise assessed the impact of noise deriving from log parsing results. Specifically, they used four log parsing techniques (Drain [HZZL17], Spell [DL16], AEL [JHFH08], and IPLoM [MZHM09]) to generate log parsing results for two log datasets (BGL [OS07] and Spirit [OS07]). Then, for each log dataset, they used the different log parsing results as input of five anomaly detection approaches (DeepLog [DLZS17], LogAnomaly [MLZ$^+$19], PLELog [YCW$^+$21], LogRobust [ZXL$^+$19], and CNN [LWLW18]), and measured the accuracy of the latter. Their experimental results showed that log parsing approaches highly influence the accuracy of anomaly detection; for example, the F1-Score of DeepLog on Spirit logs [OS07] decreases from 0.755 to 0.609 when Drain is used instead of IPLoM for log parsing.

Although this is the first clear evidence showing the impact of log parsing results on anomaly detection accuracy, the scope of the underlying study is limited. For example, it simply uses different log parsing results (produced by different tools) without quantitatively assessing the accuracy of the log parsing tools; therefore, the relationship between log parsing accuracy and anomaly detection accuracy remains unclear. To this end, we define our first research question as follows: ***RQ1 - To which extent does the accuracy of log parsing affect the accuracy of anomaly detection?***

In chapter 4, we proposed a theoretical framework determining the ideal log parsing results for anomaly detection by introducing the concept of "distinguishability" for log parsing results. It is argued that, rather than accuracy as previously assumed, what really matters is the extent to which log parsing results are distinguishable. However, to the best of our knowledge, there is no empirical work assessing quantitatively distinguishability in log parsing results and its impact on anomaly detection accuracy. Therefore, we define our second research question as follows: ***RQ2 - How does the accuracy of anomaly detection vary with the distinguishability of log parsing results?***

Answering the above questions will have a significant impact on both research and industry in the field of log-based anomaly detection. For example, if the answer to the first question is that, regardless of the log parsing accuracy metrics, there is no relationship between log parsing accuracy and anomaly detection accuracy, then it means that there is no need to use the existing accuracy metrics to evaluate log parsing results for anomaly detection. This would completely change the way log parsing tools are evaluated. Similarly, if the answer to the second question is that the distinguishability of log parsing results indeed affects anomaly detection, as expected from chapter 4, then this must be the focus of log parsing evaluations. As a result, our answers will provide essential insights on better assessing the quality of log parsing techniques for more accurate anomaly detection.

5.3 Experimental Design

All experiments presented in this chapter were carried out using the HPC facilities of the University of Luxembourg (see https://hpc.uni.lu). Specifically, we used Dual Intel Xeon Skylake CPU (8 cores) and 64GB RAM for running individual log parsing and anomaly detection techniques.

5.3.1 Datasets

To answer the research questions introduced in Section 5.2, we used publicly available datasets based on the LogHub benchmark [HZHL20], which contains a large collection of log messages from various types of systems including operating systems (Linux, Windows, and Mac), distributed systems (BGL, Hadoop, HDFS, Thunderbird, and OpenStack), standalone programs (Proxifier and Zookeeper), and mobile systems (Android). The benchmark has been widely used in various studies focused on log parsing [KSBB22b, ZHL+19, DLC+20] and anomaly detection [LZ22, FYX+23].

Among the benchmark datasets, we selected HDFS, Hadoop, and OpenStack datasets because of the following reasons: (1) they have labels for normal and abnormal logs to be used for assessing the accuracy of anomaly detection techniques *and* (2) the source code of the exact program version used to generate the logs is publicly available; this allows us to extract correct oracle templates (i.e., ground truth templates) for each log message. The oracle templates are especially important in our study as we need to carefully assess both log parsing accuracy and anomaly detection accuracy. Although the benchmark provides some oracle templates for all log datasets, they are *manually generated* (without accessing the source code) and cover *only* 2K log messages randomly sampled for each dataset. As discussed in chapter 3, those manually generated oracle templates are *error-prone*; therefore, we used the logging statements in the source code to extract correct oracle templates. Table 5.1 shows all the log datasets in the LogHub benchmark and whether they meet each of the above-mentioned criteria; the rows highlighted in gray meet both criteria.

Table 5.1: Datasets in LogHub benchmark [HZHL20]

Datasets	Anomaly Label	Source Code
Android	✗	✗
Apache	✗	✗
BGL	✓	✗
HDFS	✓	✓
HPC	✗	✗
Hadoop	✓	✓
HealthApp	✗	✗
Linux	✗	✗
Mac	✗	✗
OpenSSH	✗	✗
OpenStack	✓	✓
Proxifier	✗	✗
Spark	✗	✗
Spirit	✓	✗
Thunderbird	✓	✗
Windows	✗	✗
Zookeeper	✗	✗

During our preliminary evaluation, we found an issue with HDFS. The original HDFS logs were too large to be processed by the slowest anomaly detection technique (i.e., LogAnomaly [MLZ$^+$19]) when setting a two-day timeout. Due to the large number of experiments we needed to conduct (i.e., all combinations of log parsing and anomaly detection techniques with additional repeats for distinguishable and indistinguishable log parsing results, see § 5.3.4 and § 5.3.5), we decided to reduce the log dataset size. As we found that the slowest log parsing technique (i.e., LogAnomaly) could process up to $n = 300K$ messages within 2 hours, we randomly and iteratively removed logs (i.e., sequences of log messages) from the HDFS dataset to reduce it until the total number of remaining messages was less than 300K. Notice that each HDFS log is a sequence of log messages having the same block ID, representing either a normal or abnormal sequence of events. To preserve individual (normal or abnormal) sequences, we randomly selected and removed them by sequence, not by message.

Table 5.2 reports on the size of our datasets, in terms of the number of oracle templates (O), the number of all logs (L_{all}), the number of normal logs (L_n), the number of abnormal logs (L_a), the number of all messages (M_{all}), the number of messages in normal logs (M_n), and the number of messages in abnormal logs (M_a). Note that the number of log messages is the same as the

Table 5.2: Size information of the log datasets used in our experiments. Number of oracle templates (O); Number of all logs (L_{all}); Number of normal logs (L_n); Number of abnormal logs (L_a); Number of all messages (M_{all}); Number of messages in normal logs (M_n); Number of messages in abnormal logs (M_a).

Dataset	O	L_{all}	L_n	L_a	M_{all}	M_n	M_a
HDFS (reduced)	26	15295	15026	269	299971	292776	7195
Hadoop	175	54	11	43	109968	14392	95576
OpenStack	21	2068	2064	4	79925	79817	108

number of log entries (see Chapter 4)

5.3.2 Log Parsing Techniques

We aimed to use as many log parsing techniques as possible, among those available in the literature. Since, in chapter 3 we provided a comprehensive evaluation of 14 log parsing techniques (i.e., AEL [JHFH08], Drain [HZZL17], IPLoM [MZHM09], LenMa [Shi16], LFA [NV10], LKE [FLWL09], LogCluster [VP15], LogMine [HDX+16], Logram [DLC+20], LogSig [TLP11], MoLFI [MPB+18], SHISO [Miz13], SLCT [Vaa03], and Spell [DL16]), we decided to reuse the replication package, including all the aforementioned techniques.

However, we had to exclude LKE since our preliminary evaluation results showed that it could not complete its run for *all* of our log datasets within the 2-day timeout. Notice that we have already reduced our log datasets (in particular, HDFS), as discussed in Section 5.3.1, based on the slowest anomaly detection technique (i.e., LogAnomaly). Although we could additionally reduce the datasets based on the slowest log parsing technique (i.e., LKE), we found that it would result in small logs that are not representative of the size and complexity of real-world logs.

As a result, we considered 13 log parsing techniques in our experiments. For all the log parsing techniques, we used their default parameters.

5.3.3 Anomaly Detection Techniques

Similar to the case of log parsing techniques, we considered the work of Le and Zhang [LZ22], a recent empirical study that evaluated five DL-based anomaly detection techniques (i.e., DeepLog [DLZS17], LogAnomaly [MLZ+19], LogRobust [ZXL+19], PLELog [YCW+21], and CNN [LWLW18]), and decided to use their replication package, including all the aforementioned techniques. For all anomaly detection techniques, we used their default parameters. These techniques are representative of the state of the art of DL-based anomaly detection techniques.

We want to note that we did not consider anomaly detection techniques based on traditional machine learning models (e.g., PCA, Logistic Regression, and Decision Tree) since Fu et al. [FYX$^+$23] recently showed that they are significantly less effective than DL-based techniques.

5.3.4 Methodology for RQ1

To answer RQ1, for each dataset, we first executed the log parsing techniques to generate log parsing results and computed their accuracy in terms of GA, PA, and FTA (see Chapter 2). We then executed the anomaly detection techniques on each of the log parsing results and computed their accuracy in terms of precision (PR), recall (RE), and F1 score. By doing so, we obtained a tuple of accuracy values $\langle GA, PA, FTA, PR, RE, F1 \rangle$ for each combination of datasets, log parsing results, and anomaly detection techniques.

For log parsing, we executed each of the log parsing techniques with a 2-day timeout. Since MoLFI is non-deterministic, we executed it three times. In total, we obtained 16 log parsing results (three from the three different executions of MoLFI and 13 from the remaining log parsing techniques) for each dataset. For each log parsing result, we computed $\langle GA, PA, FTA \rangle$ using the oracle templates (and the messages matching them) for the corresponding datasets.

For anomaly detection, we randomly divided the individual log parsing results into two disjoint sets, i.e., a training set and a test set, using a split ratio of 80:20. We trained the anomaly detection techniques on each of the training sets with a 2-day timeout, and used the corresponding test sets to compute $\langle PR, RE, F1 \rangle$. To account for the randomness of anomaly detection techniques, we repeated the train-and-test process five times and used the average F1 score.

As a result, we obtained 160 tuples $\langle GA, PA, FTA, PR, RE, F1 \rangle$ from the combinations of two datasets, 16 log parsing results, and five anomaly detection techniques.

5.3.5 Methodology for RQ2

To answer RQ2, we need distinguishable and indistinguishable log parsing results to compare in terms of anomaly detection accuracy. Although the log parsing results generated for RQ1 are available, they are mostly distinguishable, leading to unbalanced data for RQ2. To systematically assess the impact of the distinguishability of log parsing results on anomaly detection accuracy using balanced data, we generate pairs of distinguishable and indistinguishable log parsing results.

Specifically, let $d(R)$ be the distinguishability — expressed as a Boolean value, either *true* (T) or *false* (F) — of a log parsing result R. For each log parsing result R generated in the context of RQ1 (i.e., 16 log parsing results for each of the two datasets), we first created a pair of log parsing results $\langle R, R' \rangle$ by artificially generating R' from R such that $d(R') = \neg d(R)$ using Algorithms 1 and 2, detailed further below. By definition, if R is distinguishable then R' will be indistinguishable and vice versa. For the sake of simplicity, we denote the distinguishable result (be it R or R') as R_{dst} and the indistinguishable one (respectively, either R' or R) as R_{ind}. We then executed, for all pairs $\langle R_{dst}, R_{ind} \rangle$, all the considered anomaly detection techniques twice: the first time using R_{dst} as input and the second time using R_{ind} as input; for each run of each anomaly detection technique we computed its accuracy in terms of precision, recall, and F1 score. By doing so, we obtained the anomaly detection accuracy scores for pairs of distinguishable (R_{dst}) and indistinguishable (R_{ind}) versions of log parsing results, and then compared them.

For the generation of R' from R, it is important to minimize the difference between R and R' while achieving $d(R') = \neg d(R)$. This is to ensure that if there is a difference in anomaly detection scores between R and R', it is mostly due to distinguishability and not to other differences between R and R' (e.g., the number of templates or the size of log parsing results). To do this, we need to distinguish the two cases when $d(R) = T$ and when $d(R) = F$, as described below.

5.3.5.1 Generation of Indistinguishable from Distinguishable Log Parsing Results

When $d(R) = T$ (i.e., $R = R_{dst}$), it means that templates for different log messages in R are different enough to distinguish between normal and abnormal logs in R, as explained in Chapter 4. For example, let us consider two logs $l_1 = \langle m_1, m_2 \rangle$ and $l_2 = \langle m_3, m_4 \rangle$ where the templates of the four messages are identified as $\tau(m_1) = t_1$, $\tau(m_2) = t_2$, $\tau(m_3) = t_3$, and $\tau(m_4) = t_2$, respectively, using a log parsing technique τ. Figure 5.1 shows the logs, messages, and templates. In this case, the log parsing result of τ for $\{l_1, l_2\}$ is *distinguishable*, as highlighted in blue in the figure, since $\tau^*(l_1) = \langle \tau(m_1), \tau(m_2) \rangle = \langle t_1, t_2 \rangle$ and $\tau^*(l_2) = \langle \tau(m_3), \tau(m_4) \rangle = \langle t_3, t_2 \rangle$ are different (due to $\tau(m_1) \neq \tau(m_3)$, i.e., $t_1 \neq t_3$). However, if the templates of m_1 and m_3 were the same, then the log parsing result would be *indistinguishable*. In other words, as highlighted in red in the figure, we can make the distinguishable log parsing re-

Log	Template	Parsed Log (original)	Parsed Log (after merging t_1 and t_3 into t_{13})
$l1 = \langle m_1, m_2 \rangle$	$\tau(m_1) = t_1$, $\tau(m_2) = t_2$	$\tau^*(l_1) = \langle t_1, t_2 \rangle$	$\tau'^*(l_1) = \langle t_{13}, t_2 \rangle$
$l2 = \langle m_3, m_4 \rangle$	$\tau(m_3) = t_3$, $\tau(m_4) = t_2$	$\tau^*(l_2) = \langle t_3, t_2 \rangle$	$\tau'^*(l_2) = \langle t_{13}, t_2 \rangle$

Figure 5.1: An example of making a distinguishable log parsing result indistinguishable by merging templates

sult of τ indistinguishable by merging the templates of m_1 and m_3 (e.g., by introducing a dummy log parsing technique τ' that behaves the same as τ except for $\tau'(m_1) = \tau'(m_3) = t_{13}$). Notice that τ' changes only (a few) templates, not the corresponding log messages, meaning that the original datasets remain the same. Using this idea, to generate $R' = R_{ind}$ from $R = R_{dst}$, we generated the templates of R_{ind} by iteratively merging the templates of R_{dst} until $d(R_{ind}) = F$. Furthermore, to minimize the difference between R_{dst} and R_{ind} in terms of the number of templates (i.e., to minimize the number of templates being merged), we start with merging the templates with the highest number of matching messages in the log. This is based on the intuition that the more messages affected by merging templates, the more likely normal and abnormal logs are to become indistinguishable.

One might object that artificially merging templates corresponding to different messages could introduce incorrect templates in R_{ind}, leading to an unfair comparison between R_{dst} and R_{ind}. However, it is common for the log parsing techniques to identify many templates that are already incorrect, as mentioned in chapter 3 Furthermore, the focus of RQ2 is not the correctness of templates but rather the distinguishability of log parsing results. Our goal is to generate a pair of R_{dst} and R_{ind} that are as similar as possible except for the distinguishability property.

Algorithm 1 summarizes the above-mentioned idea into the pseudocode for generating R_{ind} from R_{dst}. After initializing R_{ind} (line 1) as a copy of R_{dst}, the algorithm extracts the set of templates T of R_{dst} (line 2) and sorts the templates in T in ascending order by the number of matching messages (line 3). The algorithm then iteratively merges the last n templates (starting from $n = 2$ as initialized at line 4) in the sorted templates list T_s (i.e., merging the top-n templates that have the highest number of matching templates) until R_{ind} becomes indistinguishable (lines 5–9). Notice that the while loop does not continue endlessly since R_{ind} must be indistinguishable when n becomes $|T_s|$ (i.e., all templates are merged into one) by definition. The algorithm ends by returning R_{ind}.

Algorithm 1: Generating an indistinguishable log parsing result from a distinguishable one

```
Input  : Distinguishable Log Parsing Result $R_{dst}$
Output: Indistinguishable Log Parsing Result $R_{ind}$
1  Log Parsing Result (Set of Parsed Logs) $R_{ind} \leftarrow copy(R_{dst})$
2  Set of Templates $T \leftarrow getTemplates(R_{dst})$
3  Sorted List of Templates $T_s \leftarrow sortByNumMessages(T)$
4  Integer $n \leftarrow 2$
5  while $d(R_{ind}) = True$ do
6      Set of Templates $T_m \leftarrow getLastTemplates(T_s, n)$
7      $R_{ind} \leftarrow mergeTemplates(T_m, R_{dst})$
8      $n \leftarrow n + 1$
9  end
10 return $R_{ind}$
```

5.3.5.2 Generation of Distinguishable from Indistinguishable Log Parsing Results

When $d(R) = F$ (i.e., $R = R_{ind}$), although one could do the dual of merging templates (i.e., dividing templates), it would require to determine which templates to divide and how many templates to generate from a given template. Instead, we adopted another heuristic: we removed the normal (or abnormal) logs that are indistinguishable from abnormal (or normal) logs. This is based on our observation that, when $d(R) = F$, only a small number of normal and abnormal logs are indistinguishable. To minimize the impact of removing logs, we removed normal logs when the total number of normal logs is larger than that of abnormal logs (as it is the case for the HDFS dataset); otherwise, we removed abnormal logs (in the case of the Hadoop dataset).

Algorithm 2 shows how to generate R_{dst} from R_{ind} based on the above idea. It first extracts the set of indistinguishable logs L_{ind} from R_{ind} (line 1). It then removes either normal or abnormal logs in L_{ind} from R_{ind} to generate R_{dst} depending on the total number of normal and abnormal logs (lines 2–6). Since R_{dst} is the result of removing indistinguishable (normal or abnormal) logs from R_{ind}, R_{dst} is distinguishable. The algorithm ends by returning R_{dst}.

Algorithm 2: Generating a distinguishable log parsing result from an indistinguishable one

```
Input  : Indistinguishable Log Parsing Result $R_{ind}$
Output: Distinguishable Log Parsing Result $R_{dst}$
1 Set of Indistinguishable Logs $L_{ind} \leftarrow$ getIndistLogs($R_{ind}$)
2 if numNormalLogs($R_{ind}$) $\geq$ numAbnormalLogs($R_{ind}$) then
3     Set of Parsed Logs $R_{dst} \leftarrow R_{ind} \setminus$ getNormalLogs($L_{ind}$)
4 end
5 else
6     Set of Parsed Logs $R_{dst} \leftarrow R_{ind} \setminus$ getAbnormalLogs($L_{ind}$)
7 end
8 return $R_{dst}$
```

5.3.5.3 Treatment for Anomaly Detection Techniques using Semantic Information of Templates

Some of the anomaly detection techniques (i.e., LogRobust [ZXL$^+$19], PLELog [YCW$^+$21], LogAnomaly [MLZ$^+$19]) use the semantic information of templates, instead of simply using template IDs, by converting them into semantic vectors [JM19]. Therefore, the notion of "identical" templates for determining the distinguishability of log parsing results must be revised in terms of the semantic vectors used by these anomaly detection techniques. To do this, for each log parsing result R, we applied a clustering algorithm to the semantic vectors of all templates and considered the templates in the same cluster to be identical. Specifically, we used DBSCAN [BHN11] for clustering since it does not require the number of clusters as an input parameter. For instance, in the above example τ with m_1 and m_3, if the semantic vectors of $\tau(m_1)$ and $\tau(m_3)$ belong to the same cluster, then the templates of m_1 and m_3 are considered the same. We then followed the same heuristics described above to generate R' from R based on the clustered templates.

5.4 Results

5.4.1 RQ1: Relationship between Log Parsing Accuracy and Anomaly Detection Accuracy

All 13 log parsing techniques and 5 anomaly detection techniques completed their executions on the HDFS and Hadoop datasets. However, none of the anomaly detection techniques detects abnormal logs in the OpenStack dataset

Table 5.3: Spearman correlation coefficients between log parsing accuracy (GA, PA, and FTA) and anomaly detection accuracy (F1 score)

	HDFS (reduced)			Hadoop		
AD technique	GA	PA	FTA	GA	PA	FTA
DeepLog	-0.166	0.259	0.198	-	-	-
LogAnomaly	0.265	0.142	0.215	-	-	-
LogRobust	0.360	0.193	0.165	-	-	-
CNN	0.425	0.575	0.508	-	-	-
PLELog	0.155	0.611	0.605	-0.233	-0.017	-0.144

(i.e., the F1 score is zero). This could be due to the very small number of abnormal logs in the dataset (only 4 out of 2068, as reported in Table 5.2). Therefore, we disregard the results for OpenStack. For all tuples $\langle GA, PA, FTA, PR, RE, F1\rangle$ we collected for HDFS and Hadoop, Figure 5.2 and Figure 5.3 show the relationship between $\langle GA, PA, FTA\rangle$ (x-axis) and $F1$ (y-axis) for HDFS and Hadoop, respectively, in the form of a scatter plot. To additionally distinguish the main results for different anomaly detection techniques, we used different shapes and colors: ○ = DeepLog, ◇ = LogAnomaly, △ = LogRobust, ⬠ = CNN, and □ = PLELog. For example, the top left subfigure in Figure 5.2 shows 13 data points where 13 log parsing techniques are used in combination with DeepLog.

Table 5.3 additionally shows the values of the Spearman's rank correlation coefficient $\sigma\langle X, Y\rangle$ between $X = \langle GA, PA, FTA\rangle$ and $Y = F1$ for each pair of anomaly detection technique and dataset. The value of $\sigma\langle X, Y\rangle$, ranging between -1 and $+1$, is an indication of the strength of the monotonic (not necessarily linear) relationship between X and Y; when $\sigma\langle X, Y\rangle \geq +0.7$ (or $\sigma\langle X, Y\rangle \leq -0.7$), there is a *strong* positive (or negative) correlation between X and Y [AAAH22]. Note that, on the Hadoop dataset, $\sigma\langle X, Y\rangle$ could not be computed for DeepLog, LogAnomaly, LogRobust, and CNN since the F1 score does not vary at all with $\langle GA, PA, FTA\rangle$, indicating no relationship.

Overall, Figure 5.2, Figure 5.3, and Table 5.3 clearly show that there is no strong correlation between $\langle GA, PA, FTA\rangle$ and $F1$ in all the cases where $\langle GA, PA, FTA, PR, RE, F1\rangle$ tuples were successfully collected. For example, in Figure 5.2, LogAnomaly (◇) achieved an F1 score ranging between 0.2 and 0.5 regardless of the GA score. This means that increasing log parsing accuracy does not necessarily increase (or decrease) anomaly detection accuracy. This is counter-intuitive since anomaly detection uses log parsing results, and having "better" log parsing results is expected to increase anomaly detection accuracy. However, this happens because even inaccurate log parsing results

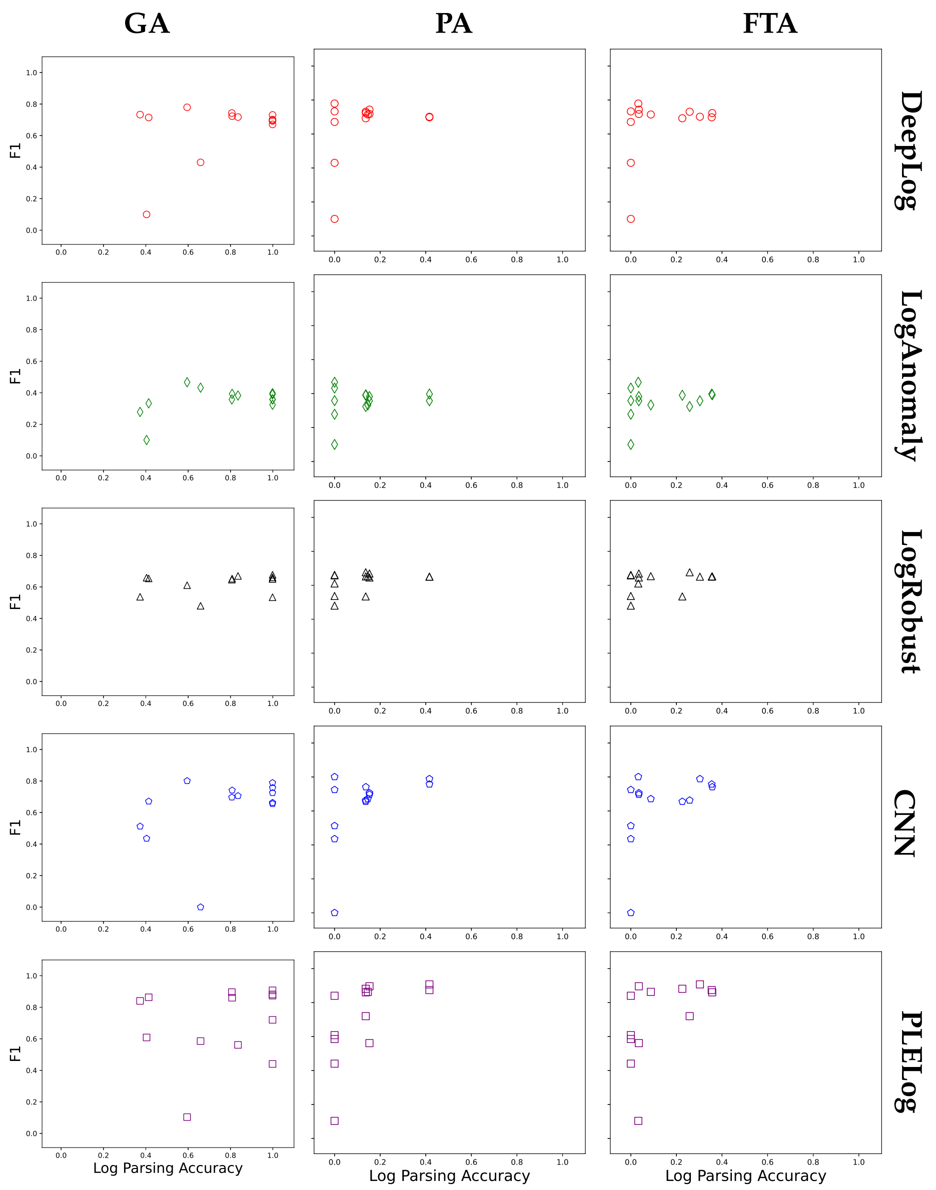

Figure 5.2: Relationship between TI accuracy and AD accuracy (HDFS)

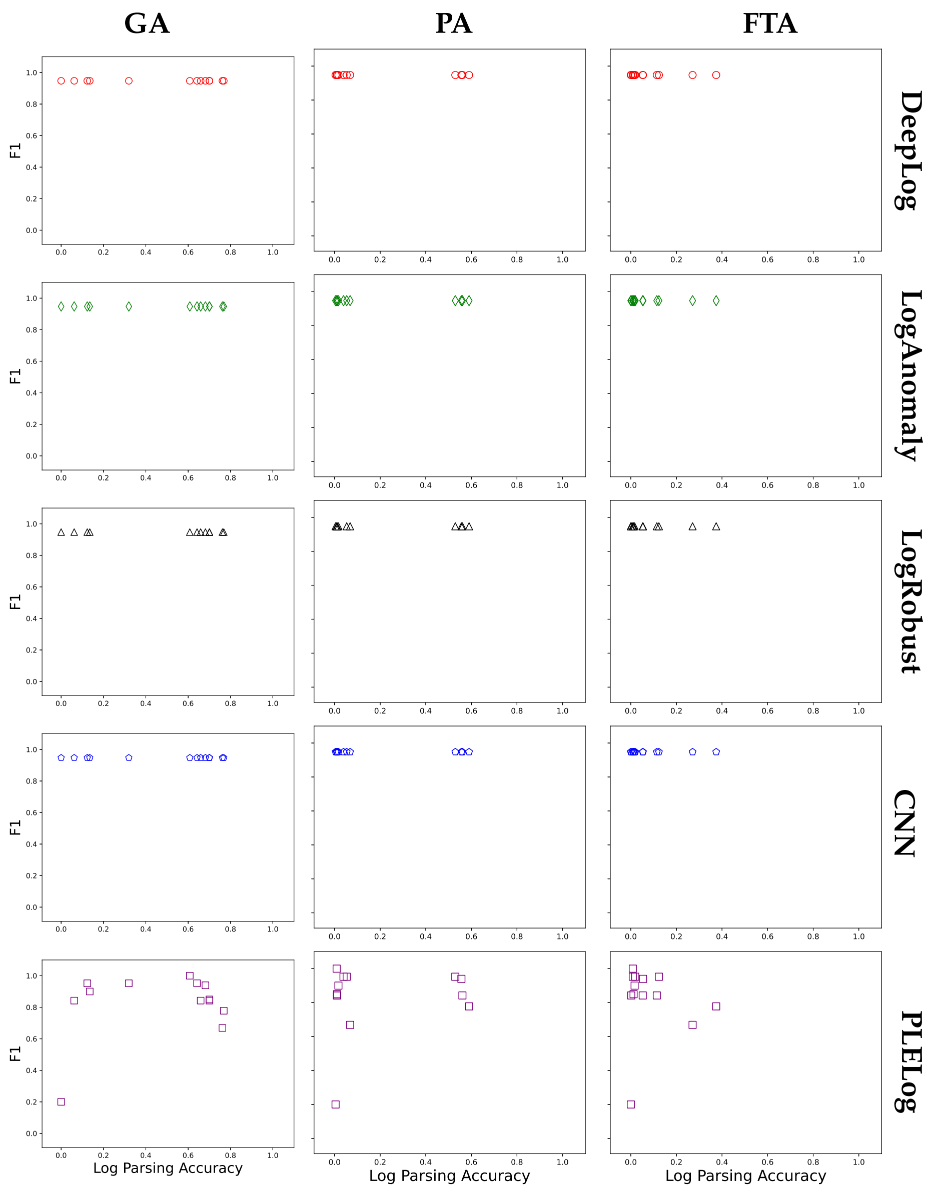

Figure 5.3: Relationship between TI accuracy and AD accuracy (Hadoop)

can lead to accurate anomaly detection results, for reasons explained below.

To better understand the reason for the above results, let us consider the following two extreme cases separately:

(C1) The log parsing accuracy values for input logs are the same, but the resulting anomaly detection accuracy values are different (i.e., the data points located on the same vertical lines in Figure 5.2 and Figure 5.3).

(C2) The log parsing accuracy values for input logs are different, but the resulting anomaly detection accuracy values are the same (i.e., the data points located on the same horizontal lines in Figure 5.2 and Figure 5.3).

To identify the root cause of C1, we manually investigated several pairs of data points in Figure 5.2 and Figure 5.3, such as two different HDFS log parsing results having almost the same log parsing accuracy value (GA scores of 0.37 and 0.40) but resulting in significantly different anomaly detection accuracy values (F1 scores of 0.73 and 0.10) for the same anomaly detection technique (DeepLog). It turned out that, although the log parsing accuracy values are similar, different log messages are correctly parsed in the two different log parsing results. This happened because the log parsing accuracy metrics (GA, PA, and FTA) summarize the log parsing results based on an implicit assumption that all log messages (and templates) are equally important. However, this assumption does not hold when it comes to anomaly detection, which must discriminate different log message templates to learn abnormal sequences of templates. Therefore, this mismatch of assumptions between log parsing and anomaly detection leads to case C1.

As for case C2, similar to the above case, we manually investigated several pairs of data points in Figure 5.2 and Figure 5.3, such as two different Hadoop log parsing results having significantly different log parsing accuracy values (GA scores of 0.12 and 0.77) but resulting in the same anomaly detection value (F1 score of 0.98) for the same anomaly detection technique (DeepLog). We found that anomaly detection techniques can distinguish between normal and abnormal patterns even when input log message templates are incorrect. To best explain this using a simplified example, let us consider a normal log $l_n = \langle m^n_1, m^n_2, \dots \rangle$ and an abnormal log $l_a = \langle m^a_1, m^a_2, \dots \rangle$, where m^x_i indicates the i-th log message in l_x for $x \in \{n, a\}$. Using oracle templates, we can group the log messages having the same template and represent l_n and l_a as groups; specifically, let $g_{orc}(l_x)$ be a sequence of message group indices (i.e., the i-th element of $g_{orc}(l_x)$ is the message group index of m^x_i). In this context, let us take two logs from the Hadoop dataset as a concrete example where $g_{orc}(l_n) = \langle 1, 2, 3, 4, \dots \rangle$ and $g_{orc}(l_a) = \langle 5, 5, 5, 6, \dots \rangle$.

When templates generated by LogMine are used to group messages instead of oracle templates, the sequences of message group indices change to $g_{LM}(l_n) = \langle 1, 2, 3, 3, \dots \rangle$ and $g_{LM}(l_a) = \langle 7, 8, 9, 10, \dots \rangle$. These are clearly different from $g_{orc}(l_n)$ and $g_{orc}(l_a)$, respectively; in particular, m_3^n and m_4^n are incorrectly grouped together in $g_{LM}(l_n)$ while m_1^a, m_2^a, and m_3^a are incorrectly separated in $g_{LM}(l_a)$. The incorrect groupings of LogMine clearly reduce the GA score (as well as PA and TA scores since incorrect groupings imply incorrect templates). However, even the incorrect $g_{LM}(l_n)$ and $g_{LM}(l_a)$ are still different enough from each other for anomaly detection techniques to distinguish between normal and abnormal patterns. This example not only shows why case C2 happened, but also demonstrates the importance of *distinguishability* in log parsing results for anomaly detection; we will further investigate this aspect in RQ2.

Before we conclude RQ1, one might be curious to know why DeepLog, LogAnomaly, LogRobust, and CNN result in the same anomaly detection accuracy value on the Hadoop dataset (as shown in Figure 5.3 [GA-Hadoop] and Table 5.3). This happens because (1) the test set of Hadoop contains only 11 logs (1 normal and 10 abnormal logs, although the number of log messages is in the same order of magnitude as HDFS; see Table 5.2 for more details) and (2) the four anomaly detection techniques classified all the 11 logs in the test set as abnormal. We speculate that PLELog shows different results from the other anomaly detection techniques because PLELog uses a very different deep learning model (i.e., an attention-based GRU [CvMBB14]). Notice that, in all cases, the results still corroborate that log parsing accuracy and anomaly detection accuracy do not have any strong relationship.

We want to note that the log parsing accuracy results shown in Figure 5.2 and Figure 5.3 are inconsistent with the ones reported in previous studies [ZHL+19, DLC+20] since the latter only considered 2K log messages, randomly sampled from the original logs, to assess log parsing accuracy.

The answer to RQ1 is that there is no strong correlation between log parsing accuracy and anomaly detection accuracy; increasing log parsing accuracy does not necessarily increase anomaly detection accuracy, regardless of the metric (GA, PA, or TA) used for measuring log parsing accuracy.

Table 5.4: Impact of the distinguishability log parsing results on anomaly detection accuracy for the HDFS (reduced) dataset

	DeepLog (F1)			LogAnomaly (F1)			LogRobust (F1)			CNN (F1)			PLELog (F1)		
Log Parser	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ
AEL	0.747	0.561	0.186	0.509	0.320	0.189	0.663	0.456	0.207	0.772	0.662	0.110	0.760	0.033	0.727
Drain	0.714	0.523	0.191	0.499	0.400	0.099	0.703	0.454	0.250	0.757	0.682	0.075	0.796	0.286	0.510
IPLoM	0.760	0.590	0.170	0.481	0.268	0.213	0.556	0.380	0.176	0.810	0.588	0.222	0.849	0.041	0.808
LFA	0.803	0.693	0.110	0.606	0.378	0.228	0.355	0.299	0.056	0.755	0.548	0.207	0.100	0.000	0.100
LenMa	0.808	0.625	0.184	0.484	0.285	0.199	0.659	0.436	0.223	0.814	0.607	0.207	0.681	0.271	0.411
LogCluster	0.263	0.097	0.166	0.380	0.243	0.138	0.542	0.300	0.241	0.498	0.306	0.192	0.426	0.317	0.108
LogMine	0.732	0.552	0.180	0.453	0.363	0.090	0.554	0.329	0.225	0.792	0.612	0.179	0.817	0.439	0.378
Logram	0.544	0.420	0.124	0.836	0.625	0.212	0.330	0.163	0.167	0.100	0.000	0.100	0.100	0.000	0.100
MoLFI	0.794	0.630	0.164	0.427	0.282	0.144	0.565	0.319	0.246	0.781	0.621	0.160	0.172	0.109	0.063
SHISO	0.778	0.629	0.149	0.544	0.238	0.306	0.679	0.446	0.233	0.796	0.589	0.207	0.839	0.341	0.498
SLCT	0.743	0.570	0.173	0.268	0.160	0.108	0.394	0.244	0.150	0.743	0.607	0.136	0.725	0.534	0.191
Spell	0.765	0.598	0.167	0.289	0.176	0.113	0.401	0.241	0.160	0.805	0.616	0.189	0.665	0.304	0.361
Average	0.704	0.541	0.164	0.481	0.312	0.170	0.533	0.339	0.194	0.702	0.536	0.165	0.577	0.223	0.355

Table 5.5: Impact of the distinguishability log parsing results on anomaly detection accuracy for the Hadoop dataset

	DeepLog (F1)			LogAnomaly (F1)			LogRobust (F1)			CNN (F1)			PLELog (F1)		
Log Parser	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ	R_{dst}	R_{ind}	Δ
AEL	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.785	0.507	0.278
Drain	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.900	0.000	0.900
IPLoM	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.848	0.000	0.848
LFA	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.888	0.799	0.090
LenMa	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.900	0.505	0.395
LogCluster	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.952	0.180	0.772
LogMine	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.848	0.530	0.318
Logram	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.842	0.797	0.046
MoLFI	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.900	0.739	0.161
SHISO	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.900	0.000	0.900
SLCT	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.842	0.000	0.842
Spell	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.703	0.188	0.515
Average	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.947	0.847	0.100	0.859	0.354	0.505

5.4.2 RQ2: Log Parsing Distinguishability and Anomaly Detection Accuracy

Table 5.4 shows the anomaly detection accuracy values (F1 scores) when different log parsing techniques (rows) and anomaly detection techniques (columns) are used together on the HDFS (reduced) dataset; under each of the anomaly detection technique columns, sub-columns R_{dst} and R_{ind} indicate the F1 scores for distinguishable and indistinguishable log parsing results, respectively, and Δ indicates the difference between R_{dst} and R_{ind}. For example, if we choose AEL for log parsing and DeepLog for anomaly detection, the F1 score decreases from 0.747 to 0.561 when R_{ind} is used instead of R_{dst}. The same structure applies to Table 5.5, which shows the results on the Hadoop dataset. In Table 5.5, except for PLELog, the values for all anomaly detection techniques are identical due to the reasons explained in the last paragraph of Section 5.4.1.

In all cases, Δ is positive, ranging from 0.056 (LFA-LogRobust on the HDFS (reduced) dataset) to 0.9 (SHISO-PLELog on the Hadoop dataset). This means that the anomaly detection accuracy decreases up to 90 percentage points (pp) when R_{ind} is used instead of R_{dst}. To see if the differences between R_{dst} and R_{ind} are significant, we applied the non-parametric Wilcoxon signed rank test [Wil92] for paired samples to the F1 scores of R_{dst} and R_{ind}, for each of the five anomaly detection techniques and the two datasets. The results show that, for all the anomaly detection techniques and datasets, the differences between R_{dst} and R_{ind} are significant (p-value < 0.005) in terms of anomaly detection accuracy.

Considering the definition of distinguishability for log parsing results, it is intuitive that indistinguishable log parsing results should lead to lower anomaly detection accuracy. However, it is surprising that this decrease in accuracy is, in some cases, rather limited, e.g., only 0.046 for Logram on the Hadoop dataset when PLELog is used for log parsing. This happens because an indistinguishable log parsing result may only have a few logs that are indistinguishable in terms of normal and abnormal behavior. Recall that we did not explicitly control the number of indistinguishable logs since we aimed to minimize the difference between distinguishable and indistinguishable versions of each log parsing result as described in Section 5.3.5. Nevertheless, the results shown in Tables 5.4 and 5.5 are sufficient to confirm the strong impact of distinguishability in log parsing results on anomaly detection accuracy.

The answer to RQ2 is that the impact of the distinguishability of log parsing results on anomaly detection accuracy is significant for all anomaly detection techniques.

5.4.3 Threats to Validity

The used oracle templates determine log parsing accuracy values. For example, as mentioned in chapter 3, manually extracting oracle templates by investigating log messages without accessing the corresponding source code could result in biased, incorrect oracle templates. This could be a significant threat to the validity of our results. To mitigate this, we perused the source code (of the exact version that generated the logs) for each software system and used the templates directly extracted from the source code. Although this made us exclude a few log datasets whose source code was unavailable, it was beneficial to ensure the validity of our results.

Individual log parsing and anomaly detection techniques have distinct hyper-parameters, which might significantly affect the log parsing and anomaly detection results. To mitigate this, we used the same hyper-parameter values proposed by the authors, when available; otherwise, we ran preliminary experiments and used the values that resulted in the same results reported in the corresponding papers.

Using a specific set of log datasets is a potential threat to external validity. Though the datasets we considered include the logs of various systems, we had to select HDFS, Hadoop, and OpenStack due to the reasons discussed in Section 5.3.1. Therefore, even though the datasets have been widely used in existing literature [LZ22, CLG$^+$21] on log-based anomaly detection, they may not capture diverse characteristics of log data. Further experiments with different datasets are required to improve the generalizability of our results.

In RQ2, we artificially generated pairs of distinguishable and indistinguishable log parsing results to systematically assess the impact of the distinguishability of log parsing results on anomaly detection accuracy using balanced data. To mitigate any bias introduced during the process, we carefully designed Algorithms 1 and 2 to minimize the difference between each pair of log parsing results, except for their distinguishability property. Note that, although the pair generation process (by merging templates) might look unrealistic, it reflects what frequently happens in real-world scenarios; for example, it is not uncommon for log parsing techniques to misidentify templates so that messages with different oracle templates are mapped to the same (misidentified) template.

5.5 Findings and Implications

One of the most surprising results from our evaluation is that, using all existing log parsing accuracy metrics in the literature, we did not find any significant correlation with anomaly detection accuracy. In other words, more accurate log parsing results are not necessarily better for anomaly detection accuracy. This implies that log parsing accuracy is not a good indicator of the quality of log parsing results for anomaly detection purposes. As explained with an example in Section 5.4.1, this happens because inaccurate log parsing results can still be useful for anomaly detection as long as normal and abnormal logs are distinguishable. At the extreme, a log parsing result R_{50} with 50% accuracy could be better for anomaly detection than a log parsing result R_{100} with 100% accuracy if R_{50} distinguishes normal and abnormal logs while R_{100} does not.

This surprising finding leads to an important practical implication: When used for anomaly detection purposes, we can no longer choose a log parsing technique based on accuracy. Instead, as shown in Section 5.4.2, the distinguishability of log parsing results should be the main selection criterion. For example, since normal and abnormal logs are often used for training anomaly detection models, candidate log parsing results should be compared in terms of their capability to distinguish normal and abnormal logs. If there are multiple techniques that can equally distinguish between normal and abnormal logs, then the one with the lowest number of identified templates would be preferred since reducing the number of templates would increase the performance of anomaly detection by reducing dimensionality (i.e., the number of features considered in machine learning models), as mentioned in chapter 4.

Though our objective here is not to identify the "best" log parsing and anomaly detection techniques, through our experiments, we found that there is no single best technique that significantly outperforms the others in all cases. In the future, to develop better log parsing techniques targeting anomaly detection, it would beneficial to focus on distinguishability, which has not been the case so far.

5.6 Related Work

Although individual techniques for log parsing and anomaly detection have been studied for a long time, systematic studies covering several techniques have only recently begun to emerge. For example, the most comprehensive evaluation studies on many log parsing techniques [ZHL$^+$19, DLC$^+$20, KSBB22b] were conducted over the last four years. Similarly, the relationship between log parsing and anomaly detection has received little attention until very recently. Below, we summarize the recent studies related to this topic.

In chapter 4, we presented the first theoretical study considering the relationship between log parsing and anomaly detection. We established the concept of ideal log parsing results for anomaly detection. We adopted the theoretical foundation, especially the notion of *distinguishability* in log parsing results, and empirically showed that distinguishability is indeed essential for anomaly detection. To the best of our knowledge, our work is the first empirical study showing the importance of log parsing distinguishability for anomaly detection.

As explained in Section 5.2, Le and Zhang [LZ22] presented an empirical study on factors that could affect anomaly detection accuracy. Although a part of their study investigated the impact of log parsing on anomaly detection accuracy, they investigated four log parsing techniques but did not

Table 5.6: Comparison with related empirical studies

Category	Le and Zhang. [LZ22]	Fu et al. [FYX$^+$23]	Our work
Objective	Investigate different factors that might affect anomaly detection accuracy	Investigate the impact of log parsing techniques on anomaly detection accuracy	Evaluate the impact of log parsing accuracy and the distinguishability of log parsing results on anomaly detection accuracy
Log parsing accuracy metrics	N/A	PA	PA, GA, and TA
Oracle templates	N/A	Manually generated for 2K sample log messages	Extracted from the corresponding source code
Logs used for measuring log parsing accuracy	N/A	Only a small fraction of logs actually used for anomaly detection	All logs used for anomaly detection
Log parsing techniques	Drain, Spell, IPLoM and AEL	Drain, Spell, IPLoM, LFA, Logram, and LenMa	Drain, Spell, IPLoM, AEL, LFA, Logram, LenMa, LogSig, LogCluster, LogMine, SHISO, MoLFI, and SLCT
Anomaly detection techniques	DeepLog, LogRobust, LogAnomaly, PLELog, and CNN	DeepLog, LogRobust, Principal Component Analysis (PCA), LogClustering, Logistic Regression (LR), and Decision Tree (DT)	DeepLog, LogRobust, LogAnomaly, PLELog, and CNN
Distinguishability [SK	Not considered	Not considered	Considered

assess the impact of log parsing accuracy. As a result, they only showed that using different log parsing techniques leads to different anomaly detection accuracy scores. In our study, on the other hand, we explicitly measured log parsing accuracy, collected 160 pairs of log parsing accuracy and anomaly detection accuracy values using different combinations of log parsing and anomaly detection techniques, and showed that there is no strong correlation between log parsing accuracy and anomaly detection accuracy.

During the writing of this thesis, Fu et al .[FYX+23] also presented an empirical study on the impact of log parsing on anomaly detection performance. Although their motivation and research questions are close to ours, there are several key differences. First, for measuring log parsing accuracy, they used the manually generated, error-prone oracle templates [KSBB22b] provided with the 2K log messages randomly sampled by Zhu et al. [ZHL+19]. In other words, only a very small fraction of the logs used for anomaly detection was used to measure log parsing accuracy in their study. In our study, however, the same logs used for anomaly detection are used to measure log parsing accuracy, and the oracle templates are directly extracted from the corresponding source code. Second, they considered only one log parsing accuracy metric (GA), whereas we considered all three log parsing metrics (GA, PA, and TA) since different metrics assess complementary aspects of log parsing, as mentioned in chapter 3. Third, log parsing distinguishability, which is an essential factor that substantially affects anomaly detection accuracy (as shown in our RQ2), is only considered in our study. Finally, they only considered two deep-learning-based anomaly detection techniques (DeepLog and LogRobust), and focused also on more traditional machine learning approaches (such as Principal Component Analysis, clustering, logistic regression, and decision trees). Such differences allow us to report new findings and provide concrete recommendations, as summarized in Section 5.5.

Wu et al. [WLK23] recently presented an empirical study on the effectiveness of log representation for machine learning-based anomaly detection. They considered different log representation techniques, such as FastText [JGB+16], Word2Vec [MCCD13], TF-IDF [SB88] and BERT [DCLT18], used to convert textual log data into numerical feature vectors for machine learning algorithms, such as Support Vector Machine, Logistic Regression, Random Forest, CNN, and LSTM. As a part of their study, they investigated the impact of log parsing on anomaly detection when used with different log representation techniques (in particular, FastText and Word2Vec). The empirical results showed that, in general, using log parsing (i.e., Drain [HZZL17]) improves the quality of log representations (over raw, unparsed data) and thereby the performance of anomaly detection; they also reported that some models (e.g., CNN and LSTM) are less sensitive to whether the log data is parsed or not, possibly due to the strong feature extraction and representation ability, and can offset the impact of noise generated by log parsing. In addition to these results, they also investigated the impact of additionally refining log parsing results using regular expressions and the impact of using different log parsing techniques. The results showed that refining log parsing results do not significantly increase anomaly detection performance but using different log parsing techniques yields slight variations in anomaly detection performance. However, for these additional investigations, they used only one anomaly detection technique (i.e., Logistic Regression) and two log parsing techniques (i.e., Drain [HZZL17] and LogPPT [LZ23]). Furthermore, they did not study the relationship between log parsing accuracy and anomaly detection accuracy. On the contrary, we use 13 log parsing techniques and 5 DL-based anomaly detection techniques to comprehensively investigate the relationship between log parsing accuracy and anomaly detection accuracy.

Table 6.8 summarizes the key differences between the closely-related previous empirical studies (i.e., Le and Zhang [LZ22], Fu et al. [FYX$^+$23]) and our work.

5.7 Summary

In this chapter, we reported on a comprehensive empirical study investigating the impact of log parsing on anomaly detection accuracy, using 13 log parsing techniques and five DL-based anomaly detection techniques on two publicly available log datasets. When analyzing log parsing results for anomaly detection, we were surprised not to find any significant relationship between log parsing accuracy and anomaly detection accuracy, regardless of the metric used for the former (including GA, PA, and FTA). This implies that, as opposed to common research practice to date, we can no longer select a log parsing technique based on its accuracy when used for anomaly detection. Instead, we experimentally confirmed existing theoretical results showing that the distinguishability of log parsing results plays an essential role in achieving accurate anomaly detection. It is therefore highly recommended that in the future we rely on distinguishability to evaluate, compare, and guide the development of log parsing techniques.

Chapter 6

Deep Learning-Based Anomaly Detection: A Comparison of Non-Log-Parsing-Based and Log-Parsing-Based Approaches

6.1 Overview

With the recent developments of many log-parsing techniques [JHFH08, HZZL17, MZHM09, Shi16] and log-parsing-based anomaly detection techniques [ZXL$^+$19, MLZ$^+$19, DLZS17] in the literature, the relationship between log parsing and anomaly detection has gained more and more attention. For example, in Chapter 4, we defined a theoretical framework for discussing ideal log parsing results for anomaly detection. Specifically, we formalized the concepts of distinguishability and minimality showing that log parsing results that conforms to these two concepts between normal and abnormal logs achieve the ideal operating conditions for anomaly detection. In Chapter 5, we thoroughly investigated the impact of log parsing on anomaly detection, and provided a comprehensive evaluation by quantitatively assessing the impact of log parsing on anomaly detection accuracy. Other researchers also studied the impact of log parsing on anomaly detection. Fu et al. [FYX$^+$23] empirically investigated the impact of log parsing techniques on the effectiveness and efficiency of anomaly detection methods. Le and Zhang [LZ22] also empirically showed that different log parsing techniques can affect anomaly detection accuracy.

However, not all log-based anomaly detection techniques require log parsing as a pre-processing step. One of the state-of-the-art anomaly detection techniques, NeuralLog [LZ21], directly processes semi-structured logs. This is done by representing log messages' semantic information in the form of semantic vectors, including information such as verbosity level and component information. These semantic vectors are then used by the Transformer-based classification model to capture the contextual information from the log sequences and to detect anomalies. This brings a clear advantage over the other log-parsing-based techniques in that it is not affected by log parsing. However, the use of semantic information recorded in log messages would require significant computations, which might result in a scalability issue. Nevertheless, how the log-parsing-based and non-log-parsing-based anomaly detection techniques fare in terms of accuracy and execution time remains unclear.

In this chapter, we comprehensively compare one state-of-the-art non-log-parsing-based technique (NeuralLog [LZ21]) and five widely used log-parsing-based anomaly detection techniques (DeepLog [DLZS17], LogAnomaly [MLZ$^+$19], LogRobust [ZXL$^+$19], PLELog [YCW$^+$21], and CNN [LWLW18]) in combination with two different state-of-the-art log parsing techniques (Drain [HZZL17] and LogPPT [LZ23]) using eight datasets (HDFS, HDFS US, Hadoop, Hadoop US, BGL, Spirit, Thunderbird, and Thunderbird US, where *US* stands for under-sampled datasets).

We will focus on anomaly detection accuracy and execution time but additionally consider other elements (e.g., the number of log templates) to further investigate the results. To this end, we aim to guide practitioners in selecting the most suitable anomaly detection technique for their specific use cases.

The remainder of the chapter is organized as follows: Section 6.2 discusses the motivation of the study. Section 6.3 describes the evaluation subjects, including the datasets utilized, and anomaly detection techniques used. Section 6.4 presents the experimental results. Section 6.5 discusses the findings and implications derived from the experiments conducted. Section 6.6 reviews the related work. Finally, Section 6.7 concludes the chapter.

6.2 Motivation

Many log-parsing-based anomaly detection techniques have been proposed in the literature [DLZS17, ZXL$^+$19, YCW$^+$21, LWLW18]. As mentioned in chapter 2, these anomaly detection techniques require pre-processing log data to extract essential patterns or features prior to applying anomaly detection algorithms. Transforming semi-structured or free-formed log data into a more structured representation using log parsing might enhance the performance of anomaly detection techniques by utilizing not all the data available in logs, but only the event template IDs identified [LWLW18, DLZS17] or the semantic information in event templates [ZXL$^+$19, YCW$^+$21].

While these anomaly detection techniques have demonstrated promising results, certain limitations still remain. One of the limitations of log-parsing-based techniques is that they incur into pre-processing (log-parsing) overhead, which can be time-consuming, specifically when dealing with large amounts of log data. Moreover, the accuracy of these methods might rely on the accuracy of the log parsing process. However, as mentioned in chapter 5, there is no significant relationship between log parsing accuracy and anomaly detection accuracy, regardless of the log parsing accuracy metric used.

To investigate alternative approaches and alleviate the limitations of log-parsing-based techniques, Le and Zhang [LZ21] proposed an anomaly detection technique called NeuralLog. What distinguishes NeuralLog from log-parsing-based techniques is its ability to perform anomaly detection without requiring log parsing as a pre-processing step. By leveraging neural network architectures and transformers, NeuralLog claims more accurate and streamlined anomaly detection while avoiding the overhead associated with log preprocessing. They compared NeuralLog accuracy with traditional machine learning techniques, such as Support Vector Machine (SVM), Logistic Regression (LR), Invariant Mining (IM), Log2Vec, and a deep-learning-based technique called LogRobust. Their results show that NeuralLog is more accurate than the others.

However, using one deep-learning-based technique to compare is not enough to generalize the results, and thus it is crucial to compare more deep-learning-based anomaly detection techniques to have a fair evaluation ground for comparison. To this end, we define our first research question as follows: **How does the accuracy of anomaly detection compare between non-log-parsing and log-parsing-based techniques?** Furthermore, although NeuralLog offers improved accuracy, Le and Zhang [LZ21] did not consider comparing anomaly detection efficiency, an important factor in real-time-based anomaly detection use cases. Therefore, we define our second research question as follows: **How does the efficiency of anomaly detection compare between non-log-parsing and log-parsing-based techniques?**

By evaluating the efficiency and accuracy of the two approaches, we in-tend to provide valuable insights into their respective limitations and strengths. This comprehensive empirical study aims to serve as a guide for selecting the most suitable anomaly detection technique based on specific constraints and requirements. For example, applications or industries with strict real-time processing demands might benefit from the efficiency gains of a certain technique; on the other hand, those requiring precise log data understanding may still depend on log-parsing-based techniques.

Furthermore, the insights acquired from this study could motivate the development of hybrid approaches that exploit the strengths of non-log-parsing-based and log-parsing-based techniques. By bringing together the advantages of each approach, researchers may develop more sophisticated and robust anomaly detection techniques capable of handling the challenges imposed by log data volumes being generated in vast quantities.

6.3 Evaluation Subjects

6.3.1 Datasets

To answer the research questions, we used publicly accessible datasets from the LogHub benchmark [HZHL20]. This benchmark encompasses an extensive selection of log messages originating from diverse system types, including operating systems (Linux, Windows, and Mac), distributed systems (BGL, Hadoop, HDFS, Thunderbird, and OpenStack), standalone programs (Zookeeper, and Proxifier), as well as mobile systems (Android). Among them, we selected five datasets (HDFS, OpenStack, Hadoop, Thunderbird, and BGL), since each log message in these datasets is manually labeled as normal or anomaly, which is necessary for anomaly detection models based on supervised learning [ZXL$^+$19, LZ21]. Furthermore, their widespread adoption in the research community emphasizes their reliability and comprehensiveness. Using these datasets ensures a solid foundation for our research.

Among the five datasets, we had to exclude OpenStack because none of the anomaly detection techniques detected any anomalies in the OpenStack logs due to the very small number of abnormal logs. Instead, we included the Spirit dataset [OS07], following a recently published paper [LZ22], to make our study more comprehensive.

For the remaining five datasets (HDFS, Hadoop, Thunderbird, BGL, and Spirit), we additionally generated variants of them using undersampling. Undersampling is a technique to balance the majority and minority classes in a training dataset in machine learning. We had to apply undersampling to the original datasets because only NeuralLog internally performs a specific undersampling[1], which introduces a significant bias by modifying the training dataset. Specifically, NeuralLog undersampling consists of two conditions i.e., (i) if the dataset contains more normal instances than abnormal instances, then limit the normal instances to five times the number of abnormal instances, and (ii) if the dataset contains more abnormal instances than normal instances, then limit abnormal instances equal to the number of normal instances. For example, Thunderbird has 7971548 and 28432 normal and abnormal logs, respectively, in a training set, so the number of normal logs is reduced to 142160 ($= 28432 \times 5$) after the undersampling. Furthermore, NeuralLog performs the special undersampling for supercomputer datasets (BGL, Spirit, Thunderbird), although (1) Hadoop and HDFS also meet the condition for doing so, and (2) BGL and Spirit are not impacted by the undersampling. To address such issues, we decided to apply the same undersampling to all datasets consistently, resulting in new datasets dubbed Thunderbird-US, Hadoop-US, and HDFS-US, where US refers to the undersampled version. As a result, we have eight datasets (five original datasets and three US datasets). This allows us to use the same datasets for all anomaly detection techniques.

Table 6.1 provides detailed information about these datasets, in terms of the number of all messages (M_{all}), number of templates identified by Drain (T_{Drain}) [HZZL17], number of templates identified by LogPPT (T_{LogPPT}) [LZ23], grouping strategy used, the number of all logs (L_{all}), the number of normal logs (L_n), the number of abnormal logs (L_a), the total number of instances in the training set (TR_{tot}), the number of normal instances in the training set (TR_{nor}), the number of abnormal instances in the training set (TR_{abn}), the total number of instances in the testing set (TS_{tot}), the number of normal instances in the testing set (TS_{nor}), and the total number of abnormal instances in the testing set (TS_{abn}). Note that TR_{tot} and TS_{tot} are based on log parsing results divided into two disjoint sets of training and testing using 80 and 20 ratios respectively, and the number of log messages is the same as the number of log entries (see Chapter 2 for details).

6.3.2 Anomaly Detection Techniques

Our study involves a comparison between log-based anomaly detection techniques and non-log-based anomaly detection techniques. To achieve this, we leveraged five deep learning DL-based anomaly detection techniques:

[1]Although the specific undersampling of NeuralLog is not mentioned in the paper [LZ21], it is implemented in the code [ZL21] in a way that it is enabled only for BGL, Spirit, and Thunderbird logs.

Table 6.1: Log datasets

Dataset	M_{all}	T_{Drain}	T_{LogPPT}	Grouping Strategy	L_{all}	L_n	L_a	TR_{tot}	TR_{nor}	TR_{abn}	TS_{tot}	TS_{nor}	TS_{abn}
HDFS	11,175,629	48	54	Session ID	575061	558223	16838	460048	446578	13470	115013	111645	3368
HDFS US	11,175,629	48	54	Session ID	195833	178995	16838	80820	67350	13470	115013	111645	3368
Hadoop	109,968	188	740	Session ID	54	11	43	42	8	34	12	3	9
Hadoop US	109,968	188	740	Session ID	28	11	17	16	8	8	12	3	9
BGL	4,747,963	631	1549	Sliding Wind.	899960	653391	246569	719980	481540	238440	179980	171851	8129
Spirit	7,983,345	1538	-	Sliding Wind.	7983305	6343005	1640300	6386656	4755160	1631496	1596649	1587845	8804
Thunderbird	10,000,000	2641	1813	Sliding Wind.	9999960	9970615	29345	7999980	7971548	28432	1999980	1999067	913
Thunderbird US	10,000,000	2641	1813	Sliding Wind.	2170572	2141227	29345	170592	142160	28432	1999980	1999067	913

DeepLog [DLZS17], LogAnomaly [MLZ$^+$19], LogRobust [ZXL$^+$19], PLELog [YCW$^+$21], and CNN [LWLW18] on eight datasets. These techniques are considered representative of the current state-of-the-art in DL-based anomaly detection. For non-log-parsing-based techniques, we identified three anomaly detection techniques (NeuralLog [LZ21], Logsy [NBA$^+$20a], and HEART [MBJV23]), for which the source code is available. For all the anomaly detection techniques, we used their default parameters.

6.4 Experiments

6.4.1 RQ1: How does the accuracy of anomaly detection (AD) compare between non-log-parsing and log-parsing-based techniques?

6.4.1.1 Methodology

To answer RQ1, for each dataset, we executed the log parsing techniques (Drain and LogPPT) to generate the log parsing results, as the input of log-parsing-based anomaly detection techniques requires the sequence of identified templates. We selected Drain since it is the most widely used and accurate log parsing technique on many datasets based on the results provided by Zhu et al. [ZHL$^+$19], Le and Zhang [LZ23] and in chapter 3. We also selected LogPPT, since it has been recently proposed [LZ23], showing high accuracy results.

Note that, we did not consider the Spirit dataset for LogPPT and thus no results are available in Table 6.3. The reason is that LogPPT requires shots (oracle templates), as it is few-shots based log-parsing technique, and as mentioned in section 6.3.1, Spirit dataset is not included in the LogHub benchmark, thus we do not have oracle templates (or shots) available. Additionally, log parsing is not required for the under-sampled datasets since under-sampling is applied to the log parsing results.

For each log-parsing-based anomaly detection technique, we divided each log parsing result into two disjoint sets: a training set and a testing set with a ratio of 80:20. Similarly, for non-log-parsing-based anomaly detection techniques (NeuralLog, Logsy, and HEART), we also divided the dataset into two disjoint sets of 80% training and 20% testing set, as suggested in the RQ1 of reference [LZ21]. We trained the anomaly detection techniques on each of the training sets and used the corresponding test sets to compute the precision, recall, and F1score $\langle PR, RE, F1 \rangle$. To account for the randomness of anomaly detection techniques, we repeated the training and testing process five times and used the average F1 score.

The experiments presented in this chapter were carried out using the HPC facilities of the University of Luxembourg (see http://hpc.uni.lu).

6.4.1.2 Results

Table 6.2 and Table 6.3 show the anomaly detection accuracy values along with the datasets and the number of templates identified by Drain and LogPPT, respectively; Table 6.4 shows the anomaly detection accuracy for non-log-parsing-based anomaly detection techniques. We can see that NeuralLog fails to detect any anomaly on the Hadoop and Hadoop US datasets; the reason is possibly due to a low number of log sequences in the training data, as shown in Table 6.1. HEART did not complete the execution for the HDFS, BGL, Spirit, and Thunderbird datasets due to their large size. This limitation is attributed to HEART's reliance on a transformer-based model, as the computational intensity of transformer-based models, characterized by a large number of parameters, poses challenges when handling extensive datasets. Moreover, LogRobust gives out-of-memory (OOM) error on the BGL, Spirit, Thunderbird, and Thunderbird US datasets. The reason is the use of semantic information: initially, each word is replaced with FastText word vectors [JGB$^+$16]; later, event templates are transformed into a word vectors list. Next, LogRobust represents a log event as a fixed-dimension vector by aggregating all word vectors. LogRobust applies weighted aggregation using TF-IDF [GdSTC23] which is a widely-used method in information retrieval. The TF-IDF weight can effectively measure the importance of words in sentences, while it is memory memory-intensive task, and consumes a lot of memory, due to a high number of identified templates as shown in Table 6.1.

By looking at the average F1-score row of Table 6.2, Table 6.3, and Table 6.4, we can see that non-log-parsing-based techniques generally exhibit higher anomaly detection accuracy than log-parsing-based anomaly detection techniques. For instance, NeuralLog is the most accurate with an aver-

Table 6.2: Anomaly Detection (AD) accuracy comparison results [Drain]

	Deeplog			LogAnomaly			LogRobust			CNN			PLELog		
Datasets	PR	RE	F1	PR	RE	F1	PR	RE	F1	PR	RE	F1	PR	RE	F1
HDFS	0.8168	0.8991	0.8544	0.2578	0.7332	0.3814	0.9058	0.9762	0.9388	0.9422	0.9749	0.9569	0.8561	0.6391	0.7154
HDFS US	0.8525	0.8935	0.8722	0.2555	0.7420	0.3800	0.8020	0.9777	0.8811	0.8020	0.9790	0.8817	0.9659	0.6039	0.7427
Hadoop	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.7394	1.0000	0.8499
Hadoop US	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.7475	0.9500	0.8347
BGL	0.1822	0.9881	0.3076	0.1614	1.0000	0.2779	-	-	**OOM**	0.7409	0.7033	0.7159	0.6920	0.7861	0.7358
Spirit	0.3139	1.0000	0.4779	0.3138	1.0000	0.4777	-	-	**OOM**	0.9870	0.9761	0.9816	0.9347	0.9021	0.9181
Thunderbird	0.1566	1.0000	0.2708	0.1566	1.0000	0.2708	-	-	**OOM**	0.5313	0.9385	0.6750	0.6240	0.1133	0.1870
Thunderbird US	0.1566	1.0000	0.2708	0.1566	1.0000	0.2708	-	-	**OOM**	0.4753	0.9538	0.6101	0.0000	0.0000	0.0000
Average	0.5348	0.9726	0.6186	0.3877	0.9344	0.4942	0.8770	0.9885	0.9287	0.7848	0.9407	0.8395	0.6949	0.6243	0.6229
Std Deviation	0.3598	0.0473	0.3150	0.3213	0.1215	0.2887	0.0500	0.0133	0.032	0.1907	0.0982	0.1483	0.3046	0.3784	0.3375

age 98% F1-score on six datasets (HDFS, HDFS US, BGL, Spirit, Thunderbird, and Thunderbird US) out of eight datasets. Following closely is Logsy, ranking as the second most accurate anomaly detection technique with a 97% F1-score. HEART, as a non-log-parsing technique, achieves an average F1-score of 79.7% across four datasets (HDFS US, Hadoop, Hadoop US, and Thunderbird) out of eight datasets. Although it attains a high accuracy of 99.9% on the Thunderbird US dataset, its accuracy falls below 74% on the remaining three datasets, resulting in relatively lower accuracy. LogRobust is the third most accurate anomaly detection technique with 92.8% F1-score on four datasets (HDFS, HDFS US, Hadoop, and Hadoop US). In comparison, Logsy, DeepLog, LogAnomaly, CNN, and PLELog achieve an average F1-score of 61.8%, 49.4%, 83.9%, and 62.2%, respectively, using log parsing results from Drain. Meanwhile, their F1-scores drop to 45.9%, 37%, 51.6%, and 35.4%, respectively, when using log parsing results from LogPPT.

To conclude, the answer to RQ1 is that the non-log-parsing-based anomaly detection techniques achieves the highest average F1 score. Although LogRobust achieves on average 92.8% while it does not detect any anomaly on four out of eight datasets. NeuralLog, Logsy, and HEART also does not require any pre-processing step, making them even more usable. Thus, NeuralLog, and Logsy are better than the log-parsing-based AD techniques in terms of accuracy. HEART, a non-log-parsing-based anomaly detection technique, achieves high accuracy of 99.9% on the Thunderbird US dataset. However, it fails to detect anomalies on four out of eight datasets due to timeouts, making it less practical for real-world applications.

Table 6.3: Anomaly Detection (AD) accuracy comparison results [LogPPT]

	Deeplog			LogAnomaly			LogRobust			CNN			PLELog		
Datasets	PR	RE	F1	PR	RE	F1	PR	RE	F1	PR	RE	F1	PR	RE	F1
HDFS	0.6202	0.4624	0.5074	0.1566	0.3684	0.2197	0.8058	0.4732	0.5897	0.6459	0.5132	0.5357	0.6202	0.4624	0.5074
HDFS US	0.6140	0.4649	0.5112	0.1361	0.3301	0.1925	0.5824	0.5996	0.5353	0.4227	0.5761	0.4839	0.9744	0.5794	0.7246
Hadoop	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474
Hadoop US	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.9000	1.0000	0.9474	0.0000	0.0000	0.0000
BGL	0.1771	0.9881	0.3003	0.1666	0.9974	0.2855	-	-	**OOM**	0.7839	0.6781	0.7029	0.1771	0.9881	0.3003
Spirit	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Thunderbird	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-	-	**OOM**	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Thunderbird US	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-	-	**OOM**	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Average	0.4587	0.5593	0.4591	0.3227	0.5280	0.3704	0.7971	0.7682	0.75495	0.5218	0.5382	0.5168	0.3817	0.4328	0.3542
Std Deviation	0.3958	0.4502	0.3934	0.4003	0.4633	0.4086	0.1498	0.2726	0.223329122	0.3926	0.4138	0.3960	0.4389	0.4499	0.3855

Table 6.4: Anomaly Detection (AD) accuracy comparison results (non-log-parsing-based)

	NeuralLog			Logsy			HEART		
Datasets	PR	RE	F1	PR	RE	F1	PR	RE	F1
HDFS	0.9998	0.9998	0.9998	0.9990	1.0000	0.9995	-	-	-
HDFS US	0.9783	0.8273	0.8912	0.9993	1.0000	0.9997	0.9948	0.5834	0.7352
Hadoop	NaN	0.0000	NaN	0.8577	1.0000	0.9234	0.9932	0.5743	0.7276
Hadoop US	NaN	0.0000	NaN	0.8923	0.8813	0.8645	0.9807	0.5883	0.7282
BGL	0.9972	0.9998	0.9985	0.9509	1.0000	0.9748	-	-	-
Spirit	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-
Thunderbird	0.9999	0.9999	0.9999	0.9999	1.0000	1.0000	-	-	-
Thunderbird US	0.9998	0.9926	0.9962	0.9999	1.0000	1.0000	1.0000	0.9989	0.9994
Average	0.9958	0.9699	0.9809	0.9624	0.9852	0.9702	0.9922	0.6862	0.7976
Std Deviation	0.0086	0.0699	0.0440	0.0535	0.0393	0.0471	0.0071	0.1806	0.1166

6.4.2 RQ2: How does the efficiency of anomaly detection (AD) compare between non-log-parsing and log-parsing-based techniques?

6.4.2.1 Methodology

To answer RQ2, we followed the same methodology used for RQ1; distinctly, we calculated the execution time of each anomaly detection technique on each dataset used. As mentioned in section 6.4.1.1, to account for the randomness of anomaly detection techniques, we repeated the training and testing process five times and provided the average execution time for each anomaly detection technique for each dataset used.

6.4.2.2 Results

Table 6.5 and Table 6.6 show the execution time of Drain and LogPPT, respectively, for each dataset (in seconds) as well as the execution time of anomaly detection techniques (in seconds) for log-parsing-based anomaly detection techniques. Specifically, under each anomaly detection technique column, there are four sub-columns: TR indicates the training time of the model, TST indicates the testing time, TOT represents the total time taken by the model for the training and testing phase, and T_{Drain} or T_{LogPPT} represents the model's total training and testing time, including the Drain or LogPPT execution time, as log-parsing execution time should be included for log-parsing-based anomaly detection techniques. Table 6.7 shows the execution time of non-log-parsing-based anomaly detection techniques; similar to the previous tables, under each anomaly detection technique column, there are three sub-columns: TR, TST, and TOT. Table 6.5 and Table 6.6 lack results for LogRobust on BGL, Spirit, Thunderbird, Thunderbird-US, as well as LogPPT on Spirit, respectively. The absence of these results is attributed to the same reasons explained in Section 6.4.1. Additionally, Table 6.7 does not include results for HEART on HDFS, BGL, Spirit and Thunderbird datasets due to an execution timeout.

If we look at the average execution time row, we can notice that the non-log-parsing-based anomaly detection techniques are the least efficient techniques with an average execution time of 10 579 seconds ($2\,\mathrm{h} : 56\,\mathrm{min} : 19\,\mathrm{s}$) for NeuralLog, 26 965 seconds ($7\,\mathrm{h} : 29\,\mathrm{min} : 25\,\mathrm{s}$) for Logsy, and 34 278 seconds ($9\,\mathrm{h} : 31\,\mathrm{min} : 18\,\mathrm{s}$). In contrast, all log-parsing-based anomaly detection techniques complete execution in less than 2088 seconds ($0\,\mathrm{h} : 34\,\mathrm{min} : 0\,\mathrm{s}$) on average, with LogAnomaly requiring the longest time. The slower performance of non-log-parsing-based anomaly detection techniques can be

Table 6.5: Anomaly Detection (AD) efficiency comparison results [Drain]

		Deeplog				LogAnomaly				LogRobust				CNN				PLELog			
Datasets	Drain Ex.	TR	TST	TOT	T_{Drain}	TR	TST	TOT	T_{Drain}	TR	TST	TOT	T_{Drain}	TR	TST	TOT	T_{Drain}	TR	TST	TOT	T_{Drain}
HDFS	1535	810	12	823	2358	431	5	436	1971	400	4	404	1939	175	1	176	1711	114	6	120	1655
HDFS US	1535	830	13	843	2378	74	5	78	1613	69	4	73	1608	1711	30	1	31	132	7	139	1674
Hadoop	20	2	0	2	22	23	1	24	44	34	1	35	55	10	0	10	30	0	0	0	20
Hadoop US	20	2	0	2	22	13	1	14	34	13	1	14	34	4	0	4	24	0	0	0	20
BGL	578	622	9	631	1209	1383	17	1400	1978	-	-	-	OOM	557	5	563	1141	319	21	340	918
Spirit	1538	833	25	858	2396	1996	52	2048	3586	-	-	-	OOM	812	17	828	2366	339	41	380	1918
Thunderbird	1587	1126	42	1168	2755	2391	86	2476	4063	-	-	-	OOM	1013	28	1042	2629	483	187	670	2257
Thunderbird US	1587	815	43	859	2445	1745	87	1833	3419	-	-	-	OOM	2629	524	27	551	201	197	398	1985
Average	1050	630	18	648	1698	1007	31	1038	2088	129	2	131	909	863	75	331	1060	198	57	255	1305
Std Deviation	720	411	17	424	1128	981	37	1015	1538	182	1	183	1007	917	181	421	1072	171	84	232	882

attributed to two main factors: (1) they leverage additional data, beyond log messages; for example NeuralLog uses verbosity and component-specific information, to facilitate the extraction of semantic information and (2) they are based on transformer models, which have a significantly larger number of parameters, requiring more computational resources and time for training. However, despite non-log-parsing-based techniques being inefficient, they can still be considered for real-time-based anomaly detection, if model training can be done in an offline manner because, as shown in Table 6.7, model testing does not take much time.

By looking at the under-sampled datasets, we can notice that the model training time decreases compared to the non-under-sampled datasets. For example, the model training time of NeuralLog is reduced from 14 818 seconds for HDFS dataset to 2824 seconds for HDFS-US, and from 22 005 seconds for the Thunderbird dataset to 410 seconds for Thunderbird-US respectively as shown in Table 6.7. This is because Neurallog is fast when undersampling reduces the training dataset size; this is also why NeuralLog is reported as fast in the corresponding paper [LZ21].

To conclude, the answer to RQ2 is that the non-log-parsing-based techniques are generally less efficient. This presents a trade-off when selecting an anomaly detection technique based on the desired balance between accuracy and efficiency. NeuralLog is recommended for scenarios where efficiency is not a critical factor, such as in quality control or environment monitoring, where batch processing or delayed analysis is acceptable. If NeuralLog's or Logsy's model training can be conducted offline, they could be suitable for real-time anomaly detection use cases.

6.4.3 Threats to Validity

Individual anomaly detection techniques have pre-defined hyper-parameters, which might affect the anomaly detection accuracy results. To mitigate this, we used the same hyper-parameter values proposed by the authors, when

Table 6.6: Anomaly Detection (AD) efficiency comparison results [LogPPT]

		Deeplog				LogAnomaly				LogRobust				CNN				PLELog			
Datasets	LogPPT Ex.	TR	TST	TOT	T_{LogPPT}	TR	TST	TOT	T_{LogPPT}	TR	TST	TOT	T_{LogPPT}	TR	TST	TOT	T_{LogPPT}	TR	TST	TOT	T_{LogPPT}
HDFS	5708	729	14	743	6451	248	2	250	5957	229	1	230	5938	84	0	84	5792	105	4	109	5817
HDFS US	5708	467	9	475	6183	47	2	49	5757	70	2	72	5780	27	1	28	5735	105	4	109	5816
Hadoop	234	2	0	2	236	54	1	55	289	39	1	40	274	24	1	25	259	0	0	0	234
Hadoop US	234	2	0	2	236	11	1	12	245	15	1	16	250	4	0	4	238	0	0	0	234
BGL	2902	572	9	581	3483	1372	17	1390	4292	-	-	-	OOM	1320	21	1340	4242	382	26	408	3310
Spirit	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Thunderbird	5971	362	2	365	6335	838	4	842	6813	-	-	-	OOM	271	1	272	6243	389	51	440	6411
Thunderbird US	5971	314	2	316	6286	759	5	764	6734	-	-	-	OOM	460	3	463	6434	91	53	144	6115
Average	3818	350	5	355	4173	476	5	480	4298	88	1	90	3061	313	4	317	4135	153	20	173	3991
Std Deviation	2675	274	5	279	2882	523	6	529	2876	96	1	96	3232	474	8	481	2746	165	24	180	2761

Table 6.7: Anomaly Detection (AD) efficiency comparison results (non-log-parsing-based)

	NeuralLog			Logsy			HEART		
Datasets	TR	TST	TOT	TR	TST	TOT	TR	TST	TOT
HDFS	14818	143	14961	40823	598	41422	-	-	-
HDFS US	2824	143	2967	38570	509	39079	76144	9801	85945
Hadoop	10	1	10	828	3	831	1579	99	1677
Hadoop US	9	1	9	637	3	640	1573	99	1672
BGL	20685	208	20893	44501	349	44850	-	-	-
Spirit	22786	203	22989	71149	429	71577	-	-	-
Thunderbird	22005	203	22208	8248	282	8529	-	-	-
Thunderbird US	410	186	595	8507	286	8793	38918	8901	47819
Average	10443	136	10579	26658	307	26965	29554	4725	34278
Std Deviation	10600	87	10663	25782	216	25945	35702	5354	40738

available; otherwise, we ran preliminary experiments and used the values that resulted in the same results reported in the corresponding papers.

Using a particular collection of log datasets poses a potential risk to the external validity of the study. We considered to include the logs of various systems, we selected HDFS, Hadoop, Spirit, Thunderbird, and BGL to make our study as comprehensive as possible. Even though the selected datasets have been widely used in existing literature [LZ22, WLK23] on log-based anomaly detection, they may not encompass a variety of log data characteristics. Further experiments with different datasets might be required to improve the generalizability of the provided results.

As mentioned in section 6.3.1, we used under-sampling that reduces the number of normal/abnormal instances in the datasets to make it balanced. However, the selection process of instances to be kept is random to remove any bias. This might be an internal threats-to-validity.

6.5 Findings and Implications

One of the interesting results from our evaluation is that the non-log-parsing based anomaly detection techniques accuracy is significantly higher than the log-parsing-based anomaly detection techniques, while there is a trade-off between anomaly detection accuracy and efficiency. NeuralLog is recommended if the efficiency is not important, such as quality control or environment monitoring, where batch processing or delayed analysis is acceptable. Additionally, if the NeuralLog's model training can be done offline then it will be considered both accurate and efficient at the same time, and hence can be used in real-time anomaly detection use case.

Though the objective of our study is not to see the impact of using different log-parsing techniques on the impact of accuracy or efficiency of anomaly detection techniques, we added two log-parsing techniques to provide insight. We found that the non-log-parsing-based anomaly detection technique is much slower than the log-parsing-based anomaly detection techniques regardless of log parsing results.

6.6 Related Work

To the best of our knowledge, very few empirical studies exist that compare the non-log-parsing-based and log-parsing-based anomaly detection techniques. Below we summarize the recent related studies.

Le and Zhang [LZ21], the authors of NeuralLog (non-log-parsing based anomaly detection technique), mainly focused on comparison with traditional machine learning models, for example, Support Vector Machine (SVM), Logistic Regression (LR), Invariant Mining (IM), and focused on two deep-learning based techniques, i.e., Log2Vec and LogRobust. On the contrary, we considered six deep-learning based anomaly detection techniques, namely DeepLog, LogAnomaly, LogRobust, PLELog, CNN and NeuralLog. Additionally, their evaluation is based on four publicly available datasets, HDFS, BGL, Thunderbird, and Spirit, while as mentioned in section 6.3.1, we evaluated the used techniques on eight datasets, making our study as comprehensive as possible. NeuralLog mainly focuses on (i) evaluating the effectiveness, i.e., can NeuralLog effectively work on publicly available log datasets, (ii) the ability of NeuralLog to capture the semantic meaning of log messages, and (iii) effectiveness of NeuralLog under different settings, for example by replacing the BERT model with Roberta [CKG$^+$19] and GPT2 [RWC$^+$19] model. We mainly focused on the accuracy and efficiency of anomaly detection techniques.

Table 6.8: Comparison with related empirical studies

Papers	Techniques	Datasets	Source Code
Le and Zhang. [LZ21]	Support Vector Machine (SVM), Logistic Regression (LR), Invariant Mining (IM), Log2Vec and LogRobust	HDFS, BGL, Thunderbird, Spirit,	Available
Chen and Liao [CL22]	PCA, IM, LogCluster, SVM, LR, LAnoBERT, UniLog, DeepLog, LogRobust, and NeuralLog	HDFS, BGL	Not Available
Tian et al. [TLWS23]	Support Vector Machine (SVM), LogAnomaly, DeepLog, HitAnomaly, CNN, NeuralLog, LogRobust	HDFS, BGL	Not Available
Egerdoerfer et al. [EZD22]	ClusterLog, DeepLog, and NeuralLog	HDFS, Lustre	Not Available
Our Work	DeepLog, LogAnomaly, LogRobust, PLELog, CNN, NeuralLog, Logsy, and HEART	HDFS, HDFS US, Hadoop, Hadoop US, BGL, Thunderbird, Thunderbird US, Spirit	Will be made available

Chen and Liao [CL22] recently proposed an anomaly detection approach called BERT-Log. It regards the log sequence as a natural language sequence and uses a pre-trained language model to learn the semantic representation of normal and anomalous logs. Their evaluation is based on a comparison of accuracy, different scales or sizes of datasets, and classification effect and did not evaluate an important factor such as efficiency. Additionally, as compared to traditional machine learning models, they utilized deep-learning-based models such as DeepLog, LogRobust, and NeuralLog, but they only used two datasets for the evaluation such as HDFS, and BGL. We did not use BERT-Log in our evaluation study because the authors did not make the source code available.

Tian et al. [TLWS23] proposed a technique called CLDTLog, which introduces contrastive learning and dual-objective tasks in a BERT pre-trained model and performs anomaly detection on system logs through a fully connected layer. Their approach does not require log parsing and thus can avoid the uncertainty caused by log parsing. They compared the accuracy of their technique with Support Vector Machine (SVM), LogAnomaly, DeepLog, HitAnomaly, CNN, NeuralLog, and LogRobust using only two datasets, i.e., HDFS and BGL, therefore the results can not be generalized due to very few data points. Additionally, they compared different factors such as the impact of log sequence length on CLDTLog using the BGL dataset, and the performance generalization of CLDTLog, and CLDTLog_uncl (CLDTLog without contrastive learning) using the BGL dataset. We did not use BERT-Log in our evaluation study because the authors did not make the source code available.

Egersdoerfer et al. [EZD22] proposed ClusterLog, a log pre-processing method that clusters the temporal sequence of log keys based on their semantic similarity. By grouping semantically and sentimentally similar logs, this approach aims to represent log sequences with the smallest amount of unique log keys, intending to improve the ability of a downstream sequence-based model to effectively learn the log patterns. They compared the accuracy of ClusterLog with DeepLog, and NeuralLog with different hyper-parameter values using only two datasets i.e., HDFS, and Lustre. They also compared their technique with DeepLog, and NeuralLog using different training dataset sizes. We attempted to utilize the source code provided for the ClusterLog [CE23], but unfortunately, we encountered a broken or non-working link when we accessed it on September 02, 2023.

Table 6.8 summarizes the key differences between the closely related studies and our work.

6.7 Summary

In this chapter, we provided a comprehensive empirical study comparing the accuracy and efficiency of non-log-parsing-based and log-parsing-based anomaly detection techniques. We used two log parsing techniques, eight datasets, and six deep-learning-based anomaly detection techniques. While analyzing the results for research questions, it became evident that though the non-log-parsing-based anomaly detection technique exhibited superior accuracy in the datasets we analyzed, its efficiency falls short, which could limit its practicality for real-time applications when such efficiency is required. This implies that the non-log-parsing-based anomaly detection technique should be considered where appropriate, for example, in use cases where accuracy is essential. It is worth noticing that NeuralLog's model training phase can be done offline and not frequently, making the non-log-parsing based technique useful, where the tradeoff can be diminished to some extent.

Chapter 7

Conclusions & Future Work

7.1 Summary of the contributions

Software logs play a crucial role in numerous software engineering tasks, such as model inference [WTD16, MPS17] and anomaly detection [NMA$^+$16, DLZS17], since logs are the sole source of data available that captures the run-time behavior of a software system. While log messages contain valuable run-time information, they cannot be directly processed by log-based analysis techniques such as log-based anomaly detection techniques to automatically detect if the logs contain any anomalous patterns that do not conform to the expected behavior of the system [HHC$^+$21]. *Log message template identification* aims to address the issue by decomposing log messages into fixed parts called message templates (templates, in short), characterizing the event types, and variable parts containing the parameter values of the events, which are determined at run time. Though different log-parsing accuracy metrics have been proposed in the literature to evaluate the log-parsing accuracy, it is necessary to use appropriate accuracy metric for fair comparison.

Additionally, log-parsing being preliminary step for log-based anomaly detection techniques, one can speculate the impact of log-parsing on log-based anomaly detection techniques. Thus, it is essential to investigate the impact through an empirical study and provide insightful implications. Moreover, recently, a technique called NeuralLog [LZ21] is proposed that does not use log-parsing as a pre-processing step, that rules-out the impact or log-parsing, so one can speculate that non-log-parsing-based technique might be more accurate than log-parsing-based anomaly detection techniques, and might be efficient as well. Hence, in depth analysis is required to validate the speculation.

In this thesis we have made the following contributions:

1. We assessed and compared different log parsing techniques and provided guidelines for evaluating the accuracy of log parsing techniques considering different use cases.

2. We proposed a theoretical framework for understanding the relationship between log parsing and anomaly detection, formally defining the concepts of *distinguishability* and *minimality* of *ideal* log parsing results.

3. We performed a comprehensive empirical study investigating the impact of log parsing on anomaly detection accuracy.

4. We performed a comprehensive empirical study comparing the accuracy and efficiency of log-parsed-based and non-log-parsing-based anomaly detection techniques.

The first contribution focuses on providing the guidelines for assessing the accuracy of log message template identification techniques and assessing the application of such guidelines through comprehensive evaluation using 14 existing log-parsing techniques and 16 datasets. More specifically, we defined a new accuracy metric called Template Accuracy (TA) and provided guidelines for using more appropriate metrics depending on the nature of the software engineering task. We also proposed a set of heuristic rules to correct the oracle templates, as based on manual analysis we found out that the provided oracle templates are not correct. Moreover, we also provided additional information about incorrectly identified templates the analysis of incorrect templates, that can provide insights to improve template identification techniques by highlighting the limitations of individual log-parsing techniques.

The second contribution focuses on the theoretical framework for defining and discussing what are ideal log parsing results for anomaly detection, more specifically we formalized the concepts of distinguishability and minimality, showing that log parsing results that minimally maintain distinguishability between normal and abnormal logs provide the best operating conditions for anomaly detection. We also performed a systematic and comprehensive evaluation of the impact of log parsing on anomaly detection. We showed that there is no strong correlation between log parsing accuracy and anomaly detection accuracy, hence increasing log parsing accuracy does not necessarily increase anomaly detection accuracy, regardless of the log-parsing accuracy metric. Additionally, the impact of the distinguishability of log parsing results on anomaly detection accuracy is significant for all anomaly detection techniques.

The third contribution involves guiding practitioners in selecting the most suitable anomaly detection technique for their specific use cases by comparing the anomaly detection accuracy between non-log-parsing-based and log-parsing-based techniques, and comparing anomaly detection efficiency between non-log-parsing-based and log-parsing-based techniques. We showed that the non-log-parsing-based anomaly detection technique is more accurate than the log-parsing-based anomaly detection techniques, as it is not impacted by the log-parsing errors generated by log-parsing techniques, and thus recommended. Additionally, the non-log-parsing-based anomaly detection technique is less efficient than log-parsing-based anomaly detection techniques. Depending on the use case, one might be interested in efficiency, such as real-time anomaly detection. This suggests that non-log-parsing-based anomaly detection is not suitable for such a task. One should also consider that if offline training can be done for non-log-parsing-based technique then it can be used for real-time anomaly detection use case, as the time-consuming phase is model training, not the model testing phase (a.k.a. model testing phase).

7.2 Future Research Directions

This dissertation sets the basis to follow different research directions in the future:

Refining the log-parsing accuracy metrics: In chapter 3, we concluded that the choice of log-parsing accuracy metrics matters based on the intended use case, and one should use the appropriate log-parsing accuracy metric for comparison. As part of future work, one can further refine the log-parsing accuracy metrics, for example by incorporating the concept of edit distance to compute the similarity between identified templates and their corresponding oracles, leveraging the idea proposed by Nedelkoski et al. [NBA$^+$20b]. Additionally extend the guidelines chapter by including more log-parsing techniques, for example LogPPT [LZ23].

Efficient Ideal Log Parsing for Experiments. We saw that having ideal log parsing results is important in controlled experiments to properly identify the cause of inaccurate anomaly detection results. However, getting ideal log parsing results for a given set of logs is not that simple since, for the logs containing n unique log messages, the number of all possible log parsing results (i.e., the number of all possible abstraction functions) is equal to the Bell number B_n (i.e., the number of all partitions of a set of size n) [Aig99]. Indeed, the problem of identifying the ideal log parsing results can be regarded as an optimization problem to minimize the amount of information contained in the log parsing results while maintaining distinguishability between normal and abnormal logs. Also, there is additional information that is potentially relevant to address this problem, such as the similarity between messages. Therefore, developing an efficient approach is an appealing research direction. We plan to extend the theoretical framework to measure the degree of distinguishability and minimality of given log parsing results and use such measures as fitness functions in meta-heuristic search algorithms [Luk13] to find the optimal log parsing results for anomaly detection.

Extension of empirical study: In chapter 5, we conducted an empirical study to investigate the influence of log parsing on the accuracy of anomaly detection. Our analysis revealed an absence of any significant correlation between log parsing accuracy and anomaly detection accuracy, irrespective of the log parsing metric employed, such as GA, PA, and FTA. We plan to extend our study on impact of log-parsing on deep-learning-based anomaly detection with more publicly available datasets and anomaly detection techniques to increase the generalizability of our results. We aim to include state-of-the-art few-shot anomaly detection techniques [HGJ$^+$22, PDSH21], which require only a limited amount of training data and could be more effective in practice. Additionally, given the high anomaly detection accuracy score obtained without using log parsing, integrating semantic information from templates (similar to non-log-parsing based anomaly detection techniques) alongside clustering techniques [CM20, XQW$^+$20] presents a promising avenue for enhancing the precision of log parsing in anomaly detection.We also plan to provide a more granular analysis of the distinguishability for log parsing results by defining a new metric that assesses the degree of distinguishability.

Bibliography

[AAAH22] Khawla Ali Abd Al-Hameed. Spearman's correlation coefficient in statistical analysis. *International Journal of Nonlinear Analysis and Applications*, 13(1):3249–3255, 2022. doi:10.22075/IJNAA.2022.6079.

[Aig99] Martin Aigner. A characterization of the bell numbers. *Discrete Mathematics*, 205(1):207–210, 1999. doi:10.1016/S0012-365X(99)00108-9.

[BHN11] Henrik Backlund, Anders Hedblom, and Niklas Neijman. A density-based spatial clustering of application with noise. *Data Mining TNM033*, 33:11–30, 2011. doi:10.54097/hset.v7i.1054.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 2009. doi:10.1145/1541880.1541882.

[CE23] Dong Dai Chris Egersdoerfer, Di Zhang. Clusterlog: Clustering logs for effective log-based anomaly detection, 2023. URL: https://github.com/DIR-LAB/ClusterLog.

[CKG$^+$19] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *Association for Computational Linguistics*, 58:8440–8451, 2019. doi:10.18653/v1/2020.acl-main.747.

[CL22] Song Chen and Hai Liao. Bert-log: Anomaly detection for system logs based on pre-trained language model. *Applied Artificial Intelligence*, 36(1):2145642, 2022. `doi:10.1080/08839514.2022.2145642`.

[CLG+21] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R Lyu. Experience report: deep learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908*, 2021. `doi:10.48550/arXiv.2107.05908`.

[CM20] Zhang Chunyong and Xiaojing Meng. Log parser with one-to-one markup. In *2020 3rd International Conference on Information and Computer Technologies (ICICT)*, pages 251–257, New York, NY, USA, 2020. IEEE, IEEE. `doi:10.1109/ICICT50521.2020.00045`.

[CTZ+21] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Conditional positional encodings for vision transformers. *arXiv preprint arXiv:2102.10882*, 2021.

[CvMBB14] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, 2014. ACL, Association for Computational Linguistics. `doi:10.3115/v1/W14-4012`.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DL16] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864, Los Alamitos, CA, USA, 2016. IEEE, IEEE. `doi:10.1109/CNSM.2015.7367331`.

[DLC+20] H. Dai, H. Li, C. S. Chen, W. Shang, and T. Chen. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering (TSE)*, 48(3):1–1, 2020. `doi:10.1109/TSE.2020.3007554`.

[DLZS17] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *2017 ACM Conference on Computer and Communications Security (SIGSAC)*, page 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3133956.3134015.

[Ely12] Alexander Elyasov. Log-based testing. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1591–1594, Los Alamitos, CA, USA, 2012. IEEE, IEEE. doi:10.1109/ICSE.2012.6227029.

[EZD22] Chris Egersdoerfer, Di Zhang, and Dong Dai. Clusterlog: Clustering logs for effective log-based anomaly detection. In *2022 IEEE/ACM 12th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*, pages 1–10, Los Alamitos, CA, USA, 2022. IEEE, IEEE. doi:10.1109/FTXS56515.2022.00006.

[FLWL09] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 IEEE international conference on data mining (ICDM)*, pages 149–158, Los Alamitos, CA, USA, 2009. IEEE, IEEE. doi:10.1109/ICDM.2009.60.

[FYX$^+$23] Ying Fu, Meng Yan, Zhou Xu, Xin Xia, Xiaohong Zhang, and Dan Yang. An empirical study of the impact of log parsers on the performance of log-based anomaly detection. *Empirical Software Engineering*, 28(1):1–39, 2023. doi:10.1007/s10664-022-10214-6.

[GdSTC23] Luiz Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. Bert-and tf-idf-based feature extraction for long-lived bug prediction in floss: a comparative study. *Information and Software Technology*, 160:107217, 2023. doi:10.1016/j.infsof.2023.107217.

[HDX$^+$16] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. Logmine: Fast pattern recognition for log analytics. In *25th ACM International on Conference on Information and Knowledge Management (CIKM)*, pages 1573–1582, New York, NY, USA, 2016. ACM, Association for Computing Machinery. doi:10.1145/2983323.2983358.

[HGJ+22] Chaoqin Huang, Haoyan Guan, Aofan Jiang, Ya Zhang, Michael Spratling, and Yan-Feng Wang. Registration based few-shot anomaly detection. In *European Conference on Computer Vision*, pages 303–319, New York, NY, USA, 2022. Springer, Springer. doi:10.1007/978-3-031-20053-3_18.

[HHC+21] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. A survey on automated log analysis for reliability engineering. *ACM Comput. Surv.*, 54(6), 2021. doi:10.1145/3460345.

[HZHL20] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Loghub: A large collection of system log datasets towards automated log analytics, 2020. URL: https://arxiv.org/pdf/2008.06448.pdf, arXiv:2008.06448.

[HZZL17] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40, Los Alamitos, CA, USA, 2017. IEEE, IEEE. doi:10.1109/ICWS.2017.13.

[JGB+16] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[JHFH08] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. Abstracting execution logs to execution events for enterprise applications. In *2008 The Eighth International Conference on Quality Software (QSIC)*, pages 181–186, Los Alamitos, CA, USA, 2008. IEEE, IEEE. doi:10.1109/QSIC.2008.50.

[JJSL20] Sooyong Jeong, Ajay Kumar Jha, Youngsul Shin, and Woo Jin Lee. A log-based testing approach for detecting faults caused by incorrect assumptions about the environment. *IEICE Transactions on Information and Systems*, 103(1):170–173, 2020.

[JM19] Daniel Jurafsky and James H Martin. Vector semantics and embeddings. *Speech and language processing*, pages 1–31, 2019.

[JYC+17] Tong Jia, Lin Yang, Pengfei Chen, Ying Li, Fanjing Meng, and Jingmin Xu. Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 447–455, Los Alamitos, CA, USA, 2017. IEEE, IEEE. doi:10.1109/CLOUD.2017.64.

[KSBB22a] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. Artifact for ”guidelines for assessing the accuracy of log message template identification techniques”, Jan 2022. URL: https://figshare.com/articles/software/Artifact_for_Guidelines_for_Assessing_the_Accuracy_of_Log_Message_Template_Identification_Techniques_/18858332/1, doi:10.6084/m9.figshare.18858332.

[KSBB22b] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)*, page 1095–1106, New York, NY, USA, 2022. ACM, ACM. doi:10.1145/3510003.3510101.

[KSBB23] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. Impact of log parsing on log-based anomaly detection, 2023. arXiv:2305.15897.

[Luk13] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[LWLW18] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. Detecting anomaly in big data system logs using convolutional neural network. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 151–158, Los Alamitos, CA, USA, 2018. IEEE, IEEE. doi:10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037.

[LXL+21] Zhongxin Liu, Xin Xia, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Which variables should i log? *IEEE Transactions on Software Engineering*, 47(9):2012–2031, 2021. doi:10.1109/TSE.2019.2941943.

[LYDH20] Xuanqing Liu, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. Learning to encode position for transformer with continuous dynamical model. In *International conference on machine learning*, pages 6327–6335, New York, NY, USA, 2020. PMLR, ACM. doi:10.48550/arXiv.2003.09229.

[LZ21] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 492–504, Los Alamitos, CA, USA, 2021. IEEE, IEEE. doi:10.1109/ASE51524.2021.9678773.

[LZ22] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection with deep learning: how far are we? In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 1356–1367, New York, NY, USA, 2022. IEEE, ACM. doi:10.1145/3510003.3510155.

[LZ23] Van-Hoang Le and Hongyu Zhang. Log parsing with prompt-based few-shot learning. *International Conference on Software Engineering (ICSE)*, pages 2438–2449, 2023. doi:10.1109/ICSE48619.2023.00204.

[LZL$^+$16] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 102–111, New York, NY, USA, 2016. ACM, ACM. doi:10.1145/2889160.2889232.

[MBJV23] Paul K Mvula, Paula Branco, Guy-Vincent Jourdan, and Herna L Viktor. Heart: Heterogeneous log anomaly detection using robust transformers. In *International Conference on Discovery Science*, pages 673–687, New York, NY, USA, 2023. Springer, Springer. doi:10.1007/978-3-031-45275-8_45.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[Miz13] Masayoshi Mizutani. Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing (SCC)*, pages 595–602, Los Alamitos, CA, USA, 2013. IEEE, IEEE. `doi:10.1109/SCC.2013.73`.

[MLZ+19] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, pages 4739–4745, New York, NY, USA, 2019. ACM, ACM. `doi:10.24963/ijcai.2019/658`.

[MPB+18] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 167–16710, New York, NY, USA, 2018. ACM, Association for Computing Machinery. `doi:10.1145/3196321.3196340`.

[MPS17] L. Mariani, M. Pezzè, and M. Santoro. Gk-tail+ an efficient approach to learn software models. *IEEE Transactions on Software Engineering (TSE)*, 43(8):715–738, 2017. `doi:10.1109/TSE.2016.2623623`.

[MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. URL: `https://dl.acm.org/doi/book/10.5555/3360093`.

[MW47] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. `doi:10.1080/01621459.1938.10502338`.

[MZHM09] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. Clustering event logs using iterative partitioning. In *15th ACM SIGKDD international conference on Knowledge discovery and data mining (SIGKDD)*, pages 1255–1264, New York, NY, USA, 2009. ACM, Association for Computing Machinery. `doi:10.1145/1557019.1557154`.

[NBA+20a] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. Self-attentive classification-based anomaly detection in unstructured logs. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1196–1201, New

York, NY, USA, 2020. IEEE, IEEE. doi:10.1109/ICDM50108.2020.00148.

[NBA+20b] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. Self-supervised log parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track - European Conference, ECML PKDD 2020, Proceedings, Part IV*, pages 122–138, New York, NY, USA, 2020. Springer, Springer. doi:10.1007/978-3-030-67667-4_8.

[NMA+16] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, and Subhrajit Bhattacharya. Anomaly detection using program control flow graph mining from execution logs. In *22nd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 215–224, New York, NY, USA, 2016. ACM, Association for Computing Machinery (ACM). doi:10.1145/2939672.2939712.

[NV10] Meiyappan Nagappan and Mladen 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)*, pages 114–117, Los Alamitos, CA, USA, 2010. IEEE, IEEE. doi:10.1109/MSR.2010.5463281.

[OS07] Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*, pages 575–584, Edinburgh, UK, 2007. IEEE, IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07). doi:10.1109/DSN.2007.103.

[PDSH21] Guansong Pang, Choubo Ding, Chunhua Shen, and Anton van den Hengel. Explainable deep few-shot anomaly detection with deviation networks. *arXiv preprint arXiv:2108.00462*, abs/2108.00462, August 2021. doi:10.48550/arXiv.2108.00462.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information process-*

ing & management, 24(5):513–523, 1988. doi:10.1016/0306-4573(88)90021-0.

[Shi16] Keiichi Shima. Length matters: Clustering system log messages using length of words, 2016. URL: https://arxiv.org/abs/1611.03213, arXiv:1611.03213.

[SKBB21] Donghwan Shin, Zanis Ali Khan, Domenico Bianculli, and Lionel Briand. A theoretical framework for understanding the relationship between log parsing and anomaly detection. In *International Conference on Runtime Verification (RV'21)*, pages 277–287, Cham, 2021. Springer, Springer. doi:10.1007/978-3-030-88494-9_16.

[SQ19] Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. *Advances in neural information processing systems*, 32:12058–12068, 2019.

[TLP11] Liang Tang, Tao Li, and Chang-Shing Perng. Logsig: Generating system events from raw textual logs. In *20th ACM international conference on Information and knowledge management (CIKM)*, pages 785–794, New York, NY, USA, 2011. ACM, ACM. doi:10.1145/2063576.2063690.

[TLWS23] Gaoqi Tian, Nurbol Luktarhan, Haojie Wu, and Zhaolei Shi. Cldtlog: System log anomaly detection method based on contrastive learning and dual objective tasks. *Sensors*, 23(11):5042, 2023. doi:10.3390/s23115042.

[Vaa03] Risto Vaarandi. A data clustering algorithm for mining patterns from event logs. In *3rd IEEE Workshop on IP Operations & Management (IPOM)*, pages 119–126, Los Alamitos, CA, USA, 2003. IEEE, IEEE. doi:10.1109/IPOM.2003.1251233.

[VP15] 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)*, pages 1–7, Los Alamitos, CA, USA, 2015. IEEE, IEEE. doi:10.1109/CNSM.2015.7367331.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[Wil92] Frank Wilcoxon. *Individual Comparisons by Ranking Methods*, pages 196–202. Springer New York, New York, NY, USA, 1992. doi:10.1007/978-1-4612-4380-9_16.

[WLK23] Xingfang Wu, Heng Li, and Foutse Khomh. On the effectiveness of log representation for log-based anomaly detection. *Empirical Software Engineering*, 28(6):39, 2023. doi:10.1007/s10664-023-10364-1.

[WTD16] Neil Walkinshaw, Ramsay Taylor, and John Derrick. Inferring extended finite state machine models from software executions. *Empirical Software Engineering*, 21(3):811–853, 2016. doi:10.1007/s10664-015-9367-7.

[XQW$^+$20] Tong Xiao, Zhe Quan, Zhi-Jie Wang, Kaiqi Zhao, and Xiangke Liao. Lpv: A log parser based on vectorization for offline and online log parsing. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1346–1351, New York, NY, USA, 2020. IEEE, IEEE. doi:10.1109/ICDM50108.2020.00175.

[YCW$^+$21] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1448–1460, Madrid, Spain, 2021. IEEE, IEEE. doi:10.1109/ICSE43902.2021.00130.

[YPH$^+$12] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 293–306, Hollywood, CA, 2012. ACM, USENIX Association. doi:10.5555/2387880.2387909.

[YZP$^+$12] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. Improving software diagnosability via log enhancement. *ACM Trans. Comput. Syst.*, 30(1), 2012. doi:10.1145/2110356.2110360.

[ZHL$^+$19] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*

(ICSE-SEIP), pages 121–130, Los Alamitos, CA, USA, 2019. IEEE, IEEE. doi:10.1109/ICSE-SEIP.2019.00021.

[ZL21] Hongyu Zhang and Van-Hoang Le. Source code for ”log-based anomaly detection without log parsing”, 2021. URL: https://github.com/LogIntelligence/NeuralLog.

[ZRL$^+$17] Xu Zhao, Kirk Rodrigues, Yu Luo, Michael Stumm, Ding Yuan, and Yuanyuan Zhou. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In *2017 26th Symposium on Operating Systems Principles (SOSP)*, page 565–581, New York, NY, USA, 2017. ACM, Association for Computing Machinery. doi:10.1145/3132747.3132778.

[ZXL$^+$19] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 807–817, Tallin, Estonia, 2019. ACM, The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). doi:10.1145/3338906.3338931.