Empirical Software Engineering (2023) 28:137
https://doi.org/10.1007/s10664-023-10364-1

# On the effectiveness of log representation for log-based anomaly detection

Xingfang Wu[1] · Heng Li[1] · Foutse Khomh[1]

Accepted: 27 June 2023 / Published online: 9 October 2023

**Abstract**
Logs are an essential source of information for people to understand the running status of a software system. Due to the evolving modern software architecture and maintenance methods, more research efforts have been devoted to automated log analysis. In particular, machine learning (ML) has been widely used in log analysis tasks. In ML-based log analysis tasks, converting textual log data into numerical feature vectors is a critical and indispensable step. However, the impact of using different log representation techniques on the performance of the downstream models is not clear, which limits researchers and practitioners' opportunities of choosing the optimal log representation techniques in their automated log analysis workflows. Therefore, this work investigates and compares the commonly adopted log representation techniques from previous log analysis research. Particularly, we select six log representation techniques and evaluate them with seven ML models and four public log datasets (i.e., HDFS, BGL, Spirit and Thunderbird) in the context of log-based anomaly detection.We also examine the impacts of the log parsing process and the different feature aggregation approaches when they are employed with log representation techniques. From the experiments, we provide some heuristic guidelines for future researchers and developers to follow when designing an automated log analysis workflow. We believe our comprehensive comparison of log representation techniques can help researchers and practitioners better understand the characteristics of different log representation techniques and provide them with guidance for selecting the most suitable ones for their ML-based log analysis workflow.

**Keywords** Log representation · Anomaly detection · Automated log analysis

Communicated by: Mika Mäntylä

✉ Xingfang Wu
xingfang.wu@polymtl.ca

Heng Li
heng.li@polymtl.ca

Foutse Khomh
foutse.khomh@polymtl.ca

[1] Department of Computer Engineering and Software Engineering, Polytechnique Montreal, Montreal, QC, Canada

## 1 Introduction

Logs are textual data generated by logging statements in the source code of software systems. Log data records important runtime information so that software practitioners can use it to understand the running state of a software system or diagnose a system failure. Traditionally, developers and operators manually examine logs or use rule-based approaches to search and analyze log data (Hansen and Atkins 1993; Prewett 2003; Rouillard 2004), which proves to be very inefficient and error-prone (Oliner et al. 2012). Modern software systems are large-scale, especially distributed systems that run on thousands of commodity machines, which usually generate large volumes of logs each day (Oliner and Stearley 2007; Schroeder and Gibson 2007). Logs are usually semi-structured and exhibit a mixture of formats and vocabularies, making the traditional manual or rule-based approaches tremendously challenging, if not infeasible (Dai et al. 2020; Zhu et al. 2019). Furthermore, structures and maintenance practices of modern software systems change rapidly, which poses new challenges for log analysis (Shang et al. 2014; Yuan et al. 2012). Automated log processing has drawn many software engineering researchers' interest in this context.

Prior studies have proposed various approaches that leverage information retrieval, natural language processing, traditional machine learning, and deep learning to support automated log analysis tasks (He et al. 2021). Automated log analysis approaches have been playing an important role in software maintenance and operation efforts (e.g., anomaly detection (Chen et al. 2021; Du et al. 2017; Fu et al. 2009; He et al. 2016b; Le and Zhang 2021; Lu et al. 2018; Meng et al. 2019; Nedelkoski et al. 2020; Wang et al. 2018; Xu et al. 2009; Zhang et al. 2019), failure diagnosis (Fu et al. 2013; Yuan et al. 2010), performance regression analysis (Chow et al. 2014; Liao et al. 2020; Nagaraj et al. 2012)). Many of these automated log analysis tasks leverage machine learning (ML) techniques. An indispensable step of ML-based log analysis is to transform the textual log data into numerical formats (e.g., feature vectors or digital sequences) that ML models can consume as features. We refer to this step as **log representation**: the process that transforms textual log data into numerical formats to be used as features in ML models.

Prior work uses different log representation techniques in their ML-based log analysis tasks (i.e., downstream tasks), including classical techniques (e.g., counting the occurrences of log templates or TF-IDF) and (deep) neural network based techniques (e.g., Word2Vec or FastText). For example, He et al. (2016b) match Message Count Vector representation with a logistic regression model to detect anomalies in log sequences. Zhang et al. (2019) leverages pre-trained FastText model to generate log template embeddings to construct their anomaly detection workflow. However, no work has focused on evaluating the effectiveness of these representations, thus the impact of using different log representation techniques on the performance of the downstream models is not clear. Although there are some ablation studies of automated log analysis to evaluate the effectiveness of their adopted representations for log data (Chen et al. 2021), researchers can hardly compare the studied log representation techniques with that of other works to know about the impacts that these techniques may have on the performance of downstream tasks. Therefore, our work aims to provide a comprehensive investigation of log representation techniques with the goal of providing a reference for future research on automated log analysis. We select six commonly used log representation techniques and evaluate them with seven ML models and four public log datasets in the context of log-based anomaly detection task. We select the context of log-based anomaly detection as it is the most widely studied topic of automated log analysis (Chen et al. 2021; Du et al. 2017; Fu et al. 2009; He et al. 2016b; Le and Zhang 2021; Lu et al. 2018; Meng et al. 2019; Nedelkoski et al. 2020; Wang et al. 2018; Xu et al. 2009; Zhang et al. 2019). Our findings are likely to be generalizable to other automated log analysis tasks, given the similarity among log representation techniques used in various downstream tasks (He et al. 2021). Therefore, the key factors we identified for selecting log representation techniques are expected to hold for other automated log analysis downstream tasks as well. We achieve our research objectives by answering the following research questions (RQs):

- RQ1: How effective are existing log representation techniques for automated log analysis?

This research question aims at making a fair comparison of the existing common log representation techniques. In this research question, we combine different log representation techniques with different anomaly detection models. By comparing and analyzing the performances across the combinations, we derive some observations for developers and researchers to help better choose log representation techniques when designing automated log analysis frameworks.

- RQ2: How does log parsing influence the effectiveness of log representations in automated log analysis?

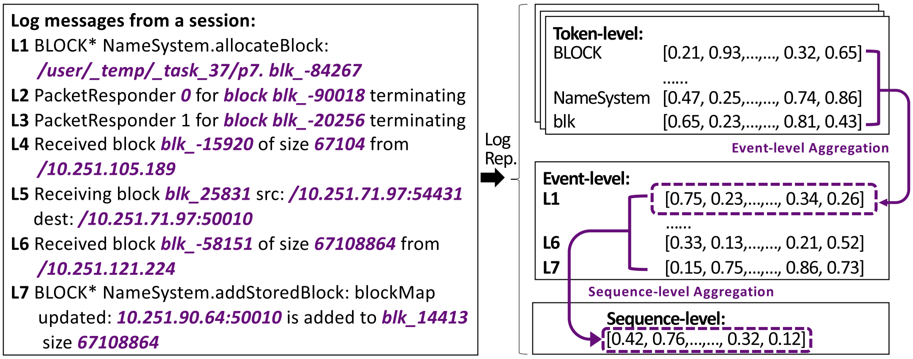
Log parsing is a common pre-processing step before the log representation step. It is not clear how log parsing and log representation together impact the performance of downstream tasks. Thus, in this RQ, we investigate the potential impacts that log parsing, when used with different log representation techniques, may have on the performance of downstream models. Findings confirm that the log parsing process has non-negligible impacts on the performance of the downstream models.

- RQ3: How do representation aggregation methods influence the effectiveness of log representation in automated log analysis?

Log representation techniques can generate the representation at different levels (e.g., token level or log event level). Sometimes, low-level representations need to be merged into high-level ones according to the need of the follow-up models. In this RQ, we aim to explore the potential influence of different aggregation configurations when used together with different log representation techniques. The findings indicate that the impacts of aggregation configurations may vary according to different factors, and the aggregation configurations may have non-negligible influences on the quality of log representations. Researchers should be careful when doing feature aggregation as there is no single best solution for all log data, representation techniques, and models.

Our work makes several important contributions:

1. We provide a comprehensive evaluation of the impact of log representation techniques on log-based anomaly detection task. Our results can be used as a guide for researchers and software practitioners in selecting the most suitable log representations for their anomaly detection frameworks or other log analysis workflows.



```
Log messages from a session:
L1 BLOCK* NameSystem.allocateBlock:
   /user/_temp/_task_37/p7. blk_-84267
L2 PacketResponder 0 for block blk_-90018 terminating
L3 PacketResponder 1 for block blk_-20256 terminating
L4 Received block blk_-15920 of size 67104 from
   /10.251.105.189
L5 Receiving block blk_25831 src: /10.251.71.97:54431
   dest: /10.251.71.97:50010
L6 Received block blk_-58151 of size 67108864 from
   /10.251.121.224
L7 BLOCK* NameSystem.addStoredBlock: blockMap
   updated: 10.251.90.64:50010 is added to blk_14413
   size 67108864
```

Log
Rep.
Token-level:
BLOCK [0.21, 0.93,...,..., 0.32, 0.65]
......
NameSystem [0.47, 0.25,...,..., 0.74, 0.86]
blk [0.65, 0.23,...,..., 0.81, 0.43]
Event-level Aggregation
Event-level:
L1 [0.75, 0.23,...,..., 0.34, 0.26]
......
L6 [0.33, 0.13,...,..., 0.21, 0.52]
L7 [0.15, 0.75,...,..., 0.86, 0.73]
Sequence-level Aggregation
Sequence-level:
[0.42, 0.76,...,..., 0.32, 0.12]

**Fig. 1** Different levels of abstraction of log representation

2. We provide an analysis of the impact of log parsing and feature aggregation approaches when they are used together with different log representation techniques. The insights obtained through this analysis can help optimize workflows of log analysis.
3. We share an implemented pipeline for log-based anomaly detection which supports convenient configurations of log parsing, different log representations, and different aggregation methods. Our implementation of the pipeline together the steps to replicate our main results are included in our replication package.[1]

**Organization** The remainder of this paper is organized as follows: We introduce the background of our work in Section 2. Section 3 surveys related works. Section 4 describes the design of our experiments, including the selection and overview of studied log representations, the downstream task and the datasets used. The evaluation metrics for the downstream task are also introduced. Section 5 is organized by the research questions we proposed. For each research question, we present the corresponding approach and results. Section 6 discusses our findings from the three research questions and summarizes the take-home messages. Section 7 identifies the threats to validity of our findings. At last, we summarize this paper in Section 8.

## 2 Background

### 2.1 Log Representations

Log representation is a process that converts textual log data into numerical feature vectors. Log representation techniques take semi-structured raw log data or parsed log data as input and generate representations at different abstraction levels. Figure 1 illustrates an example of different levels of representation for a log session. Different log representation techniques may work on different levels. Aggregation is the process that merges low-level

[1] Scripts and data files used in our research are available online and can be found in our replication package: https://github.com/mooselab/suppmaterial-LogRepForAnomalyDetection.

representation into high-level one. Based on these different levels of representations, various follow-up models can be designed to perform downstream tasks according to the needs. Different log analysis tasks may work on different levels of log representation according to their needs of information.

**Token-Level Representation** One piece of log message itself is a sequence of tokens. Tokens can be represented as embeddings with pre-trained language models (e.g., word2vec). We call this level of abstraction as **token-level** representation, which is the lowest level of log representation. Instead of directly feeding the token-level representation to the follow-up models to fulfill downstream tasks, representations of this level are usually aggregated into higherlevel ones, where the aggregation techniques are applied. However, there exist log anomaly detection methods that mainly work on token-level representations. For example, Logsy (Nedelkoski et al. 2020) tokenizes the preprocessed log messages and generates embeddings for tokens in log templates. Together with the positional encoding of the tokens, these representations are fed into a transformer-based structure. By training the neural network, the token-level representation is updated.

**Event-Level Representation** A log **event-level** embedding is a vector representation that encodes a single log message. This level of representation can be merged from token-level embeddings with different aggregation approaches (Meng et al. 2019). Besides, some language models can directly generate this level of representation directly (Devlin et al. 2018; Le and Zhang 2021). For example, Swisslog (Li et al. 2020) employs pre-trained BERT as a sentence encoder and directly generates sequence-level embeddings for log templates.

**Sequence-Level Representation** Usually, log data contains a sequence of log entries that can be sorted according to the chronological order indicated by timestamps. The whole log data can be grouped into a set of log sequences with different approaches (e.g., fixed windows, sliding windows, and session windows (Chen et al. 2021; He et al. 2016b; Le and Zhang 2022) according to the needs of downstream tasks.We call the embedding for this abstraction level as **sequence-level** representation. Most of the traditional ML models (e.g., SVM, decision tree) work on the representations of this level (He et al. 2016b). Sequence-level representation can be acquired by aggregating log event-level representations or by using sequential models (e.g., RNNs, Transformer).

## 2.2 Applications of Log Representations in Automated Log Analysis

When log data is represented as vectors or other structured data structures, various automated log analysis models can be built upon to realize various downstream tasks, such as anomaly detection (Chen et al. 2021; He et al. 2016b), performance modeling (Liao et al. 2020), predictive analysis (Katkar and Kasliwal 2014), or casual analysis (Jarry et al. 2021).

Anomaly detection is the most representative downstream task of log analysis (Chen et al. 2021; Du et al. 2017; Fu et al. 2009; He et al. 2016b; Le and Zhang 2021; Lu et al. 2018; Meng et al. 2019; Nedelkoski et al. 2020; Wang et al. 2018; Xu et al. 2009; Zhang et al. 2019). Log-based anomaly detection approaches identify anomalies inside a log sequence according to occurrence patterns of log events. With log representation, the anomaly detection task can be formulated as both unsupervised and supervised methods. Unsupervised methods adopt unsupervised machine learning algorithms (e.g., isolation forest) to mine the normal patterns of log data usually with the hypothesis that anomalies are unusual in the data sequence (Liu et al. 2012), while the supervised methods usually treat anomaly detection as a classification problem and employ classifiers (e.g., decision tree) to learn the normal and abnormal modes (He et al. 2016b).

Based on the availability of well-annotated datasets and the richness of related works, this work adopts the log-based anomaly detection task as our protocol to study the log representation. According to prior works (Chen et al. 2021; He et al. 2016b), supervised anomaly detection models usually achieve better performance and have better stability across datasets than their unsupervised counterparts. Also, the performance of unsupervised models is sensitive to their hyper-parameters. According to our experiments, unsupervised models favour different hyper-parameters when working on different datasets and need manually tuning. Otherwise, they may generate inferior results that will influence the comparisons of log representations. Based on these observations, we only focus on supervised models in this work to eliminate interference from these factors.

## 3 Related Work

In this section, we discussed existing log representation techniques and their applications in log analysis tasks. Generally, existing log representation techniques can be classified into two categories based on the mechanism to generate log representation: the classical approaches based on handcrafted features and semantic-based approaches. In addition, we discuss prior art on anomaly detection which is our focused downstream task in this work.

### 3.1 Classical Log Representation Techniques and Their Applications

There are several kinds of features manually designed by researches according to their domain knowledge to represent log data.

**Log Template ID** As log data is sequential and log messages are generated by a limited amount of logging statements, a log sequence can be easily presented as a sequence of log template id (a.k.a. log key) after being parsed with a log parser (Zhu et al. 2019). Although this approach ignore a lot of information from logs, it is an effective representation that reflects occurrence patterns of log templates in a log sequence. Log template ID is an event-level representation, which can be aggregated into Message Count by a count vectorizer.

Prior works use log template IDs to detect anomalies, as anomalies may be spotted out with abnormal occurrence pattern of log templates (Du et al. 2017; Lu et al. 2018). For example, Du et al. (2017) proposed the DeepLog anomaly detection framework, in which a sequential anomaly detection model is trained with log keys. Combined with another performance anomaly detection model, the framework achieved the state-of-the-art detection performance at the time it was proposed.

**Message Count** Unlike log template ID representation, Message Count (a.k.a. event count, log count, log message counter) Vector counts the occurrences of log templates in a log sequence and the length of representation depends on the amount of log templates in a whole log data, and thus is unrelated to the length of the log sequence. Message Count is a sequence-level representation.

It is one of the most common traditional log representation approach that adopted by various log analysis frameworks. For example, He et al. (2016b) use the log count as features and fed them a logistic regression model to detect anomalies. Xu et al. (2009) adopt the unsupervised dimension reduction method PCA with event count matrix to detect anomalies. Lou et al. (2010) input event count to invariant mining algorithm to detect anomalies.

**TF-IDF** Is a commonly used weighting technique in information retrieval and data mining. For log data, TF-IDF weighting can be either used to weight values in Message Count Vector or serve as a feature itself to present tokens in a log entry. For example, Wang et al. (2018) use the TF-IDF values of tokens in a log event to form the feature vectors. Some researchers modified TF-IDF to better suit the characteristics of log data. For example, Meng et al. (2021) apply the popular bag-of-words model to generate embedding and design the Inverse Location Frequency (ILF) method (a modified version of IDF(Salton and Buckley 1988) designed for logs) to weight the words of logs in feature construction. When TF-IDF operates on tokens within log events, it produces representations at the event level. Alternatively, when it processes the sequence of template IDs, it generates representations at the sequence level.

**Combined Features** Also, there are other works that try to combine different features and representations for log data. Liang et al. (2007) proposed a failure prediction model for log data generated from IBM Blue Gene/L. In this work, six groups of features are generated, including the number of events of different severity, event distribution, inter-failure times, and so on. These sequence-level representations are further processed by four classifiers(e.g., SVM, KNN) for later anomaly prediction.

### 3.2 Semantic-Based Log Representation Techniques and Their Applications

Unlike classical approaches, semantic-based approaches employ deep-learning techniques that do not rely on manually designed features. As logs are semistructured texts and log messages contains semantic information, some studies leveraged deep learning techniques in natural language processing and information retrieval to represent and analyze log data.

**Static Embedding** Some works are inspired by static word embeddings, which have been demonstrated to be more effective than log keys and log count. Static embedding techniques create embeddings for tokens in log events, resulting in token-level embeddings that can be further aggregated into higherlevel embeddings. Meng et al. (2019) proposed a log representation approach named Template2Vec. By embedding the log template with dLCE (Nguyen et al. 2016) to a vector, this approach presents the first step towards considering semantic and syntax information in log data.

The subsequent study proposed Logsy (Nedelkoski et al. 2020). In this work, two operations are applied to input tokens: token embedding and positional encoding. Before being embedded into vectors, log messages are split into word tokens and numerical characters and commonly used English words are removed. Then, these vectorized tokens are input into the subsequent encoder of the Transformer (Vaswani et al. 2017) module with multihead selfattention.

Zhang et al. (2019) proposed a log-based anomaly detection approach called LogRobust. They leverage Drain (He et al. 2017) to obtain log templates and encode log templates with pre-trained FastText model combined with TFIDF weight. Then, an attention-based Bi-LSTM model is used for anomaly detection. With semantic embeddings, It can identify unstable log events with similar semantic meaning.

**Contextual Embedding** Le and Zhang (2021) proposed NeuralLog, which does not rely on any log parsing. In NeuralLog, each log message is directly transformed into semantic vectors after removing numbers and special characters. A pre-trained BERT model is employed to encode log messages into a fixed dimension vector representation. Similar to static embeddings, contextual embeddings can operate at the token-level. Nonetheless, pre-trained models may also generate event-level embeddings using their unique structures, such as the pooler layer in BERT (Devlin et al. 2018).

### 3.3 Graph-Based Log Representation Techniques and Their Applications

Recently, a group of studies introduced Graph Neural Networks (GNNs) (Wan et al. 2021; Xie et al. 2022) to log anomaly detection. Unlike previouslymentioned approaches, which mainly utilize the sequential or quantitative patterns of log events in log sequences, GNN-based methods transform log sequences into graphs and leverage the spatial structural relationships among logs. Typically, these methods generate representations at the sequence-level by utilizing features from lower-levels. Some previously-mentioned representations can be incorporated into the graph structure as features of nodes and encoded by a GNN-based graph encoder. The experiments show that these approaches achieved promising results and robustness against the variation of window size. As the representation learning process is linked to downstream tasks, we are unable to incorporate this type of representation into our experimental framework. Therefore, it is not included in our experiments.

### 3.4 Anomaly Detection

As one of the most studied downstream tasks in the domain of automated log analysis, anomaly detection aims to detect abnormal system behaviours to help developers and operators uncover system issues and solve anomalies. Log data is a good source of information that can be utilized for anomaly detection models to evaluate the status of a system, as it may contain the indexes of the availability of system resources and the running status of services. The log sequence can also reflect the execution paths of a system. From these pieces of information, potential failures or unusual execution sequences can be spotted according to the regular pattern. Therefore, as a highly in-demand task of automated log analysis, log-based anomaly detection has been widely studied, and various approaches have been developed in the last decades.

Traditionally, developers may check system logs with keywords or use rules to find anomalies and locate the bugs in systems with their domain knowledge. Manual inspections are erroneous and unstable for large software systems that generate tons of logs in a short period. Rule-based approaches demand the manual construction of rules and can not adapt to fast-evolving software systems. Therefore, machine learning is adopted in many log-based anomaly detection approaches. In this study, we only focus on supervised learning methods, as we discussed in Section 2. Supervised anomaly detection is defined as a machine-learning task of deriving a classifier with annotated log sequence. The annotations mark the normal or anomalous states of log sequences or log events. Here, we list the most representative related works that utilize supervised learning methods in anomaly detection.

**Traditional Methods** Most of the traditional machine learning methods adopt message count vector as their log representation approach. A training instance for traditional models usually consists of an event count vector for a log sequence and its corresponding label. With training instances, classifiers can be trained to classify new instances. Logistic regression is a statistical model that is widely used in anomaly detection. It estimates the probability of normal and anomalous according to the input vector. The decision tree is a tree-based model that is constructed in a top-down manner with training data. Each node presents a split of an attribute with the criteria of information gain. The decision tree was also applied to log analysis in previous works (Chen et al. 2004). Event count vectors are used to construct the decision tree, and predictions for new instances are given with tree structure. Support Vector Machine (SVM) is a common supervised method for classification. A hyperplane is constructed by maximizing the distance between the hyperplane and the closest point(s) of different classes to separate instances in high-dimension space. SVM was employed to detect failures (Liang et al. 2007) with statistical features of occurrences of log events.

**Deep Learning Methods** Different from traditional methods, the input feature of deep learning methods for log anomaly detection varies greatly. The most basic model is based on Multi-layer Perception (MLP). MLP is a feedforward structure that maps a set of input vectors to a set of output vectors. MLP model serves as a baseline model for log-based anomaly detection in previous works (Lu et al. 2018). Convolutional Neural Networks (CNNs) were first adopted for log anomaly detection by Lu et al. (2018). This work uses convolutional layers containing different kernels to extract features from vectors generated with a codebook that maps the logs to embedding vectors. Long Short-Term Memory (LSTM) is commonly used for mining the patterns from log data in many automated log analysis frameworks (Du et al. 2017; Meng et al. 2019). However, the mechanisms of prior works vary: some works (Du et al. 2017) used log template ID as input, and LSTM is used to learn the occurrence patterns of log templates in normal and abnormal log sequences, while there is another line of works that take embedding vectors of log templates as input (Meng et al. 2019). Transformer-based models have been applied in the log-based anomaly detection task by some recent works (Le and Zhang 2021; Nedelkoski et al. 2020). The transformer blocks in these models can capture contextual information from input sequences with the self-attention mechanism. These models exhibit promising results in log-based anomaly detection tasks. However, the previous works utilized transformer-based models with different formulations of the log-based anomaly detection task. For example, Logsy (Nedelkoski et al. 2020) formulates the anomaly detection problem as discrimination between normal logs from the system of interest and auxiliary logs from other systems, in which anomalies are detected based on only their log messages and sequential information is ignored. The best practices for using transformers in log analysis are still unclear. Therefore, we do not include this sort of method in our experiments.

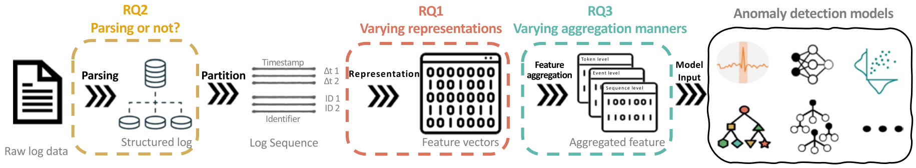


**Fig. 2** General workflow of our experiments. The variations for each research question are highlighted with dotted boxes

## 4 Experimental Design

In this section, we introduce the design of our experiments to evaluate the effectiveness of different log representation techniques by assessing their impact on the performance of selected automated log analysis tasks. We first give an overview of the workflow of the experiments. Then, we discuss log representation techniques studied in the experiments to generate feature vectors for follow-up downstream tasks. Then, our selected downstream tasks with their corresponding models are introduced. We also review the metrics that we use to evaluate the performances of each studied downstream tasks.

As our focus is on the impact that log representation techniques have on the performances of downstream tasks, we select the most representative automated log analysis task (i.e., anomaly detection) and combine different models of it with different representation techniques. Aside from the comparative evaluation of studied representation techniques on follow-up models, we also look into the log parsing and feature aggregation process that can affect the effectiveness of log representations.

### 4.1 Overview

Figure 2 shows the general workflow of our experiments. Raw log data is unstructured textual data. As most log representation techniques require structured log data as inputs, a log parsing step is often applied to obtain structured log data. In this work, the Drain parser (He et al. 2017) is adopted in our experimental workflow, as it is shown to have superior parsing performances on most of the datasets (Zhu et al. 2019). However, log parsing may not be needed for some log representation techniques. For example, log parsing process is deserted in NeuralLog (Le and Zhang 2021), a recent anomaly detection workflow that achieved results outperforming the other existing approaches. Then, log data is fed into representation algorithms to get numerical representation of different abstraction levels. According to the level of abstraction of the log representation, different models of downstream tasks are selected to mine critical information from the data according to the specific task and yield the analytical results.

### 4.2 Studied Log Representations

As our goal is to conduct an evaluation of different log representation techniques, we selected the most representative techniques for the study. We implemented the studied techniques following the common practices of previous works (Chen et al. 2021; He et al. 2016b) to better compare their characteristics and quality. However, different previous works have different implementations with minor alternations for some representation techniques. We adopt and synthesize these open-source codes or implement them ourselves from scratch. Existing log representation techniques can be classified into two categories: classical and semantic-based approaches.

**Classical Log Representation Techniques** For classical representation techniques, we select message count vector, template ID-based TF-IDF (TF-IDF ID) and text-based TF-IDF (TF-IDF Text) feature representation. The message count takes log template indexes as input. It presents a log sequence with a vector counting the event occurrences from each log template. The event template ID-based TF-IDF (TF-IDF ID) weighting weights each event template ID with their respective TF-IDF value. In template text-based TF-IDF (TF-IDF Text) representation, we used the TF-IDF values of tokens in the template of a log event to represent a log message. For a sequence, we calculate the average of feature vectors of its log events to form the representation for the sequence.

**Semantic-Based Log Representation Techniques** For semantic-based log representation techniques, we choose three commonly adopted techniques in existing automated log analysis frameworks as our objects of study:Word2Vec, FastText, and BERT. For each of them, we leverage the pre-trained models trained with natural language corpus as related works do (Le and Zhang 2021; Zhang et al. 2019). For Word2Vec, we use the word vectors generated by the model pre-trained with Google News dataset.[2] Pre-trained Word2Vec can generate many out-of-vocabulary (OOV) words when processing log data and is unable to handle them in a proper way. So, we assign the zero vector for OOVs when generatingWord2Vec representations for the studied datasets. For FastText, we leverage the off-the-shelf word vectors, which were pre-trained on Common Crawl Corpus and Wikipedia (Grave et al. 2018). FastText can handle OOV words by summing up embeddings for its component char-ngrams. Therefore, FastText is able to generate embeddings for OOV words in logs, although the embeddings may not be effective. For BERT (Devlin et al. 2018), we utilize the pre-trained base model (Turc et al. 2019). And the sentence embedding are generated by the second-to-last encoder layer of the model, which is 768 dimensions. The second-to-last hidden layer is chosen as the last layer is too closed to the target functions durning pre-training, which may contain biases.

## 4.3 Downstream Models and Datasets

### 4.3.1 Anomaly Detection Models and Implementations

We select 7 supervised machine learning anomaly detection models to evaluate the studied log representation techniques. SVM, decision tree, logistic regression, and random forest are traditional machine learning models. These models are commonly used and well-studied in various application scenarios and often serve as baseline in automated log analysis tasks (He et al. 2016b). For deeplearning models, we choose MLP, CNN, and LSTM models. The MLP model is selected as a baseline for log-based anomaly detection in prior work (Lu et al. 2018). CNN and LSTM are widely employed in many automated log analysis frameworks (Du et al. 2017; Lu et al. 2018; Meng et al. 2019).

[2] https://code.google.com/archive/p/word2vec/

We employ these well-studied machine learning models based on the fact that they are commonly adopted in anomaly detection workflows or other automated log analysis frameworks. By selecting these widely adopted models, we believe that the findings of our work may stand a better chance of generalizing to other log-related automated analysis tasks. We briefly introduce the implementation of the studied models in the following, while details can be found in our replication package[1].

**Traditional Models** For traditional anomaly detection models, we follow the implementations of Loglizer (He et al. 2016b). However, their implementations only take the event count matrix generated with session windows as input. In our case, the input dimensions vary according to the studied log representations. Same as Loglizer, all of our studied traditional models take sequence-level representation as input. We modify hyper-parameters of these models according to the input dimensions of our generated log representations.

**Multi-layer Perception (MLP)** We follow the similar implementation of the baseline model in (Lu et al. 2018), we treat the anomaly detection task as a binary classification problem and use a MLP with one hidden layer with 200 neurons as a binary classifier. The inputs are feature vectors of different log representation techniques, and the outputs are the one-hot encoding of the binary labels. Cross entropy loss is used as the criterion to train the three-layer network. MLP also takes sequence-level log representation as input.

**Convolutional Neural Network (CNN)** In our work, we implement exactly the same network structure as in the original work (Lu et al. 2018). However, instead of taking log keys as input and using the codebook to map log keys into embeddings, our network substitutes the codebook with a fully-connected layer with 50 neurons, which maintains the same embedding size as the original work for the convolutional layers to process. Network details can be find in our replication package. The CNN models require event-level log representation as input, and demand the input sequence are of same length. As the sessions of log data may contain different numbers of log messages, we sliced the sessions with a sliding window.

**Long Short-Term Memory (LSTM)** There are different mechanisms of using LSTM to detect anomalies in log sequence in prior works. As the aim is to compare different log representations, our implementation treats different log representations as the input feature of the LSTM model. Similar to CNN models, LSTM models require fixed length event-level representation as input. Network details can be found in our replication package[1].

### 4.3.2 Datasets and Preparations

Our experiments evaluate the existing representations with the following four public log datasets provided by LogHub (He et al. 2020):

- The HDFS dataset (Xu et al. 2009) is collected from the Amazon EC2 platform. It contains more than 11 million log events, and each event is associated with a block ID, by which we slice log data into a set of sessions, which are the sub-sequences of the entire log sequence. For each session, labels are given to indicate whether there exist anomalies. There are a total of 575,061 log sessions with 16,838 (2.9%) anomalies.

- The BGL dataset (Oliner and Stearley 2007) is recorded from the Blue Gene/L (BG/L) supercomputer system at Lawrence Livermore National Labs (LLNL) with a time span of 215 days. This dataset contains 4,747,963 annotated log messages, where 348,460 (7.3%) are labelled as failures. Unlike HDFS, log messages in BGL do not have identifiers for separating logs from different job executions, processes or threads. So, grouping techniques (e.g., time-based, fixed window-based, etc.) are adopted to form sub-sequences. For uniformity, we also call these sub-sequences in BGL as sessions.
- The Spirit dataset is also a well-used public log dataset (Oliner and Stearley 2007), which Sandia National Labs collected from their Spirit supercomputing system. There are more than 272 million log messages in total. As the whole dataset is too large for us to process, we use a subset containing the first 5 million log messages in our work, which follows the practice of prior work (Le and Zhang 2022). In the subset, 15.5% of the log messages are marked as anomalies. The subset is shared in the replication package.
- The Thunderbird dataset (Oliner and Stearley 2007) is also a public log dataset from Sandia National Labs. There are around 211 million log messages in totalWe followed the practices of previous works (Le and Zhang 2021, 2022) and extracted a continuous chunk of 10 million log messages from the whole dataset, among which 4.1% are labelled as anomalies. We also share the subset in our replication package.

According to the common practices (Chen et al. 2021; He et al. 2016b) of dataset preprocessing and grouping, we prepared the studied datasets with the following configurations:

**Preparation for the HDFS Dataset** For the HDFS dataset, as the available annotation labels are based on blocks ID, an identifier that marks the different execution sequences, we use it as the clue to group logs into sessions. We use 70% of the sessions as training set and the other 30% as test set by following the common practices of datasets splitting in supervised learning tasks (e.g., El-Sayed et al. (2017); Lyu et al. (2021)). During splitting, we shuffle the sessions while maintaining the time-based sequence of log messages inside each session (Chen et al. 2021). Recent work (Le and Zhang 2022; Lyu et al. 2021) suggests that the random shuffling process can cause data leaking problems. However, as the main focus of our work is the impact of log representation rather than the performance of the downstream models, the random shuffling process will not undermine our evaluation.

**Preparation for the BGL Dataset** For the BGL dataset, we do not have identifiers to separate the log items into different execution sequences. So, we choose to group the log messages according to the timestamp. We refer to the grouping approaches of prior papers that adopt the BGL dataset and group the log messages with a fixed window of 6 h (He et al. 2016b). After the time-based grouping, there are 718 sessions. As the number of sessions is far less than that of the HDFS dataset, we use 80% of the sessions as training set and 20% as test set instead of a 70%/30% splitting, following the practices in prior work (Chen et al. 2021; Le and Zhang 2021; Meng et al. 2019). Similarly, we shuffle the sessions while maintaining the time-based sequence within each session. The labels are merged from that of the log messages inside each session. If any of the log messages inside a session is labelled as an anomaly, the whole session is recognized as an anomaly, following the approach used in prior work (He et al. 2016b).

**Table 1** Grouping techniques and default window size settings for studied datasets

| Dataset | Grouping Criterion | # of sessions | | Ave. # of log per session | Window size | Stride |
|---|---|---|---|---|---|---|
| | | Train | Test | | | |
| HDFS | Session ID | 402,542 | 172,519 | 22 | 30 | 1 |
| BGL | Time (6h) | 575 | 143 | 6,565 | 50 | 50 |
| Spirit | Time (1h) | 938 | 235 | 4,208 | 50 | 50 |
| Thunderbird | Line (100l) | 79,773 | 19,944 | 100 | 30 | 10 |

**Preparation for the Spirit Dataset** Similar to the grouping configuration of the BGL dataset, we group the log messages according to their timestamps. However, we adopt a fixed window of one hour instead of six hours following the configuration in prior work (Le and Zhang 2022). After grouping, we get 1,173 sessions, with 221 anomaly samples. We further shuffle and partition the sessions into the training and test sets with an 80%/20% splitting.

**Preparation for the Thunderbird Dataset** Instead of adopting a one-hour fixed-window grouping, we employed a fix-length grouping to the Thunderbird dataset, as we noticed that the logs were unevenly distributed in time. If the sessions are grouped by a fixed-length time window, the number of logs in some sessions may be extremely large. We chose a window size of 100 lines, which is also a setting employed in experiments from previous work (Le and Zhang 2022). After grouping, we get 99,717 sessions in total, among which 33,526 sessions are anomalies. We performed a sequential split of the sessions using an 80%/20% ratio to obtain the training and test sets. This approach helped improve the generalizability of our findings and establish their validity across all data selection configurations.

**Window Size for Sequential Models** As CNN and LSTM models require inputs to be of consistent sequence lengths, we need to further slice each log session with fix-length sliding windows. According to the characteristics of each dataset and the common practices in other works (Chen et al. 2021), we select the configurations of the sliding window in Table 1 for the studied datasets as default settings. We further analyze the impacts of the variation of window size in RQ3.

**Log Parsing** Ideally, we would use a log parser that can convert the unstructured raw log data into structured log data without any error. In practice, however, existing log parsers cannot successfully parse all the log messages as the formats of log messages are usually diverse and complex. Continuous updates to existing parsing strategies and configurations are required due to new log templates and variations in log formats resulting from the evolution of software (Zhang et al. 2019).

In fact, the impact of using different log parsers for automated log analysis has been explored in prior work (He et al. 2016a). A recent work (Le and Zhang 2022) further investigated the impacts of data noise introduced by log parsing errors. The authors combined five anomaly detection models with four commonly-used log parsers and found that parsing errors induced by different parsers have distinctive impacts on downstream models. However, the patterns of the impacts remain to be explored.

As our goal is to achieve the best possible parsing results to serve as input for our following processes, we do not compare the impacts of using different log parsers which may lead to different parsing results. According to Zhu et al. (2019)'s benchmarking work for log parsers, Drain (He et al. 2017) is the most accurate parser among their studied log parsers, which attains the highest accuracy on 9 out of 16 datasets. Therefore, we choose Drain as our log parser to preprocess the raw log into structured data and extract parameters from log messages in our work. However, the Drain parser can still generate large number of inaccurate templates for our studied datasets when we follow the default configuration indicated in the paper of He et al. (2017). By examining the templates generated, it becomes apparent that certain parsing errors have occurred. For example, numerous log templates have been created with slight variations in certain fields that should be dynamic variables, but have instead been incorrectly identified as static text. To eliminate the impact of these inaccurate templates on our evaluation, we iteratively checked and appended the regular expressions designed for handling these undetected dynamic variables. We were able to decrease the number of resulting wrong templates. For example, after passing a set of regular expressions to the parser when parsing the Thunderbird dataset, the amount of log templates decreases from 2,241 to 1,488, in which many duplicate templates are removed. In a prior study (He et al. 2016a), a similar approach was utilized and it was verified that incorporating domain expertise (such as eliminating IP addresses) can enhance the precision of log parsing. The details of the regular expressions can be found in our replication package.

### 4.4 Evaluation Methods

Anomaly detection is formulated as a binary classification problem in our study. Therefore, we assess the performance of studied models using precision, recall and F1 score. We label the outcomes of these models as true positive (TP), false positive (FP), true negative (TN), and false negative (FN). Further, the precision, recall and F1 score are calculated as follows: $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, $F1 = \frac{2PrecisionRecall}{Precision+Recall} = \frac{2TP}{2TP+FP+FN}$. All the metrics are calculated on the test sets. For some of our results, we only report the $F1$ metric due to space limit. We report the the complete results of all the metrics in our replication package. For sequence-level representations, each sample represents a session, for which classifiers generate one prediction. We calculate the metrics based on predictions for sessions. However, for models that demand fixed-length input, we slice each session with sliding windows and get fixed-length sub-sequences. The labels for these sub-seqences are derived from the session they are from. And models generate predictions for each sliding window. We merge the predictions within sessions and use the labels for sessions to calculate the metrics.

## 5 Experimental Results

In this section, we present the results of our three research questions, aiming to understand the effectiveness of different log representation techniques in the context of anomaly detection, with the hope that our findings can be generalized to other similar automated log analysis tasks.

## 5.1 RQ1. How effective are existing log representation techniques for automated log analysis?

### 5.1.1 Motivation

Prior works have widely used different log presentation techniques in their automated log analysis workflow. However, no work has comprehensively compared the impact of the choice of log representation techniques in their workflow. Therefore, this research question aims to bridge the gap and provide a comprehensive comparison of the commonly used log representation techniques in the context of anomaly detection. Through analyzing the impact of different log representation techniques on the different anomaly detection models, we hope to provide a reference for future work to choose the appropriate log representation techniques for their specific data, analysis tasks, and use cases.

### 5.1.2 Approach

In this RQ, we evaluate our studied six log representation techniques with seven anomaly detection ML models and four datasets. For each log representation technique, we combine it with each ML model applied on each dataset.

**Combining Log Representations and ML Models** As our goal is to evaluate the effectiveness of the existing log representation techniques, we compare the performances of the models of selected downstream tasks with different inputs of representations generated with studied representation techniques.

Message count vector and event template ID-based TF-IDF (TF-IDF ID) are based on the count of log occurrences in log sequences and, thus, can only generate a sequence-level representation for each log sequence. As mentioned before, CNN and LSTM models require event-level log representation due to their mechanisms. Therefore, CNN and LSTM models can not be combined with these two representation techniques. Other representation techniques can generate token-level or event-level log representation. Moreover, low-level log representation can be merged with proper aggregation approaches to higherlevel representations. Therefore, these representation techniques can match anomaly detection models that demand both event-level and sequence-level input. For representation techniques that generate token-level representations, we aggregate token-level representation to form event-level representation for a log message. For models requiring sequence-level log representations, we further aggregate the event-level log representations into sequence-level with mean aggregation, which is the most common practice in previous works.

**Using Scott-Knott Effect Size Difference (SK-ESD) test to rank log representation techniques** To understand the relative rank of the different log representation techniques, we use the SK-EST test (Tantithamthavorn et al. 2017, 2018) to rank these techniques into statistically distinct groups based on their performances on studied datasets. We conduct three separate SK-EST tests: One for traditional models, one for deep learning models and a third one for all models.

Different datasets and different downstream models can significantly impact the resulting performance regardless of the chosen log representation techniques. To mitigate such impact when ranking the log representation techniques, we first derive a rank (i.e., initial rank) of each log representation techniques for each downstream model applied on each dataset based on the F1 score (i.e., a log representation technique achieving a better F1 score has a higher rank). Each initial rank of a log presentation technique for each model and each dataset serves as one observation for the log representation technique. As we have seven models and four datasets, each log representation technique has 28 observations in total in the overall test. Then, we use the SK-EST test to derive statistical ranking of the six log representation techniques based on their observations (i.e., initial ranks). The level of significance used in the SK-EST test is set to the default value of 0.05.

As CNN and LSTM models can not be combined with sequence-level representation techniques (i.e., MCV and TF-IDF by message ID), there are some observations in the tests (i.e., the MCV and TF-IDF (ID) techniques do not have observations for the CNN and LSTM models). Specifically, the statistical tests underlying the SK-EST tests would be performed with unequal sizes of samples, which may impact the power of the statistical significance (Rusticus and Lovato 2014). Thus, the missing observations may influence the ranking results. We mark the affected representation techniques in Table 6.

### 5.1.3 Results

Tables 2, 3, 4 and 5 compare the results of applying different log representation techniques to seven anomaly detection models on the four studied datasets. Table 6 shows the statistical rankings of the different log representation techniques from the SK-EST tests.

- **The choice of log representation techniques has non-negligible influences on the performance of the downstream models**. As shown in Table 2, nearly all models achieve very good performance on the HDFS dataset (with F-scores ranging from 0.938 to 0.999), the Spirit dataset (from 0.934 to 0.997), and the thunderbird dataset (from 0.933 to 1.000), while their performance on the BGL dataset is relatively lower (with F-scores ranging from 0.692 to 0.967). Nevertheless, we observe that different log representation techniques can lead to different performance of the downstream models. On the HDFS dataset, using different log representation techniques causes a Fscore differences up to 0.061 for the different models; on the BGL dataset, the different log representation techniques lead to F-score differences up to 0.275 for the different models; on the Spirit dataset, the largest discrepancy reach 0.063, which is 0.038 for the Thunderbird dataset.

Our SK-EST test results (Table 6) indicate that there exist statistical difference between the performance of the different log representation techniques. In the overall and traditional-model-only ranking, the six log representation techniques are ranked into five distinct groups. The three classical log representation techniques outperformed their semantic-based counterparts, with MCV achieving the best rank, followed by TF-IDF (ID) and TF-IDF (Text) in the second rank. The BERT embedding is ranked only in the third place, followed byWord2Vec and FastText. However, the BERT embedding is ranked first in the deep-model-only ranking, which shows that the deep-learning-based anomaly detection models can generally work better with BERT embedding than traditional models.

**Table 2** Evaluation of six log representation techniques applied to seven anomaly detection models on HDFS dataset

| Model | | | Classical | | | Semantic-based | | | Gap |
|---|---|---|---|---|---|---|---|---|---|
| | | | Message Count Vector | TF-IDF (ID) | TF-IDF (Text) | W2V | FastText | BERT | |
| Traditional models | SVM | P | 0.999 | 0.999 | 0.999 | 0.998 | 0.998 | 0.998 | 0.001 |
| | | R | 0.917 | 0.999 | 0.979 | 0.998 | 0.998 | 0.998 | 0.082 |
| | | F1 | 0.956 | **0.999** | 0.989 | 0.998 | 0.998 | 0.998 | 0.043 |
| | Decision Tree | P | 1.000 | 1.000 | 0.985 | 0.985 | 0.985 | 0.985 | 0.015 |
| | | R | 0.998 | 0.998 | 0.999 | 0.998 | 0.998 | 0.998 | 0.001 |
| | | F1 | **0.999** | **0.999** | 0.992 | 0.992 | 0.992 | 0.992 | 0.007 |
| | Logistic Regression | P | 1.000 | 0.999 | 1.000 | 0.999 | 1.000 | 0.999 | 0.001 |
| | | R | 0.996 | 0.997 | 0.900 | 0.901 | 0.884 | 0.999 | 0.115 |
| | | F1 | 0.998 | 0.998 | 0.947 | 0.948 | 0.938 | **0.999** | 0.061 |
| | Random Forest | P | 0.998 | 0.999 | 0.997 | 0.999 | 0.999 | 0.998 | 0.002 |
| | | R | 1.000 | 1.000 | 0.999 | 0.985 | 0.985 | 1.000 | 0.015 |
| | | F1 | **0.999** | **0.999** | 0.998 | 0.992 | 0.992 | **0.999** | 0.007 |
| Deep-learning models | MLP | P | 0.999 | 0.911 | 0.987 | 0.911 | 0.911 | 0.911 | 0.088 |
| | | R | 0.999 | 1.000 | 0.999 | 0.999 | 1.000 | 0.999 | 0.001 |
| | | F1 | **0.999** | 0.953 | 0.993 | 0.953 | 0.954 | 0.953 | 0.046 |
| | CNN | P | - | - | 0.982 | 0.985 | 0.990 | 0.992 | 0.010 |
| | | R | | | 0.922 | 0.923 | 0.922 | 0.921 | 0.002 |
| | | F1 | | | 0.951 | 0.953 | **0.955** | **0.955** | 0.004 |
| | LSTM | P | - | - | 0.991 | 0.997 | 0.993 | 0.998 | 0.007 |
| | | R | | | 0.922 | 0.921 | 0.920 | 0.923 | 0.003 |
| | | F1 | | | 0.955 | 0.958 | 0.955 | **0.959** | 0.004 |

[1] For each model, the highest F1-Score achieved by the representation techniques are highlighted

[2] The’Gap’ columns shows the biggest differences between the representation techniques for the dataset

The difference between the performance of different log representation techniques may be explained by the different information represented by different representation techniques. For example, one can directly tell the number of occurrences of certain log events in a sequence from the message count vector. Sometimes, this may be the most critical indicator of an anomaly and lead to a good anomaly detection performance.

- **There exists no single log representation technique that performs the best across all models and datasets.** As shown in Table 2, 3, 4 and 5, five out of the six log representation techniques (except Word2Vec) achieve the best performance for at least one combination of models and datasets. The best-performing log presentation technique in the overall ranking, Message Count Vector, achieves the best performance for 12 out of the 20 (i.e., 3/5) combinations of models and datasets. However, the technique in the last place (i.e., FastText) achieves the best for only one case out of the 28 combinations.

The findings suggest that researchers and practitioners should be cautious with the selection of log representation techniques and investigate the mechanism of their models and the characteristics of log representation techniques. Based on the knowledge, they can choose the representations that suit their follow-up models best.

**Table 3** Evaluation of six log representation techniques applied to seven anomaly detection models on BGL dataset

| Model | | | Classical | | | Semantic-based | | | Gap |
|---|---|---|---|---|---|---|---|---|---|
| | | | Message Count Vector | TF-IDF (ID) | TF-IDF (Text) | W2V | FastText | BERT | |
| Traditional models | SVM | P | 0.958 | 0.828 | 0.855 | 0.853 | 0.869 | 0.871 | 0.130 |
| | | R | 0.840 | 0.654 | 0.728 | 0.716 | 0.654 | 0.667 | 0.186 |
| | | F1 | **0.895** | 0.731 | 0.787 | 0.779 | 0.746 | 0.746 | 0.164 |
| | Decision Tree | P | 0.959 | 0.959 | 0.971 | 0.781 | 0.734 | 0.812 | 0.237 |
| | | R | 0.921 | 0.919 | 0.963 | 0.701 | 0.654 | 0.701 | 0.309 |
| | | F1 | 0.939 | 0.938 | **0.967** | 0.739 | 0.692 | 0.752 | 0.275 |
| | Logistic Regression | P | 0.947 | 0.882 | 0.868 | 0.871 | 0.844 | 0.886 | 0.103 |
| | | R | 0.889 | 0.741 | 0.728 | 0.753 | 0.667 | 0.765 | 0.222 |
| | | F1 | **0.917** | 0.805 | 0.792 | 0.808 | 0.745 | 0.821 | 0.172 |
| | Random Forest | P | 0.830 | 0.810 | 0.872 | 0.667 | 0.681 | 0.694 | 0.205 |
| | | R | 0.963 | 0.951 | 0.946 | 0.783 | 0.808 | 0.806 | 0.180 |
| | | F1 | 0.891 | 0.875 | **0.907** | 0.720 | 0.738 | 0.745 | 0.170 |
| Deep-learning models | MLP | P | 0.958 | 0.951 | 0.927 | 0.895 | 0.868 | 0.910 | 0.090 |
| | | R | 0.840 | 0.951 | 0.938 | 0.840 | 0.815 | 0.877 | 0.136 |
| | | F1 | 0.895 | **0.951** | 0.933 | 0.866 | 0.841 | 0.893 | 0.119 |
| | CNN | P | - | - | 0.900 | 0.868 | 0.857 | 0.939 | 0.082 |
| | | R | | | 1.000 | 0.975 | 0.963 | 0.951 | 0.049 |
| | | F1 | | | **0.947** | 0.919 | 0.907 | 0.945 | 0.040 |
| | LSTM | P | - | - | 0.866 | 0.755 | 0.822 | 0.871 | 0.116 |
| | | R | | | 0.877 | 0.988 | 0.914 | 1.000 | 0.123 |
| | | F1 | | | 0.871 | 0.856 | 0.865 | **0.931** | 0.075 |

[1] For each model, the highest F1-Score achieved by the representation techniques are highlighted

[2] The’Gap’ columns shows the biggest differences between the representation techniques for the dataset

*Finding 1*: The choice of log representation techniques has a non-negligible influence on the performance of the downstream models. While there is no single log representation technique that always performs the best, overall, the simplest message count vector representation performs the best across various models and datasets.

- **Traditional models generally perform better with classical log representation techniques, while deep learning models are able to work well with semantic-based representation.** For traditional anomaly detection models, 3 out of 4 models achieve the best performance with classical representations on the HDFS dataset and 4 out of 4 on the BGL, Spirit and Thunderbird datasets. In total, in 15 out of the 16 cases (four models and four datasets) of traditional machine learning models, classical log representation techniques perform better. The three classical log representation techniques are listed in the first two statistically distinct groups, which outperformed all their semantic-based counterparts with traditional machine learning models. However, the results are different for deep anomaly detection models that can leverage the sequential information (i.e., CNN and LSTM): 7 out of 8 cases favour semantic-based embeddings rather than the classical counterpart (i.e., TF-IDF (Text)). But for the MLP, which takes sequence-level representation as input,

**Table 4** Evaluation of six log representation techniques applied to seven anomaly detection models on Spirit dataset

| Model | | | Classical | | | Semantic-based | | | Gap |
|---|---|---|---|---|---|---|---|---|---|
| | | | Message Count Vector | TF-IDF (ID) | TF-IDF (Text) | W2V | FastText | BERT | |
| Traditional models | SVM | P | 0.984 | 0.978 | 0.984 | 0.952 | 0.973 | 0.973 | 0.011 |
| | | R | 0.968 | 0.963 | 0.963 | 0.963 | 0.952 | 0.968 | 0.016 |
| | | F1 | **0.976** | 0.970 | 0.973 | 0.957 | 0.962 | 0.971 | 0.019 |
| | Decision Tree | P | 1.000 | 1.000 | 1.000 | 0.952 | 0.962 | 0.942 | 0.058 |
| | | R | 0.995 | 0.995 | 0.995 | 0.947 | 0.907 | 0.952 | 0.088 |
| | | F1 | **0.997** | **0.997** | **0.997** | 0.949 | 0.934 | 0.947 | 0.063 |
| | Logistic Regression | P | 0.989 | 0.989 | 0.994 | 0.989 | 0.988 | 0.984 | 0.010 |
| | | R | 0.968 | 0.947 | 0.925 | 0.947 | 0.914 | 0.957 | 0.054 |
| | | F1 | **0.978** | 0.967 | 0.925 | 0.967 | 0.950 | 0.970 | 0.028 |
| | Random Forest | P | 0.984 | 0.979 | 0.982 | 0.941 | 0.945 | 0.939 | 0.045 |
| | | R | 0.984 | 0.984 | 0.994 | 0.954 | 0.957 | 0.958 | 0.040 |
| | | F1 | 0.984 | 0.981 | **0.988** | 0.947 | 0.951 | 0.948 | 0.041 |
| Deep-learning models | MLP | P | 0.984 | 0.989 | 0.989 | 0.978 | 0.968 | 0.978 | 0.021 |
| | | R | 0.957 | 0.952 | 0.957 | 0.963 | 0.968 | 0.973 | 0.021 |
| | | F1 | 0.970 | 0.970 | 0.973 | 0.970 | 0.968 | **0.976** | 0.008 |
| | CNN | P | - | - | 0.944 | 0.959 | 0.935 | 0.974 | 0.039 |
| | | R | | | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 |
| | | F1 | | | 0.971 | 0.979 | 0.966 | **0.987** | 0.021 |
| | LSTM | P | - | - | 0.940 | 0.943 | 0.929 | 0.944 | 0.015 |
| | | R | | | 1.000 | 0.979 | 0.984 | 1.000 | 0.021 |
| | | F1 | | | 0.969 | 0.961 | 0.956 | **0.971** | 0.015 |

[1] For each model, the highest F1-Score achieved by the representation techniques are highlighted

[2] The'Gap' columns shows the biggest differences between the representation techniques for the dataset

favours (in 3 out of 4 cases) quantitative count-based representations according to our experimental results.

The performance difference may be caused by the models' discrepancy in the ability to learn the complex and abstract representation of the log data. The classical representation techniques are based on quantitative or sequential statistics to occurrences of log templates or tokens, whose patterns are relatively simpler than that of semantic embeddings. The semantic-based representations carry higher layer information, which may be utilized by more advanced deep models. The authors of a recent study (Le and Zhang 2021) utilized a transformer-based model and demonstrated the superiority of semantic embedding over traditional representation. Their comparative experiment indicated that their model performed significantly better with BERT embedding than with the indexes of the log template on some datasets, which confirms our observation.

Moreover, event-level log representations are fed directly to the CNN and the LSTM models without the feature aggregation process that transforms event-level features to sequence-level features, which enable them to leverage the sequential information within a log session. This extra information may further boost the performance of these two models. Another explanation is that the dimensions of traditional representation

**Table 5** Evaluation of six log representation techniques applied to seven anomaly detection models on Thunderbird dataset

| Model | | | Classical | | | Semantic-based | | | Gap |
|---|---|---|---|---|---|---|---|---|---|
| | | | Message Count Vector | TF-IDF (ID) | TF-IDF (Text) | W2V | FastText | BERT | |
| Traditional models | SVM | P | 0.999 | 0.996 | 0.997 | 0.996 | 0.992 | 0.995 | 0.007 |
| | | R | 1.000 | 0.999 | 1.000 | 0.992 | 0.977 | 0.983 | 0.023 |
| | | F1 | **0.999** | 0.997 | 0.998 | 0.993 | 0.985 | 0.989 | 0.014 |
| | Decision Tree | P | 1.000 | 1.000 | 1.000 | 0.985 | 0.980 | 0.975 | 0.025 |
| | | R | 1.000 | 1.000 | 1.000 | 0.972 | 0.952 | 0.963 | 0.048 |
| | | F1 | **1.000** | **1.000** | **1.000** | 0.979 | 0.966 | 0.969 | 0.034 |
| | Logistic Regression | P | 0.999 | 0.998 | 0.997 | 0.996 | 0.996 | 0.995 | 0.004 |
| | | R | 0.999 | 0.981 | 0.987 | 0.980 | 0.934 | 0.977 | 0.065 |
| | | F1 | **0.999** | 0.989 | 0.992 | 0.988 | 0.964 | 0.986 | 0.035 |
| | Random Forest | P | 0.997 | 0.999 | 0.998 | 0.972 | 0.958 | 0.966 | 0.041 |
| | | R | 0.999 | 0.999 | 0.999 | 0.993 | 0.987 | 0.994 | 0.016 |
| | | F1 | 0.998 | **0.999** | 0.998 | 0.982 | 0.972 | 0.980 | 0.027 |
| Deep-learning models | MLP | P | 0.998 | 0.995 | 0.995 | 0.995 | 0.972 | 0.989 | 0.026 |
| | | R | 0.999 | 0.997 | 0.998 | 0.992 | 0.981 | 0.992 | 0.018 |
| | | F1 | **0.999** | 0.996 | 0.997 | 0.993 | 0.977 | 0.991 | 0.022 |
| | CNN | P | - | - | 0.977 | 0.962 | 0.955 | 0.986 | 0.031 |
| | | R | | | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 |
| | | F1 | | | 0.989 | 0.980 | 0.977 | **0.993** | 0.016 |
| | LSTM | P | - | - | 0.878 | 0.910 | 0.875 | 0.948 | 0.073 |
| | | R | | | 1.000 | 1.000 | 1.000 | 1.000 | 0.000 |
| | | F1 | | | 0.935 | 0.953 | 0.933 | **0.973** | 0.038 |

[1] For each model, the highest F1-Score achieved by the representation techniques are highlighted

[2] The'Gap' columns shows the biggest differences between the representation techniques for the dataset

techniques are usually determined by the vocabulary size or the number of log templates, which are not enormous in some datasets. Traditional models may perform well enough on the lowdimension data. However, deep learning models have more model parameters, which enable the extraction and representation of higher-dimension data, and therefore they can take advantage of the semantic information.

**Table 6** Statistical ranking of the different log representation techniques from the SK-EST test

| Model | Statistically Distinct Groups | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| Traditional only | MCV | TF-IDF (Text)<br>TF-IDF (ID) | BERT | Word2Vec | FastText |
| Deep only | MCV *<br>BERT | TF-IDF (Text) | TF-IDF (ID) * | Word2Vec | FastText |
| Overall | MCV * | TF-IDF (ID) *<br>TF-IDF (Text) | BERT | Word2Vec | FastText |

[*] The ranking of indicated techniques may be influenced by missing observations

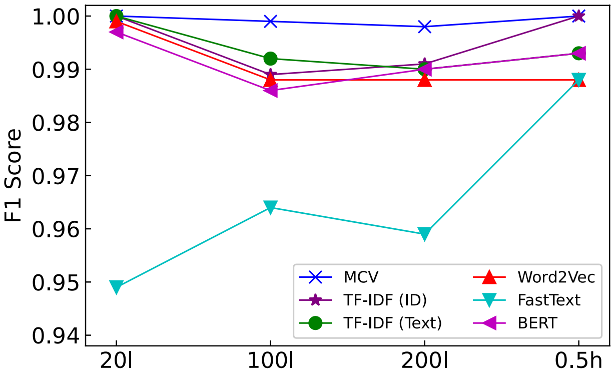


**Fig. 3** Results of logistic regression model with different grouping settings on Thunderbird dataset

- Among the classical log representation techniques, the simplest Message Count Vector technique achieves the best performance; among the semantic-based log representation techniques, the contextual embedding technique BERT achieves the best performance. The Message Count Vector technique is the simplest approach and is widely used in automated log analysis tasks (He et al. 2016b). Our results show that it achieves the best performance for 12 out of the 20 cases of the models (five models applied to four datasets) that do not leverage the sequential information of log messages. This is also confirmed by the ranking generated by the SK-EST test: In the traditional-only ranking, MCV is ranked in the first place, followed by TF-IDF-based techniques. The BERT is a contextual embedding. It achieves the best performance for 7 out of the 8 cases of the models that can leverage sequential information of log (two models applied to four datasets). In the deep-only ranking, BERT is ranked in the first group, which is superior to the other two semantic-based techniques by a large margin. Unlike static embedding, BERT, as a contextual embedding technique, generates representations based on the surrounding context and, thus, is more able to capture the semantic information of a log message. Therefore, representations generated with BERT can achieve good performance with most anomaly detection models.

Therefore, future work can leverage such general rules to choose the appropriate log representation techniques for their models. For traditional models that have limited feature extraction ability, classical representation techniques such as Message Count Vector could be considered. For more sophisticated models with more parameters, semantic-based representation techniques could be considered. And among the semantic-based representations, contextual embeddings may work better than static embeddings.

**Impact of Different Grouping Settings** Impact of Different Grouping Settings. Different log sequence lengths resulting from different grouping settings could impact the representations and the performance of models, which was shown in experiments from previous works (e.g., RQ2 in Le and Zhang (2022)). When grouping the studied datasets, we adopted different grouping settings, hoping that our findings could be tenable across varying settings. In particular, we follow prior works using the same datasets to config the group settings. To further examine the impacts that variations of the grouping process may have, we conduct an additional evaluation, in which we group the Thunderbird dataset with different fixed window settings (i.e., 20 logs, 100 logs, 200 logs, and 0.5-h logs), which are in accordance with the settings in Le and Zhang (2022). Figure 3 shows the results.

Based on the obtained results, it is evident that performance variations can occur due to different grouping configurations. In general, the relative ranking among the log representation techniques maintains the same for the different lengths of log sequences. There may be multiple factors contributing to these performance variations, making it challenging to accurately evaluate the effectiveness of log representation techniques across various grouping settings. For example, discrepancies in dataset size can impact performance variations since the composition and size of the training and test sets are influenced by different grouping configurations. Furthermore, there is no best-performing setting for all the studied techniques according to the results. As a result, we believe that the lengths of log sequences intricately influence both log representations and models through complex mechanisms. Future evaluations should focus on investigating the effects of grouping settings on log representation techniques.

> *Finding 2*: Traditional anomaly detection models perform well on classical log representations. However, deep models can achieve better performance with semantic-based representations by their stronger feature extraction and representation ability; Among the classical log representation techniques, Message Count Vector achieves the best performance. Context embedding (BERT) generally performs better among the semantic-based log representation techniques.

## 5.2 RQ2. How does log parsing influence the effectiveness of log representations in automated log analysis?

### 5.2.1 Motivation

Log parsing process transforms semi-structured raw logs into structured data by separating variables from log messages and retaining the log templates. Log parsing is a common pre-processing step before the log representation step. Although many log parsers with different mechanisms have been developed and achieved high performance and high accuracy (Zhu et al. 2019), the errors introduced by the parsing process may sometimes undermine the performances of log analysis according to the empirical study of Le et al. (Le and Zhang 2021). Essential words may be removed from a parsing error which results in information loss. As log parsing may be error-prone and cause information loss, some researchers have explored some log analysis frameworks (Le and Zhang 2021) that take row logs as inputs. It is not clear how log parsing and log representation together impact the performance of downstream tasks. Thus, in this RQ, we investigate the potential impacts that log parsing, when used with different log representation techniques, may have on the performance of downstream models.

### 5.2.2 Approach

In this RQ, we consider the log representation techniques that are compatible with both parsed and unparsed log data. Then we compare the performance of the downstream models that take the representations built from parserd and unparsed log data.

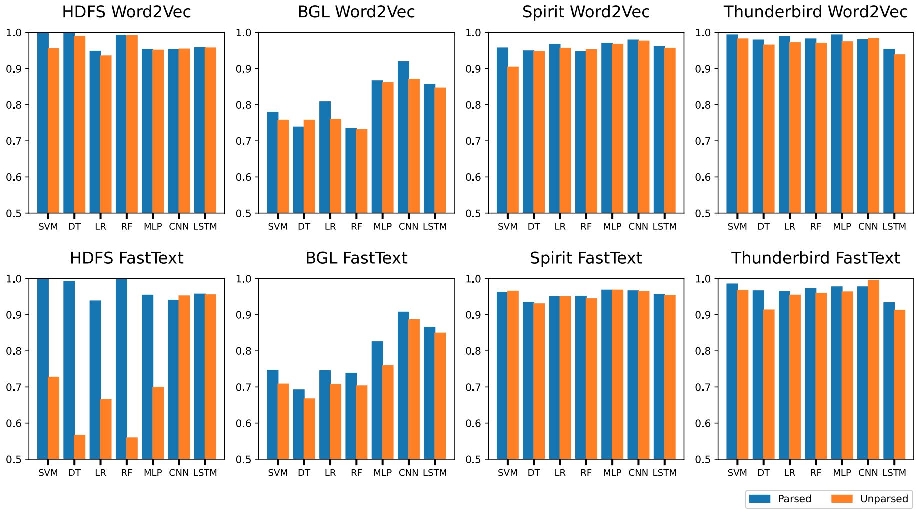


**Fig. 4** Comparison of performances of the studied anomaly detection models using the Word2Vec and FastText representations that are generated from parsed and unparsed logs

**Selection of Log Representation Techniques** From the studied log representation techniques, we select Word2Vec and FastText to answer this research question. The representation generated by these two techniques can remain the same regardless of the configuration of log parsing, which is not the case for representation like Log Template Text-based TF-IDF (TF-IDF Text), whose dimension may vary according to the vocabulary of the corpus in the dataset. As the dimension can also impact on the performance of models, we choose techniques that can generate fixed dimension representations for both parsed and unparsed log data. Also, the high dimension and enormous model size of pre-trained BERT prohibits us to generate features for unparsed logs in our server.

**Comparison of using parsed and unparsed log data to build representations** We compare and analyze the performances of the studied anomaly detection models with the features generated by these two representation techniques with both parsed and unparsed logs. Moreover, we also generate the visualization of embeddings with a dimension reduction algorithm (t-SNE (Van der Maaten and Hinton 2008)) to get some intuitions from the data to better explain the varied results.

### 5.2.3 Results

Figure 4 shows the comparison of performances of studied models with FastText representations generated with original and parsed log messages.

- In general, log parsing improves the quality of the generated log representations and thereby the performance of the downstream models. For the HDFS dataset, the two log representation techniques,Word2Vec and FastText, achieve an average performance (F1-score) improvements of 0.010 and 0.236 across the seven models, respectively. For the BGL dataset, the average improvements are 0.017 and 0.034. For Spirit, the improvements are 0.010 and 0.002, which are 0.012 and 0.016 for Thunderbird.

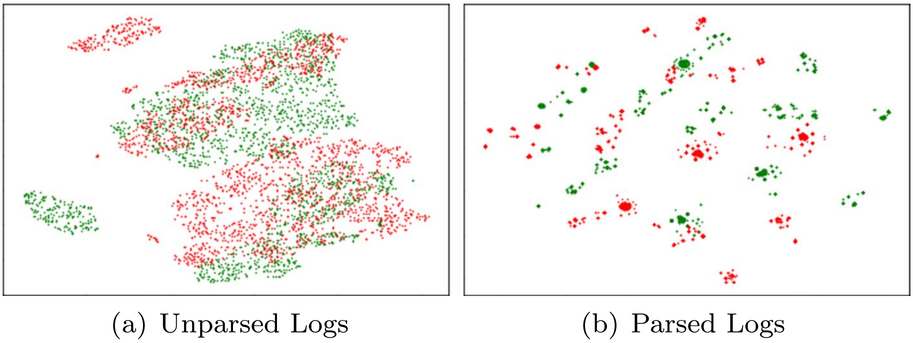

(a) Unparsed Logs (b) Parsed Logs

**Fig. 5** Visualization of representations generated with FastText using t-SNE. 200 positive (red) and negative (green) samples are randomly sampled from the HDFS dataset

In particular, for the HDFS dataset with the FastText representation, parsing leads to a very large difference in the performance of five out of the seven studied models. We randomly sample the FastText representations of 200 positive and negative samples from the HDFS dataset and use t-SNE (Van der Maaten and Hinton 2008) to visualize them. The visualization in Fig. 5 shows that representations for parsed logs are more compact than those of unparsed logs, which means the embeddings generated with the parsed logs are more distinguishable than those generated from the original log messages. However, deep learning models may work better with unparsed log data in some cases. For example, on HDFS and Thunderbird datasets, CNN performs better with unparsed logs by a small margin. The reason behind this may be that the parsing errors induced by the log parser can undermine the performance. The impact of log parsing errors was also examined in Le and Zhang (2021)'s work.

The characteristic of the representation technique can explain the general inferior performances on unparsed logs: There is no proper mechanism to represent numerical values or special tokens in logs for these representation techniques. The representations generated for these tokens would be a noise in feature representation if they are not treated as OOVs.

- **Depending on the datasets, some models (e.g., CNN and LSTM) are less sensitive to whether the log data is parsed or not.** CNN and LSTM perform similarly with the two different inputs may be a little counterintuitive. One possible explanation is that these two deep sequential models have strong feature extraction and representation ability and can offset the impacts of the noise. At the same time, unparsed logs will not introduce noises caused by the parsing errors.

Although there exist log analysis frameworks that take unparsed logs as input, to our best knowledge, they adopt preprocessing process to manually remove parameters or other fields from raw logs (Le and Zhang 2021), which can be regarded as a'vanilla' parsing process. If certain fields in logs, such as numerical values, special tokens, and error codes, are not adequately preprocessed, modelled, and utilized, it may have an adverse effect on the representation of the log. This finding implies that careful preprocessing and modelling of these fields are crucial for optimal log representation. Log parsing is an effective way to remove these unrecognizable texts for pre-trained language models and thus reduce the noise in representations. Future researchers and practitioners should pay attention to

**Table 7** Performance of logistic regression model on Thunderbird dataset parsed without extra regular expressions. The values under the F1 Scores indicate differences compared with corresponding results in RQ1

| Technique | Message Count Vector | TF-IDF (ID) | TF-IDF (Text) | W2V | FastText | BERT |
|---|---|---|---|---|---|---|
| F1 Score | 0.999<br>(=) | 0.989<br>(=) | 0.991<br>(0.001↓) | 0.987<br>(0.001↓) | 0.964<br>(=) | 0.983<br>(0.003↓) |

the preprocessing process before adopting log representation techniques, even if they abandon the log parsing process when designing a log analysis framework. Additionally, to improve the overall performance, future researchers and practitioners may also want to take into account the modelling of certain fields (e.g., component name, CPU usage, the time elapsed for a certain process, etc.) that cannot be embedded by language models but are critical to their downstream tasks (Du et al. 2017).

**Impact of Refining the Parsing Results** As mentioned in 4.3.2, we utilized additional regular expressions to improve the parsing results. We then did a sensitivity test to see its potential impact on the performance.We further evaluated our previously studied representation techniques using the Thunderbird dataset parsed by the Drain parser with regular expressions and trained logistic regression models. The results are shown in Table 7.

Although we can tell that the parsing results are refined by the observation that repetitive templates are decreased, we only observed minor accuracy gains for some representations after passing the regular expressions from the experiment. In contrast to the previous findings, which demonstrated that parsed and unparsed logs could lead to significant discrepancies, the refinement of parsing outcomes did not have a substantial impact on performance. This could be attributed to the ability of machine learning models to learn how to exclude unimportant features or irrelevant noise.

However, a large number of error templates may greatly increase the dimension of some representation techniques (e.g., for MCV, the dimension is equal to the number of resulting templates.). A large number of error templates increases the learning burden when we train the follow-up models. Sometimes it may even make the model training unprocurable.

**Impact of Using Different Log Parsers** Recent studies (Dai et al. 2020; Khan et al. 2022; Liu et al. 2022) adopt new metrics to evaluate the existing log parsers. Apart from just reporting the Group Accuracy of the parsing results, these works report other metrics (e.g., Parsing Accuracy, Edit Distance and etc.), which may give a more comprehensive evaluation of a parser. While it is true that the Drain parser achieves a high group accuracy, it presents inferior results in some metrics (i.e., Parsing Accuracy) in some recent works.

A higher Group Accuracy may benefit the representation techniques that rely on log templates (e.g., MCV). In contrast, a high Message-Level accuracy may contribute to the quality of representations based on the token-level census or embedding generation (e.g., TF-IDF (Text)). Therefore, we conducted another sensitivity test to reduce our evaluation's potential bias. In this test, we adopt the LogPPT parser (Le and Zhang 2023), which exhibits superior results over different metrics, including Parsing

**Table 8** Performance of logistic regression model on Thunderbird dataset parsed by LogPPT parser. The values under the F1 Scores indicate differences compared with corresponding results in RQ1

| Technique | Message Count Vector | TF-IDF (ID) | TF-IDF (Text) | W2V | FastText | BERT |
|---|---|---|---|---|---|---|
| F1 Score | 0.999<br>(=) | 0.996<br>(0.007 ↑) | 0.992<br>(=) | 0.981<br>(0.007↓) | 0.941<br>(0.023 ↓) | 0.985<br>(0.001↓) |

Accuracy, to further evaluate the quality of the studied representation techniques using the Thunderbird dataset and trained logistic regression models. The results are shown in Table 8.

From the results of this sensitive test, it is evident that there exist slight variations in performance when parsing the dataset using a different parser. The exploration of correlations or patterns between the performances of log parsers and the quality of log representation techniques is yet to be conducted. This presents an avenue for future evaluations in the realm of log parsers, representation techniques and downstream models.

> *Finding 3*: In general, log parsing improves the quality of the generated log representations and, thereby, the performance of the anomaly detection models. It reduces the noise in representations and thus alleviates models' learning burden by removing dynamic fields in logs. Proper preprocessing and modelling of these dynamic fields may be crucial for optimal log representation.

## 5.3 RQ3. How do representation aggregation methods influence the effectiveness of log representation in automated log analysis?

### 5.3.1 Motivation

For representation techniques that generate word embeddings(e.g., Word2Vec, FastText) for tokens in log events, we need to merge these token-level representations to event-level ones. The related works usually used mean aggregation to form the representation for log events (Meng et al. 2019), in which information may be lost, as some keywords that carry essential semantic information and severity in logs may be diluted by averaging. Therefore, we aim to compare different aggregation methods and evaluate the impacts they have on the quality of log representation.

In addition, for sequential models that take a fixed length of log messages as input, the event-level representation will be implicitly aggregated by the models to generate analytical results according to its task. So, we need to partition the session with a fixed-length window and a pre-defined step size. This implicit aggregation may also influence the performances of downstream models. Therefore, we investigate the impact that different configurations of session partition may have on the performance of downstream tasks. Although results generated with window-based inputs will be merged to generate the final predictions for log sessions, we want to quantify the impacts of different configurations of aggregation on the downstream tasks.

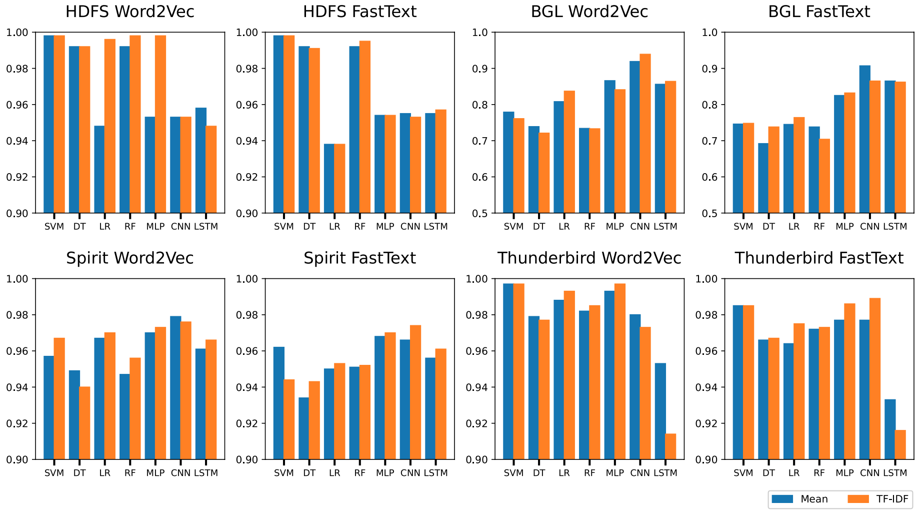


**Fig. 6** Comparison of performances of FastText Log representation with TF-IDF and Mean aggregation with LSTM anomaly detection model

### 5.3.2 Approach

In this research question, we investigate the impact of feature aggregation in log representation from two perspectives: 1) method of aggregating token-level representations, and 2) aggregation window size of sequential models.

Method of aggregating token-level representations. For the first perspective, we select the two most common aggregation practices, the **mean aggregation** and the **TF-IDF aggregation** (Chen et al. 2021). For mean average aggregation, we aggregate the token-level representations by averaging the feature vectors by each dimension. While for TF-IDF aggregation, we calculate the TF-IDF values for each token in log templates and calculate the weighted average of the token-level representation to form the event-level representation for log events. Moreover, we use the mean average to aggregate them into sequence-level representation. We select Word2Vec and FastText as they generate token-level representations.

**Window size for feature aggregation in sequential models** For the second perspective, we study the implicit aggregation process within the sequential models. From Table 1, we can find the significant difference in the average size of sessions in the four studied datasets. So, we conducted some preliminary experiments to broadly define the suitable range of the window size and further pre-defined some specific window sizes accordingly to investigate the impacts of implicit aggregation of studied sequential models. The chosen window size range for the HDFS and the Thunderbird is between 10 to 50, while for the BGL and the Spirit, whose sessions are usually longer, the range is 20 to 80.We adopt all studied techniques that can generate event-level representations (i.e., except the Message Count Vector and TF-IDF (ID) which can only generate sequence-level representations).

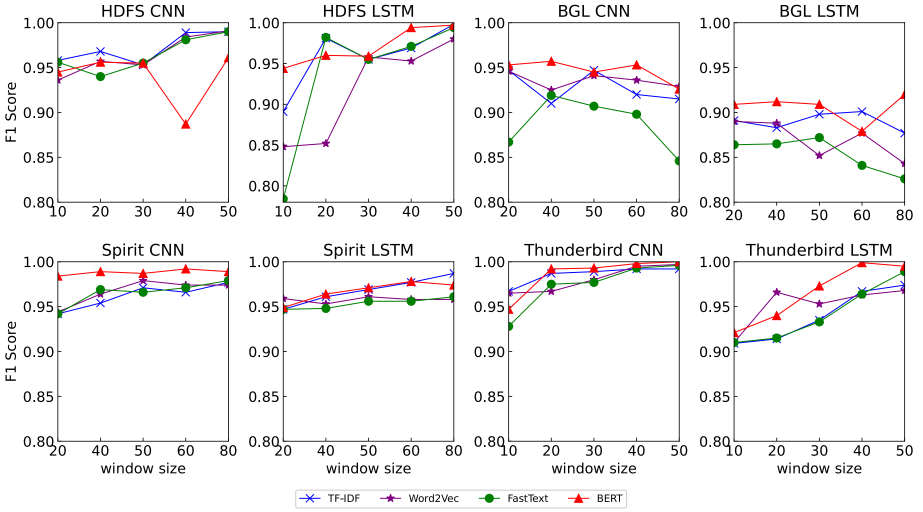


**Fig. 7** The impacts of different window sizes for feature aggregation

### 5.3.3 Results

- **Different approaches of aggregating representations can cause nonnegligible difference in the performance of the downstream models.** From Fig. 6, we can tell that there exist some performance gaps between these two aggregation methods on some combinations of the dataset, model and log representation. For example, the logistic regression model may favour TF-IDF aggregation with both representation techniques on all studied datasets: TF-IDF aggregation outperformed the mean aggregation in all eight cases. However, this conclusion is invalid on other representation techniques and follow-up models. This finding indicates that the aggregation approaches can have nonnegligible impacts on the effectiveness of log representation and thus influence the performance of log analysis models.
- **However, there is no clear pattern on which aggregation method performs better, as the impacts to the performance of aggregation method vary according to the combination of dataset, model, and representation techniques.** From the results, we can not find a clue to tell which aggregation method works better: For Word2Vec representation on the HDFS dataset, 3 out of 7 models perform significantly better with TF-IDF aggregation. However, this is not the case for other dataset and representation combinations. Different combinations of the dataset, model and representation favour different aggregation methods. Moreover, the difference in performance also vary among the combinations. Some combinations may be more sensitive to the utilization of aggregation techniques.
- **Window size for sequential models can significantly affect the performances of downstream models.** Figure 7 shows the F1 scores of CNN and LSTM models with different representations varying according to the input window size on four studied datasets. From the graph 7, we can see the fluctuation of the F1 score with the variation in window size. On the HDFS dataset, the biggest difference in the F1 score for the CNN model is 0.074, achieved by BERT, which is 0.210, achieved by FastText for the LSTM model. On the BGL dataset, the biggest difference in F1 score is 0.073, achieved

by FastText and 0.047, achieved by Word2Vec. On the Spirit dataset, the biggest difference in F1 score for the CNN model is 0.037, achieved by FastText, and 0.04 for the LSTM model, achieved by TF-IDF. And on the Thunderbird dataset, the biggest gaps for CNN and LSTM are 0.068 and 0.079, respectively, both achieved by FastText. The results show that the window size for feature aggregation can pose nonnegligible impacts on the performance of anomaly detection task.

- **The differences in performance may be caused by the intrinsic features of datasets.** The line charts show an improvement in performance when window size increases for the HDFS on almost all the studied log representation techniques with two models. We do not expand the range's upper bound for HDFS as the F1 score almost reaches 1, and the window size of 50 is larger than the length of most sessions in the dataset. For the BGL dataset, the peak is generally around 50, and the performances tend to decrease thereafter. For Spirit and Thunderbird, we also observed growth in performance when increasing the window size, while the variations are more stable compared with the other two datasets.
- **For the same dataset, window size affects the different representation techniques in a similar way.** The variation trends for different combinations of log representations and models are generally consistent on the same dataset, with some outliers (e.g., BERT with CNN when window size is 40, Word2Vec with LSTM when window size is 20). Therefore, we believe that the intrinsic features of the dataset cause the differences in performance.

More specifically, the characteristics of anomalies in a log sequence vary according to datasets. The lengths of abnormal sequences may have different ranges in different datasets. The sliding window setting can influence the distribution of anomalies in models' input windows, and some continuous anomaly log sequences may be truncated into multiple input windows in some input windows. Therefore, the sliding window setting may have a significant impact on the performance. Similarly to this, the aforementioned grouping methods, which group a log sequence into sessions, can also have impacts on the performance of different anomaly detection models, which were found by recent work (Le and Zhang 2022). In their work, their finding suggests that the performances of models suffer when dealing with shorter log sequences.

The impacts of sliding window settings may vary mainly depending on the datasets. It is a great challenge for the developer to determine the most suitable sliding window setting for their cases, as it may demand onerous experiments. Besides, we notice that recent studies introduce Graph Neural Networks (GNN) (Wan et al. 2021; Xie et al. 2022) to log representation, and the experiments from these works show that these models are robust against the variation of window size. Future researchers may utilize more stable log representation techniques, which are less sensitive to the variation of feature aggregation settings, to ensure more stable performances of their models.

*Finding 4*: Different aggregation configurations can cause non-negligible differences in the performance of the follow-up models, while there is no clear pattern on which aggregation settings may generally perform better. The different impact of the aggregation configurations on the downstream model performance may be caused by the intrinsic features of datasets. In particular, for the same dataset, the window size affects the different representation techniques in a similar way. Future researchers and practitioners are suggested to explore different feature aggregation settings by considering the characteristics of the datasets or utilizing log representation techniques that are more stable to different aggregation settings.

## 6 Discussions

In this section, based on our results for answering our research questions, we discuss the implications of our findings. Additionally, we summarize the key factors to consider when selecting the most appropriate log representation techniques for log-based anomaly detection approaches or other automated log analysis tasks. Our recommendations and discoveries will be helpful to researchers and practitioners in selecting the optimal log representation techniques and achieving favourable outcomes in their automated log analysis frameworks.

### 6.1 Implications

- **Automated log analysis approaches should pay attention to the choice of log representation techniques as they have a non-negligible impact on the follow-up models.** Existing log-based anomaly detection approaches usually consider only a single log representation technique. For example, in the work that adopts CNN to detect anomalies in log sequences (Lu et al. 2018), only log keys are used to learn the embeddings for log events, and information from log parameters and messages is lost in this process. Our results suggest that the performance of these approaches may be improved by considering other ways of log representations. Also, new representation approaches may be developed according to the specific tasks and downstream models. In particular, classic machine learning models may favour representations generated by traditional techniques. In contrast, deep-learning-based downstream models can better utilize semantic embedding to achieve better results. Also, experiments show that contextual embedding performs the best among the pre-trained language models. Our findings can provide guidance for future work to choose and design the appropriate log representation techniques for their specific tasks. For example, researchers should consider the capability of feature extraction and representation of the models they adopt when choosing the log representation techniques. Models of higher complexity (with more parameters) are more capable of dealing with higher dimensional representations.
- **Log parsing or other preprocessing are recommended before log representation process as they usually improve the performance of the downstream log analysis tasks.** Most of the prior works on log-based automated log analysis adopt a log parser to transform raw log to structured data. Recent work (Le and Zhang 2021) explores omitting the parsing process and extracting and representing information directly from the raw log data. However, they usually employ some preprocessing steps to remove unnecessary fields in log data. We find that the log parsing process generally positively impacts automated log analysis, although sometimes it may be timeconsuming and erroneous. Also, log parsing enables template-based log representation techniques and removes dynamic fields that will hinder the other semantic-based techniques. Thus, we suggest that researchers should carefully consider whether to employ the log parsing process in their workflow. As log parsers may sometimes be error-prone and consume

additional computational resources, researchers can consider substituting them with lightweight preprocessing processes.

- **Log analysis workflows should consider experimenting with different configurations of feature aggregation.** When aggregating low-level log representations to high-level ones, prior works (e.g., (Chen et al. 2021)) usually adopt a single strategy without experimenting with other configurations. However, according to our experiments, the feature aggregation process is essential for log representation. Feature aggregation configurations can significantly impact downstream models' performances. However, the impacts are closely related to the characteristics of the datasets. Prior works stand a good chance of achieving better performances when employing different feature aggregation configurations. Therefore, we advise researchers to consider the intrinsic features of the studied log data and employ different configurations when designing their automated log analysis workflow.

### 6.1.1 Key factors for selecting log representation techniques

To provide insights for researchers and practitioners in selecting appropriate log representation techniques, we summarize below the key factors that need to be considered based on our experiments and findings. We recommend researchers and practitioners consider these factors in their log-based anomaly detection and potentially other automated log analysis tasks to achieve optimal performance in such tasks.

- **Quality of representation** The quality of log representations is a crucial factor that significantly affects the performance of downstream models. In our study, we found that different models can benefit from different log representations. Across various models and datasets, we determined that the simplest message count vector representation can perform well in most cases. In addition, traditional anomaly detection models generally performed well on classical log representations, while deep models achieved better performance with semantic-based representations due to their stronger feature extraction and representation ability. Among the classical log representation techniques, the Message Count Vector approach achieved the best performance, while Context embedding (BERT) generally performed better among the semanticbased log representation techniques. Therefore, selecting high-quality log representation techniques is essential for achieving optimal downstream model performance.
- **Dimension of representation** One of the key factors to consider when selecting representation techniques is the dimension of the resulting representation. For some representation techniques, their resulting dimensions are data-invariant, which means the dimension will remain the same when they are applied to different log data. Semantic-based techniques (e.g., BERT) and graph-based techniques that utilize a neural network structure to generate embeddings for log data can usually provide fix-length outputs. By contrast, classical techniques (e.g., message count vector) usually rely on a vocabulary of tokens or log templates and thus, the dimensions are subjective to the data. The advantages of techniques with fixed output dimensions are obvious: First, they can better serve the scenarios when data shifting (e.g., vocabulary changes) exists in system logs caused by software evolution. When new log templates appear, these techniques are able to encode new templates while maintaining the feature property. Second, fixed output dimensions may enable more stable performances over different datasets on the same model. When working with datasets with a larger number of log templates,

the anomaly detection models may suffer from a performance loss due to their limited model capacity. Higher dimension input usually demands a larger model with more parameters. Classical techniques may generate representations of a wide range of dimensions over different datasets. For example, MCV generates vectors of 46 dimensions on the HDFS dataset, while for the Thunderbird, the dimension is 1,488 in our experiments. Higher dimensions may lead to higher computational costs in the training and prediction stages of follow-up downstream models.

- **Need for log parsing** As we discussed previously in RQ2, the log parsing process can generally remove noises caused by unprocessed tokens in log data, while errors induced by log parsers may cause performance loss (Le and Zhang 2022). Besides, the log parsing process can be time-consuming and require significant manual and computational resources. While log parsing is not essential for semantic-based representation techniques since they typically do not require log template information to operate, it may still be included as a preprocessing step for the logs. In this situation, a complete parsing process may be substituted by a lightweight preprocess, in which tokens that can not be processed by vectorizers are removed, when getting log templates is not mandatory for the representation technique.
- **Computational cost for representation construction** Another important consideration is the computational cost. Besides the log parsing process, log representation techniques require computational resources (time and space) to convert raw logs, log templates, or log template IDs to numeric vectors. As the mechanism varies across different techniques, the differences in computational cost are significant. For classical techniques, much memory may be used to construct dictionaries and vectorize tokens varying with datasets. For semantic-based techniques, although programmers can utilize the off-theshelf pre-trained models to escape the computational consumption for training the language models, some techniques still require heavy computations to acquire embeddings. For example, contextual embedding techniques require more computational resources to construct representations than static embedding techniques. Pre-trained BERT models process input tokens through transformer blocks, which involve significant computation and sometimes require specialized hardware (e.g., GPUs, TPUs), particularly for lengthy texts. By comparison, Word2Vec uses a shallow neural network, which is computationally efficient, to generate word embeddings. Taking this factor into account is important when designing anomaly detection workflows that are targeted for online or real-time application scenarios.
- **Granularity** It is mandatory to ensure that the level of log representation is aligned with the specific anomaly detection model being used. This is because different models require varying levels of granularity and types of information from the log data to perform according to their varying mechanisms. Semantic-based representation techniques (e.g., Word2Vec) can usually generate token-level representations, which can be aggregated into higher-level ones, while some classical techniques can only work on higher-level representations (e.g., MCV can only generate sequence-level representation). Therefore, it is crucial to carefully consider the requirements of the anomaly detection model being employed and choose the log representation accordingly.
- **Explainability** Finally, explainability is another factor to consider when selecting a log representation technique. Usually, classical log representation techniques (e.g., MCV), which represent the quantitative characteristics of log sequences, have better explainability compared with their semantic-based counterparts, which are learning-based. With a good explainability of log representation techniques, researchers can better understand the prediction given by the follow-up models and, therefore, are able to trace the

roots when performance is not satisfactory. Poor explainability of log representation techniques will make the decision-making process a black box, in which the decision-making process becomes agnostic. Future researchers should consider this factor when designing a trustworthy automated log analysis system.

In conclusion, selecting an appropriate log representation technique requires careful consideration of several factors, including the quality of representation, dimension of representation, need for log parsing, computational cost for representation construction, granularity, and explainability.

## 7 Threat to validity

We have identified the following threats to the validity of our findings: External validity. We carried out this research only based on the log anomaly detection task with the hope that our experimental results and findings can serve as a reference for other automated log analysis tasks. The conclusions may not apply to other downstream tasks, as different downstream tasks or models may have different intrinsic characteristics and favour different configurations or features of log representation. However, anomaly detection is one of the most studied downstream tasks in the domain of automated log analysis (Chen et al. 2021; Du et al. 2017; Fu et al. 2009; He et al. 2016b; Le and Zhang 2021; Lu et al. 2018; Meng et al. 2019; Nedelkoski et al. 2020; Wang et al. 2018; Xu et al. 2009; Zhang et al. 2019), demonstrating its importance and popularity. Due to the fact that many automated log analysis tasks share similar pipelines that process log data, our work may also inspire and support the designs of workflows of other tasks, despite the fact that only log-based anomaly detection is studied in our work.

As the mechanism of anomaly detection approaches differs greatly, we limit our research to the supervised log-based anomaly detection models to ensure a fair comparison among studied representations. Therefore, our findings may not apply to unsupervised methods. Future work that examines the impact of log representations on unsupervised learning tasks can complement our results.

Recently, new approaches (e.g., Transformer-based (Le and Zhang 2021; Nedelkoski et al. 2020), graph-based approaches (Wan et al. 2021; Xie et al. 2022)) have been applied to log-based anomaly detection. However, we did not evaluate them in this work, as the mechanisms of these approaches differ greatly, which makes it hard for us to fit them into our research questions. Future works may further examine these new approaches and their susceptibility to log representation techniques. To compensate for this, we discussed the most representative transformer-based approaches, and related the findings from these works with our experimental results and findings.

**Construct Validity** We followed some existing works in the experiment to use pre-trained models trained with natural language models. The experimental results may not reflect the true capability of these log representation techniques, as the effectiveness of generated log representations may suffer greatly from OOV issues or incorrect semantics caused by the different characteristics between log data and natural language.

**Internal Validity** Our configurations for dataset partition may not be optimal and may influence the accuracy of the evaluation. According to our survey, different log anomaly detection works adopt different grouping configurations for the studied public datasets. We referred to previous works and chose the most common grouping configurations to enable a better comparison. In addition, we employed different grouping configurations for the four studied datasets with the hope that our results and finding can be invariant to different grouping settings of datasets. Further study may be carried out to evaluate the impacts of the data grouping on the log representations. The hyperparameters for the machine learning models in our studies might not be fully optimized. Instead of aiming for the best performance for each particular model, our main goal was to examine how well alternative log representation strategies performed across various downstream models. As a result, we made sure that each representation technique was applied to the same dataset with identical parameter settings. Besides, our experimental results are generally consistent with those of prior studies that employed similar datasets, representations, and models. Additionally, we have included our implementations in our replication package, making it possible to reproduce our results. These factors help to mitigate the potential impact of using suboptimal hyperparameters in our study.Instead of directly assessing the quality of representations, we rely on the performance of downstream models as an indirect measure. However, the variables involved in these downstream tasks may affect the internal validity and introduce potential biases. To mitigate the potential bias, we consider multiple datasets and downstream models in our experiments.In addition, while most models performed well on the four datasets we examined, we found that the choice of log representation technique could affect downstream model performance. We did observe differences in F-scores when using different log representation techniques. These variations were statistically significant as confirmed by our SK-EST analysis.Furthermore, certain techniques' characteristics in our SK-Test resulted in some missing observations that could affect the ranking of the studied representation techniques, which we have indicated by clearly indicating the affected techniques.

## 8 Conclusions

Our work makes a comprehensive evaluation and review of six log representations on four public datasets with seven supervised anomaly detection models. We also examine the impacts of log parsing and feature aggregation of features on the effectiveness and quality of log representations. Our findings suggest that log representation techniques can significantly impact the performance of downstream models. We provide some general guidance and key factors on choosing suitable representation techniques. Also, we find that log parsing can generally improve the quality of log representation by reducing noise in some representation techniques. Moreover, the impacts of configuration for feature aggregation may vary according to the representation, data and downstream models. When designing an automated log analysis workflow, these factors should be carefully taken into account by researchers and engineers. For future work, we plan to evaluate log representation techniques with more automated log analysis downstream tasks and try to explore different features that different downstream tasks may favour. Thus, we can provide a more comprehensive direction for researchers to design their automated log analysis frameworks.

## Declarations

**Conflicts of interests** The authors have no competing interests to declare that are relevant to the content of this article.

## References

Chen M, Zheng AX, Lloyd J, Jordan MI, Brewer E (2004) Failure diagnosis using decision trees. In International Conference on Autonomic Computing, 2004. Proceedings., pages 36–43. IEEE

Chen Z, Liu J, Gu W, Su Y, Lyu MR (2021) Experience report: Deep learning-based system log analysis for anomaly detection. arXiv preprint arXiv:2107.05908

Chow M, Meisner D, Flinn J, Peek D, Wenisch TF (2014) The mystery machine: End-to-end performance analysis of large-scale internet services. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 217–231

Dai H, Li H, Chen CS, Shang W, Chen T-H (2020) Logram: Efficient log parsing using n-gram dictionaries. IEEE Transactions on Software Engineering 48(3):879–892. https://doi.org/10.1109/TSE.2020.3007554

Devlin J, Chang M-W, Lee K, Toutanova K (2018) Bert:Pretraining of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805

Du M, Li F, Zheng G, Srikumar V (2017) Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pages 1285–1298

El-Sayed N, Zhu H, Schroeder B (2017) Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pages 1333–1344. IEEE

Fu Q, Lou J-G, Wang Y, Li J (2009) Execution anomaly detection in distributed systems through unstructured log analysis. In 2009 ninth IEEE international conference on data mining, pages 149–158. IEEE

Fu Q, Lou J-G, Lin Q, Ding R, Zhang D, Xie T (2013) Contextual analysis of program logs for understanding system behaviors. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 397– 400. IEEE.

Grave E, Bojanowski P, Gupta P, Joulin A, Mikolov T (2018) Learning word vectors for 157 languages. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)

Hansen SE, Atkins ET (1993) Automated system monitoring and notification with swatch. In LISA, volume 93, pages 145–152. Monterey, CA

He S, He P, Chen Z, Yang T, Su Y, Lyu MR (2021) A survey on automated log analysis for reliability engineering. ACM Comput Sur (CSUR) 54(6):1–37

He P, Zhu J, He S, Li J, Lyu MR (2016a) An evaluation study on log parsing and its use in log mining. In 2016a 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pages 654–661. IEEE

He S, Zhu J, He P, Lyu MR (2016b) Experience report: System log analysis for anomaly detection. In 2016b IEEE 27th international symposium on software reliability engineering (ISSRE), pages 207–218. IEEE

He P, Zhu J, Zheng Z, Lyu MR (2017) Drain: An online log parsing approach with fixed depth tree. In 2017 IEEE international conference on web services (ICWS), pages 33–40. IEEE

He S, Zhu J, He P, Lyu MR (2020) Loghub: a large collection of system log datasets towards automated log analytics.arXiv preprint arXiv:2008.06448

Jarry R, Kobayashi S, Fukuda K (2021) A quantitative causal analysis for network log data. In 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), pages 1437–1442. IEEE

Katkar DGS, Kasliwal AD (2014) Use of log data for predictive analytics through data mining. Current Trends in Technology and Science. Volume: 3, Issue: 3 (Apr-May 2014)

Khan ZA, Shin D, Bianculli D, Briand L (2022) Guidelines for assessing the accuracy of log message template identification techniques. In Proceedings of the 44th International Conference on Software Engineering, pages 1095–1106

Le V-H, Zhang H (2021) Log-based anomaly detection without log parsing. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 492–504. IEEE.

Le V-H, Zhang H (2022) 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. IEEE

Le V-H, Zhang H (2023) Log parsing with prompt-based few-shot learning. arXiv preprint arXiv:2302.07435.

Li X, Chen P, Jing L, He Z, Yu G (2020) Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults. In 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), pages 92–103. IEEE.

Liang Y, Zhang Y, Xiong H, Sahoo R (2007) Failure prediction in ibm bluegene/l event logs. In Seventh IEEE International Conference on Data Mining (ICDM 2007), pages 583–588. IEEE.

Liao L, Chen J, Li H, Zeng Y, Shang W, Guo J, Sporea C, Toma A, Sajedi S (2020) Using black-box performance models to detect performance regressions under varying workloads: an empirical study. Empir Softw Eng 25(5):4130–4160

Liu FT, Ting KM, Zhou Z-H (2012) Isolation-based anomaly detection. ACM Trans Knowl Discov Data (TKDD) 6(1):1–39

Liu Y, Zhang X, He S, Zhang H, Li L, Kang Y, Xu Y, Ma M, Lin Q, Dang Y et al (2022) Uniparser: A unified log parser for heterogeneous log data. Proc ACM Web Conf 2022:1893–1901

Lou J-G, Fu Q, Yang S, Xu Y, Li J (2010) Mining invariants from console logs for system problem detection. In 2010 USENIX Annual Technical Conference (USENIX ATC 10)

Lu S, Wei X, Li Y, Wang L (2018) 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. IEEE

Lyu Y, Li H, Sayagh M, Jiang ZM, Hassan AE (2021) An empirical study of the impact of data splitting decisions on the performance of aiops solutions. ACM Trans Softw Eng Methodol (TOSEM) 30(4):1–38

Meng W, Liu Y, Zhu Y, Zhang S, Pei D, Liu Y, Chen Y, Zhang R, Tao S, Sun P et al (2019) Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In IJCAI 19:4739–4745

Meng W, Liu Y, Zhang S, Zaiter F, Zhang Y, Huang Y, Yu Z, Zhang Y, Song L, Zhang M et al (2021) Logclass: Anomalous log identification and classification with partial labels. IEEE Trans Netw Serv Manage 18(2):1870–1884

Nagaraj K, Killian C, Neville J (2012) Structured comparative analysis of systems logs to diagnose performance problems. In 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pages 353–366

Nedelkoski S, Bogatinovski J, Acker A, Cardoso J, Kao O (2020) Self-attentive classification-based anomaly detection in unstructured logs. In 2020 IEEE International Conference on Data Mining (ICDM), pages 1196–1201. IEEE

Nguyen KA,Walde SSi, Vu NT (2016) Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. arXiv preprint arXiv:1605.07766.

Oliner A, Ganapathi A, Xu W (2012) Advances and challenges in log analysis. Commun ACM 55(2):55–61

Oliner A, Stearley J (2007) 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. IEEE

Prewett JE (2003) Analyzing cluster log files using logsurfer. In Proceedings of the 4th Annual Conference on Linux Clusters. Citeseer

Rouillard JP (2004) Real-time log file analysis using the simple event correlator (sec). In LISA 4:133–150

Rusticus SA, Lovato CY (2014) Impact of sample size and variability on the power and type i error rates of equivalence tests: A simulation study. Pract Assess Res Eval 19(1):11

Salton G, Buckley C (1988) Term-weighting approaches in automatic text retrieval. Inf Process Manage 24(5):513–523

Schroeder B, Gibson GA (2007) Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In 5th USENIX Conference on File and Storage Technologies (FAST 07), San Jose, CA. USENIX Association

Shang W, Jiang ZM, Adams B, Hassan AE, Godfrey MW, Nasser M, Flora P (2014) An exploratory study of the evolution of communicated information about the execution of large software systems. J Softw: Evol Process 26(1):3–26

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2017) An empirical comparison of model validation techniques for defect prediction models. IEEE Transactions on Software Engineering 43(1):1–18. https://doi.org/10.1109/TSE.2016.2584050

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2018) The Impact of Automated Parameter Optimization on Defect Prediction Models. IEEE Transactions on Software Engineering 45(7):683–711. https://doi.org/10.1109/TSE.2018.2794977

Turc I, Chang M-W, Lee K, Toutanova K (2019) Well-read students learn better: On the importance of pre-training compact models. arXiv preprint arXiv:1908.08962v2

Van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9(11)

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc.

Wan Y, Liu Y, Wang D, Wen Y (2021) Glad-paw: Graph-based log anomaly detection by position aware weighted graph attention network. In Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part I , pages 66–77. Springer

Wang M, Xu L, Guo L (2018) Anomaly detection of system logs based on natural language processing and deep learning. In 2018 4th International Conference on Frontiers of Signal Processing (ICFSP), pages 140–144. IEEE

Xie Y, Zhang H, Babar MA (2022) Loggd: Detecting anomalies from system logs by graph neural networks. arXiv preprint arXiv:2209.07869

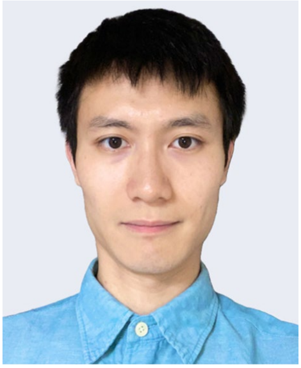
Xu W, Huang L, Fox A, Patterson D, Jordan MI (2009) Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 117– 132

Yuan D, Mai H, Xiong W, Tan L, Zhou Y, Pasupathy S (2010) Sherlog: error diagnosis by connecting clues from run-time logs. In Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems, pages 143–154

Yuan D, Park S, Zhou Y (2012) Characterizing logging practices in open-source software. In 2012 34th International Conference on Software Engineering (ICSE), pages 102–112. IEEE

Zhang X, Xu Y, Lin Q, Qiao B, Zhang H, Dang Y, Xie C, Yang X, Cheng Q, Li Z, et al. (2019) Robust log-based anomaly detection on unstable log data. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 807–817

Zhu J, He S, Liu J, He P, Xie Q, Zheng Z, Lyu MR (2019) 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. IEEE



**Xingfang Wu** is a Ph.D. candidate in the Department of Computer and Software Engineering at Polytechnique Montreal, Montreal, Canada. His current research interests are software log analysis and technical forum data mining.

**Heng Li** is an assistant professor in the Department of Computer and Software Engineering at Polytechnique Montreal, where he leads the MOOSE lab (moose.polymtl.ca). He holds a PhD in Computing from Queen's University (Canada), M.Sc. from Fudan University (China), and B.Eng. from Sun Yat-sen University (China). Prior to his academic career, he worked in the industry for years as a software engineer at Synopsys and as a software performance engineer at BlackBerry. His and his students' research aims to address the practical challenges in software monitoring, software quality engineering, intelligent operations of software systems, and quality engineering of machine learning applications.

**Foutse Khomh** is a Full Professor of Software Engineering at Polytechnique Montréal, a Canada CIFAR AI Chair on Trustworthy Machine Learning Software Systems, and an FRQ-IVADO Research Chair on Software Quality Assurance for Machine Learning Applications. He received a Ph.D. in Software Engineering from the University of Montreal in 2011, with the Award of Excellence. He also received a CS-Can/Info-Can Outstanding Young Computer Science Researcher Prize for 2019. His research interests include software maintenance and evolution, machine learning systems engineering, cloud engineering, and dependable and trustworthy ML/AI. His work has received four ten-year Most Influential Paper (MIP) Awards, and six Best/Distinguished Paper Awards. He also served on the steering committee of SANER (chair), MSR, PROMISE, ICPC (chair), and ICSME (vice-chair). He initiated and co-organized the Software Engineering for Machine Learning Applications (SEMLA) symposium and the RELENG (Release Engineering) workshop series. He is on the editorial board of multiple international software engineering journals (e.g., IEEE Software, EMSE, JSEP) and is a Senior Member of IEEE.