**ABSTRACT**

Logs are widely employed in modern systems to record critical information and serve as an important source for anomaly detection, which has attracted increasing research interests. However, logs usually suffer from perturbations and it makes the existing log-based anomaly detection methods unstable. In this paper, we aim to solve this problem from the perspective of contrastive learning, by which the intrinsic and robust representations of logs are learned for anomaly detection. We propose two data augmentation methods to generate different views at different granularity for log data and design a deep hierarchical contrastive model for anomaly detection. In the contrastive semantic embedding module, we fine-tune a language model with a message-level contrastive loss. And in the contrastive anomaly detection module, we apply a sequence-level contrastive constraint to assist the detection model to learn robust embeddings for log sequences. Experiments on three datasets verify the effectiveness of our proposed method.

**Index Terms**— Logs, Anomaly Detection, Contrastive Learning

1. INTRODUCTION

Modern software-intensive systems are becoming much more large-scale and complex, which makes anomaly detection an indispensable task to help maintain reliability and stability. These systems usually produce copious logs, and they are typically present in the form of semi-structured natural languages, where critical information describing current circumstances is wrote down. When a failure occurs, the operators can refer to the logs to track the suspicious events, analyze the underlying causality and find the possible root cause [1, 2].

Most of the existing approaches [3, 4, 5, 6] for log-based anomaly detection make an assumption explicitly or implicitly that logs are generated by the system following a predefined fixed paradigm [7]. That is, the log statements usually keep stable and the logs are regularly printed along with the execution flows. However, this assumption cannot always be satisfied due to the challenges from two aspects:

**Perturbations in the log messages** According to a previous empirical study [8, 7], the percentage of the ever-changed log statements is about $20\% \sim 45\%$. These changes can be induced by the updates of the log templates. For example, words can get inserted or appended in the description statements to make the logs more comprehensive, or some words may be replaced or removed with the business adjustments.

**Perturbations in the log sequences** As modern systems are becoming much more sophisticated, it often entails different modules or services collaborating at the same time. The log sequences produced by different modules are interleaved in a common log file. The perturbation in the log sequence can be caused by the delay or advance of sub-tasks in some modules. The existing works which model the log sequence to discover the latent sequential patterns will flag an anomaly for such situation. However, it may bring about false alarms if these tasks are not related at all.

In this paper, we propose logContrast, a robust log-based anomaly detection model based on contrastive learning, aiming to learn the intrinsic representations of logs and log sequences for the anomaly detection task. Firstly, inspired by the contrastive framework [9], we propose two methods of data augmentations to generate multiple views of different granularity, i.e., log-level and sequence-level. Specifically, to handle the perturbations in log messages, we augment the individual logs using three types of operations. To address the perturbations in the log sequences, we also propose to augment the log sequences in three ways, random repetition, random deletion, and random shuffling. Then we design a contrastive deep neural network to model the log sequence characteristics to determine whether a log sequence is an anomaly. We evaluate logContrast on three public real-world log datasets. The results demonstrate its effectiveness and robustness in the anomaly detection task.

2. BACKGROUND AND PRELIMINARIES

Logs are generated by large-scale systems to record critical
information of runtime, which includes but is not limited to timestamps, verbosity level, events, and parameters of execution details. As shown in Fig.1, a raw log message is made up of a series of tokens and can be seen as a piece of semi-structured text where certain information is arranged, following a predefined way. Then a log sequence is a series of log messages that belong to a same-time interval or are denoted by a common task identifier.

**Log parsing** aims to separate raw log messages into a constant part and a variable part [10, 11] in a meaningful manner. The constant part contains a group of keywords that represent the template of a log message, which is also called log event, and the variable part is comprised of log parameters recording some attribute information like IP address. The bottom part of Fig.1 shows an example of the parsing results of logs from the upper part. It can be seen that the parameters are removed and only the key tokens are retained. Besides, it can also be found that different log messages can have the same log events after log parsing, such as log messages M3 and M4. Traditional log-based anomaly detection methods such as PCA [12], Invariants Mining [13] and some deep methods like DeepLog [14] and logAnomaly [15] require log parsing to obtain log templates/log events as a preprocessing step. In recent decades, log parsing has been widely investigated and a lot of log parsing approaches are proposed from different aspects [11, 16], and people can refer to [10] for a comprehensive survey.

### 3. PROPOSED METHOD

#### 3.1. Preprocess Module

The preprocessing procedure mainly consists of three steps.

**Tokenization** We tokenize each raw log message into a list of tokens with three steps inspired by [17]. First, a log message is split into candidate tokens with common delimiters like white space or punctuation. Then all the tokens containing number digits or special symbols are removed, as they usually are non-informative parameters and notations. Finally, we replace each capital letters in the remaining tokens with corresponding lowercase letters. We illustrate this procedure with an log message from a dataset BGL [18]. With tokenization, a raw log message "1117842974 2005.06.03 R24-M0-N1-C:J13-U11 2005-06-03-16.55.309974 R21-M0-N1-C:J13-U11 RAS KERNEL INFO 162 double-hummer alignment exceptions" is transformed into token list "ras, kernel, info, double, hummer, alignment, exceptions".

**Log Augmentation in log-level** To learn robust semantic embeddings for log messages, we apply contrastive learning on the vectorization process, which requires log data from different views. As shown in Fig.3 (a), three data augmentation techniques are introduced to generate different views of log messages, i.e., random deletion, random addition, and random replacement, where the items in the token list are changed randomly.

**Log Augmentation in sequence-level** We also generate different views for contrastive learning at the granularity of log sequence, as shown in Fig.3 (b). The first is random deletion of log messages from the original log sequences. The second is random repetition, for which we randomly select a log message and repeat it several times in its belonging sequence. The third is random shuffle, that is, we randomly select a segment of the log sequence and shuffle the log messages in it while the other log messages maintain stable.

### 3.2. Contrastive Semantic Embedding Module

#### 3.2.1. Log Message Encoding

A log message can be considered as a sentence comprising a line of words, and we adopt the language model to extract the semantic embedding from it. As a powerful deep representation model, BERT has been pre-trained on a huge natural language corpus and has shown its great learning ability. In this paper, we follow the practice in [17] and employ the BERT base model [19] for log message encoding. We use the average of all token embeddings in a log message as the embedding of the log message. Given the $i$-th log message $x_i$, it is encoded using BERT as $x_i = \text{BERT}(x_i)$.

#### 3.2.2. Fine-tune BERT with Contrastive Learning

To make the semantic embedding more robust, we fine-tune the BERT model through a contrastive loss computed on the embeddings of log messages from different views. The em-
3.3.1. Log Sequence Encoding

We employ the classic transformer [20] to encode a sequence of log messages. Given a sequence of log message \( S_i = \{x_1, \ldots, x_{|S_i|}\} \), it is first input in the fine-tuned BERT encoder to get the semantic embeddings \( \{x_1, \ldots, x_{|S_i|}\} \). To capture the context information of a log message in the sequence, we also add positional embeddings \( \{p_1, \ldots, p_{|S_i|}\} \) before putting them into the transformer layers. The procedure can be defined as \( s_i = \text{Transformer}(x_1 + p_1, \ldots, x_{|S_i|} + p_{|S_i|}) \).

Anomaly detection can be regarded as a classification problem with two target classes, anomaly and normal. With log sequence embeddings \( \{s_1\} \), a softmax layer is appended to compute probability \( p_{i,f} \) of each log sequence \( S_i \) on each class \( f \). We denote the set of log sequences with labels as \( S_L \) and use the cross-entropy loss to construct the classification objective as follows:

\[
\mathcal{L}_d = - \sum_{S_i \in S_L} \sum_{f \in F} Y_{i,f} \ln p_{i,f} ,
\]

where \( Y_{i,f} = 1 \) denotes that log sequence \( S_i \) actually belongs to class \( f \), and otherwise \( Y_{i,f} = 0 \).

3.3.2. Sequence-level Constraint

To handle the perturbations in the log sequences, we also implement a contrastive constraint at the sequence level. Given a series of log sequence \( S^u_1, \ldots, S^u_{N_u} \), where \( u \in \{OS, RD, RR, RS\} \) corresponding to the sequence-level view of original log sequence, random deletion, random repetition and random shuffle respectively, their embedding by transformer are \( s_1^u, \ldots, s_{N_u}^u \). Similarly, we define the loss of the sequence-level contrastive loss as follows:

\[
\mathcal{L}_{u_i, u_j} = - \sum_{u_i \in \{RD, RA, RR\}} \mathcal{L}_{OS, u_i},
\]

where \( N_2 \) is the number of negative samples. And the final total loss for the contrastive anomaly detection module is \( L = \mathcal{L}_d + \mathcal{L}_{cs} \).
Table 2: Performance comparison anomaly detection on original datasets.

<table>
<thead>
<tr>
<th>Methods</th>
<th>HDFS</th>
<th>BGL</th>
<th>Thunderbird</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>F1-Score</td>
</tr>
<tr>
<td>PCA</td>
<td>0.9710</td>
<td>0.6280</td>
<td>0.7630</td>
</tr>
<tr>
<td>IM</td>
<td>0.8950</td>
<td>0.9440</td>
<td>0.9450</td>
</tr>
<tr>
<td>DeepLog</td>
<td>0.9340</td>
<td>0.9950</td>
<td>0.9640</td>
</tr>
<tr>
<td>LogRobust</td>
<td>0.8600</td>
<td>1.0000</td>
<td>0.9250</td>
</tr>
<tr>
<td>LogContrast</td>
<td>0.9798</td>
<td>1.0000</td>
<td>0.9898</td>
</tr>
</tbody>
</table>

Table 3: Performance comparison anomaly detection on noise datasets.

<table>
<thead>
<tr>
<th>Methods</th>
<th>HDFS</th>
<th>BGL</th>
<th>Thunderbird</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>F1-Score</td>
</tr>
<tr>
<td>PCA</td>
<td>0.9586</td>
<td>0.6063</td>
<td>0.7426</td>
</tr>
<tr>
<td>IM</td>
<td>0.0146</td>
<td>1.0000</td>
<td>0.0288</td>
</tr>
<tr>
<td>DeepLog</td>
<td>0.0078</td>
<td>0.3829</td>
<td>0.0153</td>
</tr>
<tr>
<td>LogRobust</td>
<td>0.2764</td>
<td>0.1418</td>
<td>0.1874</td>
</tr>
<tr>
<td>Logsy</td>
<td>0.0340</td>
<td>0.1936</td>
<td>0.0578</td>
</tr>
<tr>
<td>LogContrast</td>
<td>0.9125</td>
<td>0.8670</td>
<td>0.8892</td>
</tr>
</tbody>
</table>

Table 1: The details of datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>#Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>Hadoop distributed file system log</td>
<td>11,175,629</td>
</tr>
<tr>
<td>BGL</td>
<td>Blue Gene/L supercomputer log</td>
<td>4,747,963</td>
</tr>
<tr>
<td>Thunderbird</td>
<td>Thunderbird supercomputer log</td>
<td>5,000,000</td>
</tr>
</tbody>
</table>

4.2. Overall Performance

Table 2 gives the anomaly detection results on the original non-noise datasets. Note that for all baselines on original HDFS datasets, we directly report the results from a comprehensive survey [22] without rerunning. As we can see, proposed logContrast constantly gives the best performance on all three datasets. It demonstrates that even for non-noise circumstance, applying the contrastive constraints can benefit the log-based anomaly detection task. Table 3 shows the results on the synthetic data where noise are added randomly. Mostly methods suffer from a drastic drop when confronting perturbations, however, the proposed logContrast still holds a pretty good performance. That is because compared with other method, logContrast can learn more intrinsic embeddings for logs and be more robust to perturbations.

4.3. Ablation Study and Parameter Analysis

We denote the variant with only log-level contrastive learning as logContrast-L and the variant with only sequence-level as logContrast-S, and apply them on the BGL respectively. As shown in Fig.4 (a), both modules play important role in robust anomaly detection. We also experiment on the noise BGL dataset with different noise ratio, the results of which are shown in Fig.4 (b). With the increase in noise ratio, the performance shows a trend of decline.

5. CONCLUSION

In this paper we propose logContrast for robust log-based anomaly detection. To handle the perturbations in log data, we design different data augmentation methods for logs and learn robust representations with a designed deep hierarchical contrastive model. Experiments on multiple datasets exhibit the effectiveness and robustness of the logContrast.

6. ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under grant 61972270.
7. REFERENCES


