Robust Log-Based Anomaly Detection on Unstable Log Data

Xu Zhang[*]
Microsoft Research
Beijing, China
Nanjing University
Nanjing, China

Yong Xu
Qingwei Lin
Bo Qiao
Microsoft Research
Beijing, China

Hongyu Zhang
University of Newcastle
Callaghan, Australia

Yingnong Dang
Microsoft Azure
Redmond, USA

Chunyu Xie
Microsoft Research
Beijing, China

Xinsheng Yang
Qian Cheng
Ze Li
Microsoft Azure
Redmond, USA

Junjie Chen
College of Intelligence and Computing, Tianjin University
Tianjin, China

Xiaoting He
Microsoft Research
Beijing, China

Randolph Yao
Microsoft Azure
Redmond, USA

Jian-Guang Lou
Microsoft Research
Beijing, China

Murali Chintalapati
Microsoft Azure
Redmond, USA

Furao Shen
Nanjing University
Nanjing, China

Dongmei Zhang
Microsoft Research
Beijing, China

ABSTRACT

Logs are widely used by large and complex software-intensive systems for troubleshooting. There have been a lot of studies on log-based anomaly detection. To detect the anomalies, the existing methods mainly construct a detection model using log event data extracted from historical logs. However, we find that the existing methods do not work well in practice. These methods have the *close-world assumption*, which assumes that the log data is stable over time and the set of distinct log events is known. However, our empirical study shows that in practice, log data often contains previously unseen log events or log sequences. The instability of log data comes from two sources: 1) the evolution of logging statements, and 2) the processing noise in log data. In this paper, we propose a new log-based anomaly detection approach, called *LogRobust*. LogRobust extracts semantic information of log events and represents them as semantic vectors. It then detects anomalies by utilizing an attention-based Bi-LSTM model, which has the ability to capture the contextual information in the log sequences and automatically learn the importance of different log events. In this way, LogRobust is able to identify and handle unstable log events and sequences. We have evaluated LogRobust using logs collected from the Hadoop system and an actual online service system of Microsoft. The experimental results show that the proposed approach can well address the problem of log instability and achieve accurate and robust results on real-world, ever-changing log data.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**.

KEYWORDS

Anomaly Detection, Log Analysis, Deep Learning, Log Instability, Data Quality

ACM Reference Format:
Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust Log-Based Anomaly Detection on Unstable Log Data. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3338906.3338931

[*]zhangxu037@smail.nju.edu.cn

1 INTRODUCTION

Large and complex software-intensive systems, such as online service systems and big data systems, produce logs for troubleshooting. The log messages are usually semi-structured text strings, which are used to record events or states of interest. Engineers can examine recorded logs to understand the status of software systems, detect system anomalies and locate the root causes. Because of its simplicity and effectiveness, logging has been commonly adopted in practice. For example, an empirical study [47] on two Microsoft systems and two open source projects shows that logging is commonly used. Statistically, there is one line of logging code in every 58 lines of source code.

As the scale and complexity of the system increase, it is becoming more difficult to detect system anomalies by manual examination of the logs. Over the years, many automated log-based approaches have been proposed to detect system anomalies [3, 10, 18, 29, 44]. These work retrieve useful information from logs and adopt data mining and machine learning techniques to analyze log data and detect the occurrence of system anomalies. For example, Xu, et al. [44] formulated the log-based anomaly detection problem as an unsupervised learning problem and utilized Principal Component Analysis (PCA) to detect anomalies. Lou, et al. [31] mined the invariants from console logs, where a system anomaly is detected if the occurrences of log events break a certain invariant during the lifecycle of the system execution.

Although effective, the existing log-based anomaly detection approaches are not sufficiently robust in practice. To detect anomalies, almost all existing approaches require to construct a detection model using the known *log events* (i.e., the templates of log messages) and *log sequences* (i.e., series of log events that record specific execution flows) extracted from the training data. They fail to work with previously unseen log events and log sequences. However, our empirical study has found that real-world log data is *unstable*, meaning that new but similar log events and log sequences often appear.

We have identified the following two sources of instability in real-world log data:

(1) *Evolution of logging statements*: Like other software artifacts, logs are always evolving. Developers may frequently modify source code including logging statements, which in turn leads to changes to log data. As Kabinna, et al. [24] observed, around 20%~45% of logging statements in their studied projects changed throughout their lifetime. Many new log events and log sequences are generated by the ever-changing logging statements. It was reported that Google's systems have up to thousands of new log printing statements every month [42]. Therefore, log-based anomaly detection approaches must be able to cater to the evolution of log data. This is especially important for organizations that adopt the Continue Delivery/Deployment approaches [6, 22].
(2) *Processing noise in log data*: During collection, retrieval, and pre-processing of log data, it is inevitable that a certain degree of noise is introduced into the original log data. For example, the noise could come from the data collection process. In a large-scale system, many logs are produced by geographically distributed components separately and then uploaded to a centralized location for further analysis. Missing, duplicated, or disordered log messages could be resulted from such a process (e.g., due to network errors, limited system throughput, storage issues, etc.). Another important source of noise comes from log parsing. In general, an early step of log data analysis is to extract the log events from the raw log messages using a log parser. However, it has been observed that the existing log parsers are not sufficiently accurate [17], which could lead to many misidentified log events. The noise in log data hampers the effectiveness of the existing log-based anomaly detection approaches.

To investigate the log instability issue, we have performed an empirical study on log data produced by a large-scale online service system of Microsoft. We find that unstable log data is very common in real-world systems. The details of the study are reported in Section 2.2.

Due to the instability of log data, the effectiveness of the existing anomaly detection approaches is significantly affected. The existing approaches are based on a *close-world assumption*: the patterns of log events and log sequences are constant, which is not practical in real-world systems where unstable log events and sequences always appear. On the one hand, a small change to an existing log event (as shown in Figure 3) can introduce a different but semantically similar one, which is recognized by existing approaches as a brand new log event [10, 18, 29, 44]. Therefore, existing approaches will either fail to work due to the incompatibility with these unseen log events, or result in low performance due to the incorrect classification. On the other hand, log sequences are also likely to be changed due to new execution paths or processing noise. However, traditional approaches only leverage the occurrence information of log sequences and ignore its context attribute and different importance of logs. It is also worth noting that a large-scale online service system is always under active development and maintenance, the log data could change frequently. Therefore, it is unpractical to update log-based anomaly detection tools continuously due to the large amount of effort it requires. A detailed discussion of existing approaches will be given in Section 2.3 .

To overcome the instability issue, in this paper, we propose LogRobust, a novel log-based anomaly detection approach, which can achieve accurate and robust anomaly detection on real-world, ever-changing and noisy log data. Unlike the existing approaches, LogRobust does not rely on the simple occurrence information of log events. Instead, it transforms each log event into a *semantic vector* of the fixed dimension. Semantic vectors are capable of capturing the semantic information embedded in log events. Through semantic understanding, this representation method is able to identify and handle new but similar log events that emerge from evolving logging statements and parsing errors. Then, taking the sequence of semantic vector as input, an attention-based Bidirectional Long-Short-Term Memory Neural Network (Bi-LSTM) classification model is applied to detect the anomalies. The attention-based Bi-LSTM model has the ability to capture the contextual information in the log sequences and automatically learn the importance of different log events. Thus, it is robust to the variations in the sequences.

We have evaluated the proposed approach using the public log data collected from Hadoop. We inject different ratios of changes to the Hadoop log data and evaluate the effectiveness of the proposed approach. The experimental results show that LogRobust is robust: when the injection rate is increased from 5% to 20%, the F1-score is only slightly decreased from 0.96 to 0.89. We have also applied LogRobust to the industrial log data collected from a large-scale online service system of Microsoft with real instability issue. LogRobust achieves F1-Score of 0.84, which is 30% higher than other traditional methods. The experimental results show that the proposed approach can effectively detect anomalies of the online service system, with the ever-changing and noisy log data.

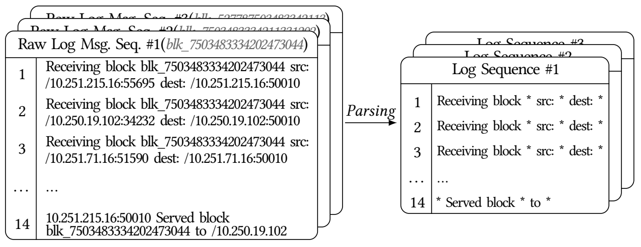
The main contributions of this paper are as follows:

(1) We point out the problem of log instability in log-based anomaly detection. We also conduct an empirical study on log instability in real-world systems to confirm our findings.
(2) We propose LogRobust, a new log-based anomaly detection approach, which is robust to the unstable log data. To our best knowledge, we are the first to address the instability issue of log data in anomaly detection.
(3) We have evaluated LogRobust using both public and real-world industrial datasets. The results confirmed the effectiveness of our approach.

The remainder of this paper is organized as follows: We introduce the background of our work and an empirical study of unstable log data in Section 2. Section 3 describes our approach. Section 4 presents our experimental design and results. Section 5 discusses the incremental updating of our LogRobust and the threats to validity. Section 6 surveys related work followed by Section 7 which concludes this paper.

2 BACKGROUND AND EMPIRICAL STUDY

2.1 Log Terminology


Raw Log Msg. Seq. #3(blk_5277875034823342113)
Raw Log Msg. Seq. #2(blk_7503483334211331202)
Parsing
Log Sequence #3
Log Sequence #2

```
Raw Log Msg. Seq. #1(blk_7503483334202473044)
1    Receiving block blk_7503483334202473044 src:
     /10.251.215.16:55695 dest: /10.251.215.16:50010
2    Receiving block blk_7503483334202473044 src:
     /10.250.19.102:34232 dest: /10.250.19.102:50010
3    Receiving block blk_7503483334202473044 src:
     /10.251.71.16:51590 dest: /10.251.71.16:50010
...  ...
14   10.251.215.16:50010 Served block
     blk_7503483334202473044 to /10.250.19.102
```

```
Log Sequence #1
1    Receiving block * src: * dest: *
2    Receiving block * src: * dest: *
3    Receiving block * src: * dest: *
...  ...
14   * Served block * to *
```

(a) Raw Log Messages **(b) Log Events**

Figure 1: An Example of Raw Log Messages, Log Events and Log Sequence

Logs, which contain abundant information (e.g., events, parameters, execution details, etc.) about the running status of a software-intensive system, play a crucial role in the maintenance of online service systems. Once a problem/anomaly occurs, engineers often rely on system logs for further investigation.

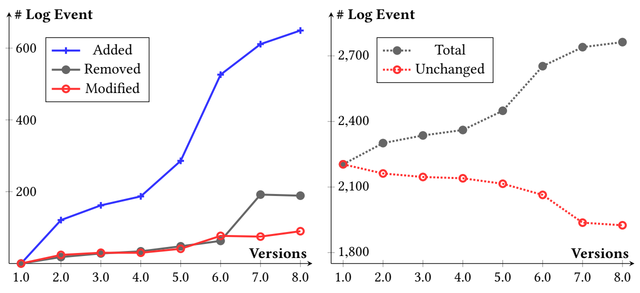
Figure 1 shows an example of log data from the public HDFS dataset [43]. Each line printed to the system console is a *log message*. Here we omit some fields for clarity. Each *log message* consists of a constant part (*log event*) and a variable part (*log parameter*). The *log parameter* records some system attributes (e.g., URL, file name or IP address, etc.). The *log event* is comprised of fixed text strings and is a template of a log message. A log event is acquired from a log message through *log parsing*.

A *log sequence* consists of a sequence of log events which records an execution flow of a specific task. Log events from the same log sequence share the same task ID, which can be used to link the events chronologically. Figure 1b shows an example of log sequence with the task ID `blk_7503483334202473044`.

2.2 An Empirical Study on Log Instability

In this section, we describe an empirical study in real-world industrial scenarios, which demonstrates the problem of log instability and reveals the subsequent consequences it causes.

2.2.1 Evolution of Logging Statements. We study log evolution on a real-world online service system Service X from Microsoft. We identify evolving log events by analyzing the changes to logging statements in the source code. We use the statistics of log events in version 1.0 as the baseline and observe the changes in the numbers of new versions. The results are shown in Figure 2. It can be seen from Figure 2a that there are a lot of newly added log events in newer versions (the blue line). In addition, some old log events are also removed from the source code (the black line) and a part of original log events have been modified in the new versions (the red line). Furthermore, we can see from the Figure 2b that the total number of log events keeps increasing, from 2,204 in version 1.0 to 2,763 in version 8.0. The unchanged log events are becoming less and less in the newer versions. In the latest version (version 8.0), the number of changed log events accounts for 30.3% of the total log events.



(a) Changed log events **(b) All log events**

Figure 2: The Evolution of Log Events across Versions

Figure 3 gives some real cases of evolving log events from Service X in Microsoft, where the two log messages before and after the changes are given.

Case 1:

```
Creating and opening new * channel factory
```

```
Creating and opening new * channel factory for bindingId *
```

Case 2:

```
{"Result":* "BaseHash":* UploadHash":* TipHash":* NewTipHash":* }
```

```
MiniMerge: {"Result":* "BaseHash":* UploadHash":* TipHash":* NewTipHash":* }
```

Case 3:

```
PutSharedRemoteServiceLocation - Entering with inputs [LocationType: * ]
[SrsLocation: [Server: * ] [Cluster: * ] [Session: * ] [BackupServer: * ] [Back-
upSession: * ] [Environment: * ] [IsADocumentSession: * ]] [IsClear: * ]
```

```
PutSharedRemoteServiceLocation - Entering with inputs [LocationType: * ]
[SrsLocation: [Server: * ] [Cluster: * ] [Session: * ] [BackupServer: * ] [Back-
upSession: * ] [Environment: * ]] [IsClear: * ]
```

Figure 3: Examples of Evolving Log Events

- Case 1: Two new words (*"for bindingId"*) are added to the original log event as a supplementary explanation for clarity.

- Case 2: A keyword (*"MiniMerge"*) is prepended to the original log event to indicate the program execution stage.
- Case 3: A keyword (*"IsADocumentSession"*) is removed as the related feature is deprecated.

We also study the evolving log sequence in Service X. We collect log data from two days spanning about one month. We notice that the amount of new log sequences accounts for more than 90% of the total dataset, which indicates that almost all log sequences have changed during a month time. After further analysis, we found 84.2% new log sequences are caused by a new added log event *"Setting OCSSession as *"*. Other new log sequences are caused by brand new execution paths.

2.2.2 Processing Noise. Processing noise is introduced during collection, retrieval, and pre-processing of log data, which are not truly anomalies of the system. For example, inaccurate log parsing can hamper the performance of an anomaly detection model [17]. Log parsing is one of early steps in log data analysis. He, et al. [17] evaluated four commonly used log parsers, including SLCT [40], IPLoM [32], LKE [12] and LogSig [39]. Zhu, et al. [48] tested 13 log parsers on a total of 16 datasets. They all found that existing log parsers are not sufficiently accurate. These log parsers could perform very differently on different datasets. For example, the LogSig [39] parser could achieve an accuracy of 0.91 in HDFS dataset but only 0.26 in BGL dataset [17]. In [48], LogSig can achieve accuracy of 0.967 in Proxifier dataset but only 0.169 in Linux dataset. More importantly, He, et al. also pointed out that log mining is sensitive to some critical events [17]. They claimed that 4% errors in parsing could even cause an order of magnitude performance degradation in anomaly detection (from 40% to 3.7%).

We also perform a study on Service X system using a commonly-used log parser, *Drain* [19]. We randomly sample 7,401 Service X log messages generated from 82 log events, and apply *Drain* to parse them. We find that 1,377 log messages are incorrectly parsed, which account for 18.6% of the total log messages. Most of parsing errors are caused by missing/adding a few keywords from/into log events. Figure 4 illustrates two examples of parsing error in Service X log data. The ground truth log events and parsing results are given, respectively.

- Case 1: Log parser omits one keyword "resize" compared to the ground truth log event.
- Case 2: Log parser misidentifies parameter "rtc" as a keyword of the log event.

These parsing errors lead to many extra log events, which are not conducive to the follow-up analysis.

Case 1:
```
HttpRequestAsync::EnsureBuffer - allocated:* resize:*
```
```
HttpRequestAsync::EnsureBuffer - allocated:*
```
Case 2:
```
Skipping update of with app alias [*]
```
```
Skipping update of with app alias [rtc]
```

Figure 4: Examples of Log Parsing Error

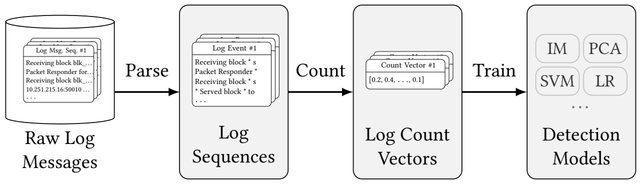


Figure 5: The Overview of Traditional Approaches

2.3 Limitation of Existing Methods

Over the years, there are many studies focused on log-based anomaly detection [3, 10, 18, 29, 31, 44]. Some exemplary approaches are as follows:

- IM: Lou, et al. [31] proposed Invariant Mining (IM) to mine the invariants (linear relationships) among log events from log count vectors. Those log sequences that violate the invariants are considered as anomalies.
- PCA: Xu, et al. [44] constructed normal space and anomalous space of log count vectors using Principal Component Analysis (PCA) to detect anomalies. If a log count vector is far from the normal space, it is considered as an anomaly.
- SVM and LR: [18, 29] represented log sequences as log count vectors and applied supervised learning algorithms to detect anomalies. In their papers, they train the classifier models using Support Vector Machine (SVM) and Logistic Regression (LR).

The existing approaches mentioned above have many common characteristics. Figure 5 illustrates the overview of these approaches. They all transform the log sequences into *log count vectors*, then build unsupervised or supervised machine learning models on it to detect anomalies. Here a log count vector holds the occurrence of each log event in a log sequence. The dimension of the log count vector is equal to the number of distinct known log events.

Due to the log instability issue, the limitation of this method is obvious. Firstly, it is incompatible with unstable log events. Even if only one unstable log event occurs (e.g., caused by evolving logging statement or parsing error), the dimension of log count vector must be changed and the model also needs to be retrained accordingly. Secondly, log count vector only counts the number of log events thus it ignores the context information embedded in the log sequences and cannot identify the different importance of various log events. Thirdly, to update the log-based anomaly detection tools, continuous model re-training is required, which could incur unacceptable cost for a large-scale software system that is under active development and maintenance. Therefore, it is also impractical to update the existing anomaly detection tools in an online manner. As a consequence, the existing approaches will either fail to work, or result in low detection accuracy.

3 LOGROBUST: AN APPROACH TO ROBUST LOG-BASED ANOMALY DETECTION

To overcome the instability problem of real-world log data, we propose LogRobust, a novel log-based anomaly detection approach. The overview of LogRobust is shown in Figure 6. The first step is log parsing. After that, LogRobust does not rely on the log count

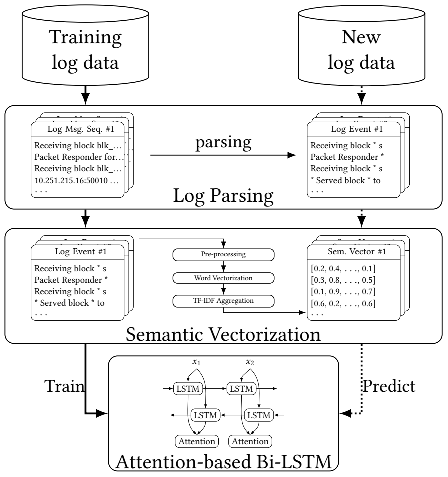


Figure 6: The Overview of LogRobust

vector like what the existing methods [3, 18, 29, 31] do. Instead, it transforms each log event into a vector by considering its semantic information. We call it *Semantic Vectorization*. In this way, even though a log event is changed during the evolution (or introduced by processing noise), it can be still represented as a similar vector to the original log event. Therefore, our approach is able to handle unstable log events. After semantic vectorization, LogRobust leverages the attention-based Bi-LSTM neural network [21] to detect the anomalies, which can capture the contextual knowledge of log sequence and learn to assign various degrees of importance to different log events. In this way, our approach is able to handle unstable log sequences.

Overall, LogRobust consists of three steps: log parsing (Section 3.1), semantic vectorization (Section 3.2), attention-based classification (Section 3.3). Also, we present the usage of LogRobust in Section 3.4.

3.1 Log Parsing

Since raw log messages are unstructured data and contain much specific information (e.g., IP address, file name, etc.) that can hinder automatic log analysis [18], LogRobust needs to parse each log message to extract its log event by abstracting away the parameters in the message. In this way, the log messages become structured data, which facilitates follow-up analysis. Considering accuracy and efficiency, LogRobust adopts Drain [19] method to conduct log parsing. Drain is proposed in [19] and is characterized by its high parsing accuracy and efficiency. In fact, it achieves the best performance compared with other related methods evaluated in [48]. Thus, we choose this method to conduct log parsing. For example, in Figure 1, the first raw log message "*Receiving block blk_7503483334202473044 src:/10.251.215.16:55695 dest:/10.251.215.16:50010*" is parsed into the log event "*Receiving block * src: * dest: **". It is worth noting that the parsing process of Drain may also introduce the noise due to its inaccuracy, but our approach can handle this problem (to be described in the following subsections).

3.2 Semantic Vectorization

LogRobust extracts the semantic information of log event and transforms each log event into a fixed-dimension vector (we call it *semantic vector*), regardless of whether the log event exists before. The workflow of semantic vectorization is shown in Figure 7, which consists of three steps: pre-processing of log event, word vectorization, and TF-IDF based aggregation.

3.2.1 *Pre-processing of Log Events.*

To capture the semantics of log events, LogRobust treats a log event E as a sentence in natural language, denoted as $S = [t_1, t_2, ...t_N]$, where $t_i, i \in [1, N]$ represents the i-th token and N is the length of the log-event sentence. Most tokens are valid English words, which have their own meanings. However, there are also some non-character tokens and many variable names in a log event. LogRobust conducts pre-processing for each log-event sentence as follows.

We firstly remove all non-character tokens from log-event sentences S, such as delimiters, operators, punctuation marks, and number digits. Then we remove all the stop words such as "a", "the", etc. Finally, some variable names in log events are actually a concatenation of words. For example, the variable name "TypeDeclaration" contains two words: "type" and "declaration", and the variable name "isCommitable" is composed of two words: "is" and "Commitable". LogRobust splits these composite tokens into individual tokens according to *Camel Case* [9].

3.2.2 *Word Vectorization.*

After pre-processing, LogRobust transforms each log-event sentence S into a semantic vector $\boldsymbol{V}$. The transformation should satisfy the following two requirements:

- Discrimination: Semantic vectors should be able to represent different log events with high discrimination. For example, "*Receiving block * src: * dest: **" and "*PacketResponder * for block * terminating*" are two different log events, thus their corresponding semantic vectors should be different. Formally, it means that the cosine similarity between the two vectors should be low.
- Compatibility: Semantic vectors should be able to identify unstable log events with similar semantics. For example, when a log event is changed from "*Receiving block * src: * dest: **" to "*Receiving block * src: * dest: * time: * content **" during the evolution, both of them actually have similar semantics. Therefore it is important to represent them as similar vectors.

In order to meet the above two requirements, LogRobust integrates the *semantic* information of log events into the vectorization. Specifically, LogRobust leverages off-the-shelf word vectors, which were pre-trained on Common Crawl Corpus dataset using the FastText algorithm [23], to extract the semantic information from the English vocabularies. FastText can sufficiently capture the intrinsic relationship (i.e., semantic similarity) among words in natural language and map each word to a d-dimension vector (where $d = 300$ in FastText word vectors). After replacing words with corresponding vectors, a log-event sentence S is transformed into word vectors list $L = [\boldsymbol{v_1}, \boldsymbol{v_2}, ..., \boldsymbol{v_N}]$, where $\boldsymbol{v_i} \in \mathbb{R}^d, i \in [1, N]$ denotes the word vector, N is the number of tokens in a log-event sentence.

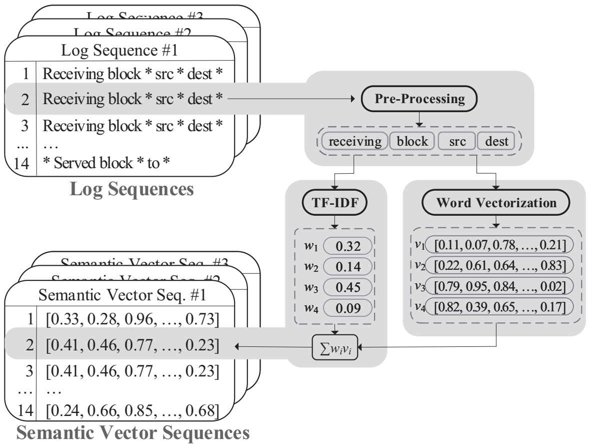


Figure 7: The Work Flow of Semantic Vectorization

3.2.3 TF-IDF Based Aggregation. Next, LogRobust represents a log event as a fixed-dimension vector by aggregating all N word vectors in L. In this way, even though the numbers of words (i.e., N) in different log events are varying, the dimension of the semantic vectors are fixed (i.e., d). Here LogRobust applies weighted aggregation using TF-IDF [36], which is a widely-used method in information retrieval. The TF-IDF weight can effectively measure the importance of words in sentences, which exactly serves the requirement of high discrimination. For example, if the word "Block" appears frequently in a log event, it means that this word may be more representative for this log event. We thus use the *Term Frequency* (TF) to describe its importance, where $TF(word) = \frac{\#word}{\#total}$, $\#word$ is the number of target word in a log event, $\#total$ is the number of all words in a log event.

On the other hand, if the word "Block" appears in all log events, it becomes too common to be able to distinguish those log events, and thus its weight should be reduced. Therefore, we also utilize the *Inverse Document Frequency* (IDF) as the metric, where $IDF(word) = \log\left(\frac{\#L}{\#L_{word}}\right)$, $\#L$ is total number of all log events and $\#L_{word}$ is number of log events containing target word. For each word, its TF-IDF weight w is calculated by $TF \times IDF$.

Finally, we can obtain the semantic vector $\boldsymbol{V} \in \mathbb{R}^d$ to represent a specific log event by summing up the word vectors in L with respect of TF-IDF weights, according to the Eq. 1.

$$\boldsymbol{V} = \frac{1}{N}\sum_{i=1}^{N} w_i \cdot \boldsymbol{v_i} \qquad (1)$$

The semantic vector is able to identify the semantically similar log events and also distinguish different log events. In this way, LogRobust is able to handle the instability of log data.

3.3 Attention-based Classification

Through semantic vectorization, each log event E is transformed into a semantic vector $\boldsymbol{V}$. At the same time, each log sequence is accordingly represented as a list of semantic vectors (like $[\boldsymbol{V_1}, \boldsymbol{V_2}, \ldots, \boldsymbol{V_T}]$), we call it as *semantic vector sequence.* Taking such a semantic vector sequence as input, LogRobust adopts the *Attention-based*

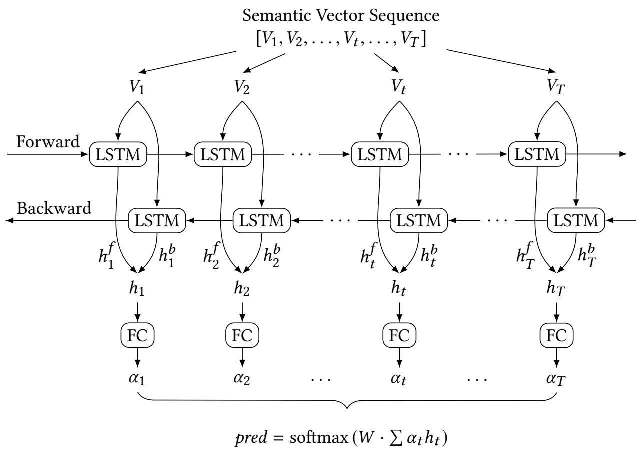


Figure 8: Attention-based Bidirectional Long Short-Term Memory Neural Network as Anomaly Detection Model

Bi-LSTM neural network for anomaly detection to cope with the unstable log sequences.

The LSTM model [21], a variant of the Recurrent Neural Network (RNN), is specifically designed for sequential data. It is able to capture the contextual information of the sequence because it belongs to a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. LSTM consists of an input layer, a hidden neurons layer and an output layer. Bi-LSTM splits the hidden neurons layer of a standard LSTM into two directions: the forward pass and the backward pass [21], in order to capture sufficient information of input log sequences in both directions. As shown in Figure 8, $\boldsymbol{h_t^f}$ and $\boldsymbol{h_t^b}$ are hidden state vectors at time step t in forward pass or backward pass, respectively, where time step t represents the position of the input log sequence. We concatenate both hidden states as h_t to capture the information from both directions, i.e. $\boldsymbol{h_t} = \text{concat}\left(\boldsymbol{h_t^f}, \boldsymbol{h_t^b}\right)$.

Since different log events have different impacts on the classification result, we introduce the attention mechanism to the Bi-LSTM model to assign different weights to log events. In this way, the impact of log data noise can be also reduced, since those noisy log events tend to have less importance and are more likely to be given low attention. The importance can be learned automatically from the attention layer. More specifically, we add a fully connected layer (i.e., FC layer in Figure 8) as the attention layer to the concatenated hidden state h_t and its output is the weight of attention (denoted as α), which reflects the importance of a log event. The larger the α is, the more the model pays attention to this log event. The computation of α is shown in Eq. 2, where W_t^a is the weight of the attention layer at time step t.

$$\alpha_t = \tanh\left(\mathbf{W_t^\alpha} \cdot \boldsymbol{h_t}\right) \qquad (2)$$

Finally, we sum all the hidden states with respect to all these α, and then construct a softmax layer to output the classification result. As shown in Eq. 3, W is the softmax layer weight and T is the total length of the log sequence.

$$pred = \text{softmax}\left(\mathbf{W} \cdot \left(\sum_{t=0}^{t=T} \alpha_t \cdot \boldsymbol{h_t}\right)\right) \tag{3}$$

During the training phase, we use the prediction outputs and the ground-truth provided by datasets to calculate the cross-entropy [27] as the loss function. Through this loss function, we utilize the Stochastic Gradient Descent algorithm [25] to train the parameters of the model (including the weights of Bi-LSTM and Attention layer).

3.4 Usage of LogRobust

Following the above steps, LogRobust can build a robust model for anomaly detection on unstable log data. As depicted in Figure 6, given a set of new log data, we firstly conduct log parsing. After that, for each log sequence to be detected, LogRobust transforms each log event in the log sequence into a semantic vector. The log sequence is represented as a list of semantic vectors accordingly. Then we feed it into the trained model. Finally, the attention-based Bi-LSTM can predict whether this sequence is an anomaly or not.

4 EXPERIMENT

In this section, we evaluate our approach by answering the following research questions:

RQ1: How effective is the proposed LogRobust approach on unstable log data?

RQ2: How effective is the attention mechanism in the proposed LogRobust approach?

RQ3: How effective is the proposed LogRobust approach on stable log data?

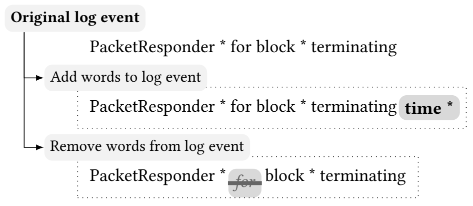
4.1 Experimental Design

4.1.1 Datasets. We evaluate our proposed LogRobust on three datasets, including the original HDFS dataset, the synthetic unstable HDFS datasets and the real-world industrial dataset collected from Microsoft.

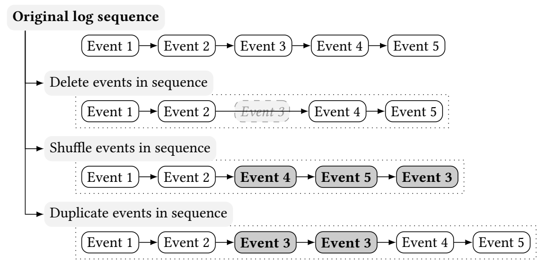
HDFS data: The HDFS dataset is a commonly-used benchmark for log-based anomaly detection [3, 31, 44]. It is produced through running Hadoop-based map-reduce jobs on more than 200 Amazon's EC2 nodes, and labelled by Hadoop domain experts. In total, 24,396,061 log messages are generated from 29 log events. These log messages form different log sequences according to their *block_id*, among which about 2.9% indicate system anomalies. More details of this dataset can be found in [44]. We randomly collect 6,000 normal log sequences and 6,000 anomalous log sequences from the original HDFS dataset as the *Training set.*

The synthetic HDFS data: In order to show the effectiveness of our approach in dealing with the instability of log data, we have created unstable testing datasets based on the original HDFS dataset. We mainly simulate two kinds of log instability as illustrated in Figure 9. We synthesize the unstable log data according to the experience we obtained from our empirical study (as described in Section 2.2). We believe that the synthetic log data can reflect the unstable characteristics of real-world logs.

- Unstable log events: As described in Section 1 Introduction, developers often insert/remove some words when they update a logging statement in the source code. Thus, we will encounter evolving new log events. Furthermore, there are also many pseudo new log events produced from the inaccurate log parsing. We create a set of synthetic log events by randomly inserting/removing a few words into/from the original log events in advance. The synthetic log events do not significantly change the semantic meaning of the original ones, therefore the corresponding anomaly label status is not affected. We inject these synthetic log events into the original log data according to a specific ratio.
- Unstable log sequences: Log sequences are likely to be changed during the process of log evolution or collection. In order to simulate the unstable log sequences, we randomly remove a few unimportant log events (which do not affect the corresponding anomaly status labels) from the original log sequences. We also randomly select an unimportant log event and repeat it several times in a log sequence, or shuffle the order of a few events. We inject these synthetic unstable log sequences into the original log data according to a specific ratio.


Original log event
PacketResponder * for block * terminating
Add words to log event
PacketResponder * for block * terminating time *
Remove words from log event
PacketResponder * ~~for~~ block * terminating

(a) Synthetic log events


Original log sequence
Event 1
Event 2
Event 3
Event 4
Event 5
Delete events in sequence
Shuffle events in sequence
Duplicate events in sequence

(b) Synthetic log sequences

Figure 9: Synthetic Log Examples on HDFS Dataset

To prepare for the synthetic dataset, we randomly collect 51,000 log sequences from the original HDFS dataset consisting of 50,000 normal and 1,000 anomaly sequences. The percentage of anomalies is 2%, which is close to that of the original HDFS dataset. We inject the unstable log data into it and create two testing sets: NewTesting 1 and 2, which contains injected unstable log events and unstable log sequences, respectively. Table 1 summaries the synthetic unstable HDFS datasets.

Table 1: The synthetic HDFS dataset

Set	Unstable event	Unstable seq.	Normal	Anomaly	Total
Training	No	No	6,000	6,000	12,000
NewTesting1	Yes	No	50,000	1,000	51,000
NewTesting2	No	Yes	50,000	1,000	51,000

Microsoft's data: Apart from the public HDFS data, we also collect the real-world industrial log data from Microsoft, termed as Service X dataset. Service X is a geographically distributed, web-based online service serving millions of users. Millions of log sequences (about hundreds of terabytes of raw log message data) are generated online by Service X every day. Service X is a fast-evolving service which is deployed in a week-level frequency and conducts many A/B tests for new features in each deployment. Therefore, the system constantly introduces new execution paths and yields unprecedented log events.

The Service X dataset consists of logging messages collected on two days spanning about one month. The number of log messages in these two sets is 3,178,317 and 5,227,446, respectively. The data is labelled by the support engineers of Service X and contains a small percentage of anomalies. Thus it is an imbalanced dataset. For confidentiality reasons, we do not disclose the number of anomalies here.

4.1.2 Implementation and Environment. The neural network of LogRobust is trained using the Stochastic Gradient Descent (SGD) algorithm [25]. We use a weight decay of 0.0001 with a momentum of 0.9 and set the initial learning rate to 0.01. We use the cross-entropy as the loss function. The size of mini-batches is set to 128. We terminate the training process after 10 epochs. We build our model based on Keras toolbox [8] using an NVIDIA Tesla M40 GPU.

4.1.3 Evaluation Metrics. To measure the effectiveness of LogRobust in anomaly detection, we use the Precision, Recall and F1-Score as metrics. We calculate these metrics as follows:

- *Precision:* the percentage of log sequences that are correctly identified as anomalies over all the log sequences that are identified as anomalies by the model: $Precision = \frac{TP}{TP+FP}$;
- *Recall:* the percentage of log sequences that are correctly identified as anomalies over all abnormal log sequences: $Recall = \frac{TP}{TP+FN}$;
- *F1-Score:* the harmonic mean of *Precision* and *Recall*.

TP is the number of abnormal log sequences that are correctly detected by the model. *FP* is the number of normal log sequences that are wrongly identified as anomalies by the model. *FN* is the number of abnormal log sequences that are not detected by the model.

4.2 RQ1: Experiments on Unstable Log Data

4.2.1 Experiments on the Synthetic HDFS Dataset. We train LogRobust on the original HDFS data, i.e., the *Training set* in Table 1. Then we test the trained model on the synthetic dataset with new log events injected (i.e., the *NewTesting1 set*). We compare the results of LogRobust and four traditional approaches, including SVM [29], LR [3], IM [31] and PCA [44]. As the discussion in Section 2.3, traditional approaches cannot take new log events as input because the dimension of log count vectors is related to the fixed number of original log events. Even if only one new log event appears, these traditional approaches also fail to work. For the sake of comparison, we treat all new log events as an extra special dimension in log count vector as we cannot know the number of new log events in advance. In this way, the related methods could still work on the dataset containing unstable log events.

Table 2: Experiment results on synthetic HDFS dataset of unstable log events (the *NewTesting1 set*)

Injection Ratio	Metric	LR	SVM	IM	PCA	**LogRobust**
5%	Precision	0.25	0.36	0.78	0.90	1.00
	Recall	0.92	0.96	0.56	0.66	0.91
	F1-Score	0.39	0.53	0.65	0.76	**0.95**
10%	Precision	0.18	0.11	0.88	0.90	0.89
	Recall	0.95	0.89	0.40	0.64	1.00
	F1-Score	0.30	0.20	0.56	0.74	**0.94**
15%	Precision	0.08	0.11	0.84	0.82	0.86
	Recall	0.85	0.90	0.41	0.42	0.99
	F1-Score	0.14	0.20	0.55	0.55	**0.92**
20%	Precision	0.06	0.09	0.82	0.82	0.99
	Recall	0.87	0.89	0.43	0.41	0.81
	F1-Score	0.11	0.16	0.56	0.54	**0.89**

The experimental results on the *NewTesting1 set* are shown in Table 2. It can be seen that LogRobust performs much better than other approaches. With the increasing injection ratio of unstable log events, the performance of the related approaches has declined in different degrees. However, LogRobust still maintains a high accuracy even under a high injection ratio. For example, LogRobust can still achieve an F1-Score of 0.89 when the injection ratio is 20% (when 20% of original log events have been replaced by synthetic unstable log events). It confirms that our approach is robust enough to the unstable log events. The reason is that LogRobust is able to capture the semantic information embedded in log events through semantic vectorization, so it can identify unstable log events with similar semantics meaning. Nevertheless, traditional approaches cannot support previously unseen log events thus they are not able to achieve satisfying results.

Table 3: Experiment results on synthetic HDFS dataset of unstable log sequences (the *NewTesting2 set*)

Injection Ratio	Metric	LR	SVM	IM	PCA	**LogRobust**
5%	Precision	0.97	0.94	0.03	0.95	0.99
	Recall	0.85	0.98	0.84	0.65	0.93
	F1-Score	0.96	0.96	0.06	0.77	**0.96**
10%	Precision	0.44	0.77	0.03	0.96	0.94
	Recall	0.93	0.97	0.97	0.63	0.99
	F1-Score	0.61	0.86	0.06	0.76	**0.96**
15%	Precision	0.09	0.21	0.02	0.83	0.98
	Recall	0.88	0.93	0.97	0.39	0.91
	F1-Score	0.17	0.33	0.04	0.53	**0.94**
20%	Precision	0.07	0.07	0.01	0.87	0.92
	Recall	0.82	0.86	0.98	0.37	0.97
	F1-Score	0.12	0.14	0.03	0.52	**0.95**

Secondly, we conduct an experiment on the synthetic HDFS dataset with injected unstable log sequences (the *NewTesting2 set*). We train LogRobust on the *Training set* and use the trained model to detect anomalies in the *NewTesting2 set*. The results are shown in Table 3. LogRobust performs best under all injection ratios. Even when the injection ratio is 20%, i.e., 20% of log sequences suffer from missed, duplicated or shuffled problem, we still get a good

F1-Score of 0.95. The reason is that our approach uses attention-based Bi-LSTM as the classification model. It takes the contextual information embedded in a log sequence into account and learns different importance of log events by the means of attention mechanism. Therefore, it is robust to small variations in the sequences. On the contrary, the related approaches rely on the number of occurrences of log events. If there are variations in the quantitative relationship among the log events, the performance of the detection model will be seriously affected. For example, IM highly depends on the invariants extracted from log sequences, so it is very sensitive to changes in sequences.

4.2.2 Experiment on Microsoft's Industrial Log Dataset. In order to verify the effectiveness of LogRobust in the real-world industrial practice, we perform an experiment on Service X dataset, collected from Microsoft. The Service X dataset consists of logging messages of two days (with one day in June 2018 and one day in July 2018). During this time period, the log statements of Service X were updated weekly. Therefore it suffers from the instability problem. We use the data collected in June as the training set and the data collected in July as the testing set.

Table 4 shows the results. LogRobust achieves satisfactory performance, with an F1-Score of 0.81. Other approaches achieve F1-Score ranging from 0.39~0.52, which are much lower than that of LogRobust. In the real-world scenario, traditional approaches cannot cope with the unstable log data generated from the evolving system and processing noise. LogRobust, however, is capable of maintaining much higher performance, which confirmed its practicality.

Table 4: Results on the Microsoft industrial dataset

Method	Precision	Recall	F1-Score
LR	0.55	0.37	0.44
SVM	0.99	0.26	0.42
IM	0.40	0.39	0.39
PCA	0.43	0.66	0.52
LogRobust	0.69	0.99	**0.81**

4.3 RQ2: Experiment on Attention Mechanism

In this section, we validate the effectiveness of the attention mechanism in LogRobust. We compare the results of standard Bi-LSTM and attention-based Bi-LSTM on synthetic HDFS dataset with injected unstable log sequences, i.e., the *NewTesting2 set*. We illustrate F1-Score in Figure 10.

We can see that, under a low injection ratio, both standard Bi-LSTM and attention-based Bi-LSTM have the similar performance. However, as the injection ratio increases, the advantage of attention-based Bi-LSTM becomes more explicit. For example, under the injection ratio of 40%, the F1-Score of attention-based Bi-LSTM is 0.91, which is much better than the F1-Score of 0.35 achieved by standard Bi-LSTM. The standard Bi-LSTM without attention is greatly affected by the changes of log sequence because it cannot learn the importance of different log events, while the performance of the attention Bi-LSTM remains relatively stable.

4.4 RQ3: Experiments on the Stable Log Data

This RQ evaluates whether or not LogRobust can work effectively with stable log data. We use the original HDFS dataset as the stable

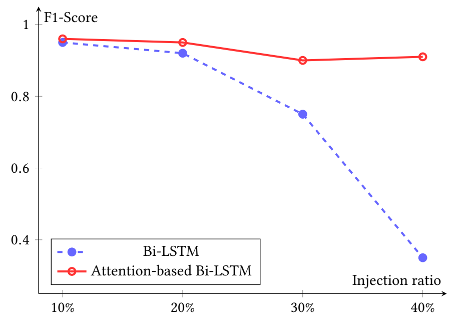


Figure 10: F1-Score of the attention model on synthetic HDFS dataset of unstable log sequences (the *NewTesting2 set*)

dataset. The HDFS dataset is collected from unmodified off-the-shelf Hadoop systems [44] so there is no evolution of logging statements in the source code. In addition, all log events of HDFS dataset are directly identified from the source code [44] thus we can rule out the influence of the processing noise such as parsing errors. We are confident that the original HDFS data can be regarded as a stable log dataset.

We apply LogRobust and related approaches to the original HDFS dataset. We use the same *Training set* as shown in Table 1. The remaining log sequences in the original HDFS dataset are regarded as the testing set, which contains 562,855 log messages in total, among which 10,838 indicate anomalous behavior.

The experimental results are shown in Table 5. The recall, precision and F1-score achieved by LogRobust on this dataset are 1.00, 0.98 and 0.99, respectively. Compared with the two typical unsupervised methods of PCA and IM, LogRobust achieves even better results. LR and SVM are two classic supervised classifiers. They achieve similar precision but lower recall compared to our proposed approach. The results show that LogRobust can work effectively not only on unstable log datasets but also on the stable ones.

Table 5: Results on the stable HDFS dataset

Method	Precision	Recall	F1-Score
LR	0.99	0.92	0.96
SVM	0.99	0.94	0.96
IM	1.00	0.88	0.94
PCA	0.63	0.96	0.77
LogRobust	0.98	1.00	**0.99**

5 DISCUSSION

5.1 Incremental Updating

During the system evolution, the number of accumulated new log events and new log sequences could become larger and larger. This may cause significant performance degradation if training the model on a fixed set of historical data. In LogRobust, we can leverage newly arriving data to maintain the detection performance of LogRobust. Since retraining a model on new data from scratch is costly, LogRobust can be *incrementally updated* to achieve consistently good performance with negligible cost.

Our attention-based Bi-LSTM is trained using a back propagation algorithm [37] and thus it can naturally be updated incrementally through the gradient descent method [25] to fine-tune its parameters automatically. This is also one of the advantages of LogRobust over existing approaches [3, 10, 18, 29, 31, 44]. As described in Section 2.3, the existing approaches are based on the log count vector whose dimension cannot be changed as it is constrained by the number of known log events. Thus they cannot support incremental updating.

We also conduct exploration on the synthetic HDFS dataset under high injection ratio of unstable log events and sequences. The performance of the model also drops significantly when there are very large changes in the dataset (F1-Score from 0.92 to 0.48). However, after incremental updating, the performance of LogRobust can be greatly improved to 0.90.

5.2 Threats to Validity

We have identified the following threats to validity:

- Subjects: in this work, we only use datasets collected from HDFS and Service X system. Although HDFS is a typical open source project and Service X is a large-scale, real-world software system in Microsoft, the number of subject systems is still limited. In the future, we will experiment with LogRobust on a variety of datasets.
- Drastic changes: our approach supports software systems that are under active maintenance, including the scenarios of continuous delivery and deployment. However, if there are sudden, drastic changes to the entire code base or logging mechanism and LogRobust is not updated incrementally, the performance of LogRobust would drop significantly.
- Diversity of changes: in real-world systems, there is a wide variety of changes to log events and log sequences. In our synthetic data creation, we only consider some common types occurred in the studied industrial systems. In our future work, we will explore other possible types of changes.

6 RELATED WORK

6.1 Log-based Anomaly Detection

Logs are widely used in practice for postmortem analysis. There have been a lot of studies on log-based anomaly detection[2, 4, 7, 11, 12, 14, 15, 20, 26, 28, 29, 33–35, 38, 41]. The current approaches are mainly divided into three categories: supervised learning methods, unsupervised learning methods, and deep learning methods.

Many supervised learning methods are applied to log-based anomaly detection. For example, [29] trained an SVM classifier to detect failures using event logs. [7] leveraged the decision tree model to detect anomalies in application operations. [11] proposed a regression-based method to detect anomalies using log data in cloud systems. [3] summarized some classical supervised classification models that are applied to log-based anomaly detection. As described in Section II, these conventional approaches are not robust to unstable log data, which significantly restricts their applicability in real-world practice.

Apart from supervised learning approaches, many unsupervised learning approaches have been proposed. For example, Lou et al. [31] proposed Invariant Mining (IM) to mine the invariants (linear relationships) among log events from log event count vectors. Those log sequences that violate the invariant relationship are considered as anomalous samples. Xu et al. [44] constructed normal space and abnormal space of log event count matrix using Principal Component Analysis to detect anomalies. Lin et al. [30] and He et al.[20] designed clustering-based methods to identify problems of online service systems. Unsupervised learning approaches have an advantage that they do not require manual labels in the training set. However, as shown in this paper, they are not robust enough to the instability of log data either.

The recent rise of deep learning methods has given a new solution for log-based anomaly detection. [46] used LSTM to predict the anomaly of log sequence based on log keys. [10] also used LSTM to forecast the next log event and then compare it with the current ground truth to detect anomalies. [41] trained a stacked-LSTM to model the operation log samples of normal and anomalous events. However, their input of neural networks is the one-hot vector of log events. Thus it cannot cope with evolving log data, especially in the scenario when new log events appear. Some studies have leveraged NLP techniques to analyze log data based on the idea that log is actually a natural language sequence. [46] proposed to use LSTM model and TF-IDF weight to predict the anomalous log messages. [1] used word2vec and traditional classifiers, like SVM and Random Forest, to check whether a log event is an anomaly or not. [5] combined various attention-based models and word vector to detect anomalous log events. However, these approaches only focused on the granularity of log events rather than log sequences. They ignored the contextual information in log sequences. In our work, we take the entire sequence into account and train an attention-based Bi-LSTM model on the log sequence rather than a single log event.

6.2 Logging Practice and Log Data Quality

Recently, much research has been devoted to the logging practice and log data quality [13, 16, 17, 45]. For example, Zhu, et al. [13] have performed an empirical study of logging practice in Microsoft. They found that developers could adopt different ways of performing logging and there are no "ground truth" logging methods. Yuan, et al. [45] reported the characteristics of logging modifications by investigating the revision histories of open-source software systems. He, et al. [17] analyzed the quality issues of log parsers. They observed that many existing log parsers are not sufficiently accurate or efficient, which make them ineligible for log parsing in modern systems [17]. Moreover, the predefined parameters required by the parsers limit the robustness of the online parsers against the logging statement updates.

7 CONCLUSION

Over the years, many log-based approaches have been proposed to detect anomalies of large-scale software systems [3, 10, 18, 29, 44]. However, the existing approaches cannot handle the instability of log data, which comes from the evolution of logging statements and the processing noise. To overcome this problem, in this paper, we propose a new log-based anomaly detection approach, called LogRobust. Our approach represents a log event as a fixed-dimension semantic vector and utilizes an attention-based Bi-LSTM classification model to detect anomalies. We have evaluated the proposed approach using both synthetic public log data and real-world industrial data from Microsoft. The experimental results confirm the effectiveness of LogRobust.

REFERENCES

[1] Christophe Bertero, Matthieu Roy, Carla Sauvanaud, and Gilles Trédan. 2017. Experience Report: Log Mining using Natural Language Processing and Application to Anomaly Detection. In *Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on*. IEEE, 351–360.

[2] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: automated classification of performance crises. In *Proceedings of the 5th European conference on Computer systems*. ACM, 111–124.

[3] Jakub Breier and Jana Branišová. 2015. Anomaly detection from log files using data mining techniques. In *Information Science and Applications*. Springer, 449–457.

[4] Jakub Breier and Jana Branišová. 2017. A dynamic rule creation based anomaly detection method for identifying security breaches in log records. *Wireless Personal Communications* 94, 3 (2017), 497–511.

[5] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. 2018. Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection. *arXiv preprint arXiv:1803.04967* (2018).

[6] Lianping Chen. 2015. Continuous delivery: Huge benefits, but challenges too. *IEEE Software* 32, 2 (2015), 50–54.

[7] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *null*. IEEE, 36–43.

[8] François Chollet et al. 2015. Keras. https://keras.io.

[9] Bogdan Dit, Latifa Guerrouj, Denys Poshyvanyk, and Giuliano Antoniol. 2011. Can Better Identifier Splitting Techniques Help Feature Location. In *2011 IEEE 19th International Conference on Program Comprehension*. 11–20.

[10] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 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*. ACM, 1285–1298.

[11] Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, and John Grundy. 2015. Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*. IEEE, 24–34.

[12] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 149–158.

[13] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. 2014. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 24–33.

[14] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. LogMine: fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 1573–1582.

[15] Mehran Hassani, Weiyi Shang, Emad Shihab, and Nikolaos Tsantalis. 2018. Studying and detecting log-related issues. *Empirical Software Engineering* (2018), 1–33.

[16] Pinjia He, Zhuangbin Chen, Shilin He, and Michael R Lyu. 2018. Characterizing the Natural Language Descriptions in Software Logging Statements. (2018).

[17] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu. 2016. An Evaluation Study on Log Parsing and Its Use in Log Mining. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 654–661. https://doi.org/10.1109/DSN.2016.66

[18] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. 2017. Towards Automated Log Parsing for Large-Scale Log Data Analysis. *IEEE Transactions on Dependable and Secure Computing* (2017).

[19] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40.

[20] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying Impactful Service System Problems via Log Analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, 60–70. https://doi.org/10.1145/3236024.3236083

[21] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991* (2015).

[22] Jez Humble and David Farley. 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

[23] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[24] Suhas Kabinna, Cor-Paul Bezemer, Weiyi Shang, Mark D. Syer, and Ahmed E. Hassan. 2018. Examining the stability of logging statements. *Empirical Software Engineering* 23, 1 (01 Feb 2018), 290–333. https://doi.org/10.1007/s10664-017-9518-0

[25] Jack Kiefer, Jacob Wolfowitz, et al. 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics* 23, 3 (1952), 462–466.

[26] Christopher Kruegel and Giovanni Vigna. 2003. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 251–261.

[27] Yann Lecun, Yoshua Bengio, and Geoffrey E Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[28] Tao Li, Yexi Jiang, Chunqiu Zeng, Bin Xia, Zheng Liu, Wubai Zhou, Xiaolong Zhu, Wentao Wang, Liang Zhang, Jun Wu, et al. 2017. FLAP: An end-to-end event log analysis platform for system management. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1547–1556.

[29] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE, 583–588.

[30] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 102–111.

[31] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection.. In *USENIX Annual Technical Conference*. 23–25.

[32] Adetokunbo Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2012. A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions on Knowledge and Data Engineering* 24, 11 (2012), 1921–1936.

[33] Leonardo Mariani and Fabrizio Pastore. 2008. Automated identification of failure causes in system logs. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*. IEEE, 117–126.

[34] Karthik Nagaraj, Charles Killian, and Jennifer Neville. 2012. Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 26–26.

[35] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H Chin, and Sumayah Alrwais. 2015. Detection of early-stage enterprise infection by mining large-scale log data. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE, 45–56.

[36] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[37] Robert J Schalkoff. 1997. *Artificial neural networks*. Vol. 1. McGraw-Hill New York.

[38] Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. 2008. SALSA: Analyzing Logs as StAte Machines. *WASL* 8 (2008), 6–6.

[39] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 785–794.

[40] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *IP Operations & Management, 2003.(IPOM 2003). 3rd IEEE Workshop on*. IEEE, 119–126.

[41] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. 2017. Long short-term memory based operation log anomaly detection. In *Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on*. IEEE, 236–242.

[42] Wei Xu. 2010. *System problem detection by mining console logs*. Ph.D. Dissertation. UC Berkeley.

[43] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Largescale system problem detection by mining console logs. *Proceedings of SOSPâĂŽ09* (2009).

[44] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 117–132.

[45] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. 2012. Characterising Logging Practices in Open-Source Software. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*.

[46] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach.. In *BigData*. 1291–1300.

[47] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2015. Learning to Log: Helping Developers Make Informed Logging Decisions. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 415–425. http://dl.acm.org/citation.cfm?id=2818754.2818807

[48] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2018. Tools and Benchmarks for Automated Log Parsing. *arXiv preprint arXiv:1811.03509* (2018).