Unsupervised Anomaly Detection Based on System Logs

Hao Chen, Ruizhi Xiao and Shuyuan Jin*
School of Data and Computer Science, Sun Yat-sen University, Guangzhou,China
Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, China
{chenh529, xiaorzh3}@mail2.sysu.edu.cn, jinshuyuan@mail.sysu.edu.cn

***Abstract*—The anomaly detection based on rich and descriptive system logs is critical to securing information systems. Existing techniques rarely consider semantic information of logs in the detection, resulting in their incapability to handle unseen log events, neither further improve their detection rates. This paper proposes a CNN and LSTM based anomaly detection approach. It utilizes the meaning of log entries — the semantic information of logs in the detection, where the relations among short sequences are automatically learned. The results of comparative experiments demonstrate the effectiveness of the proposed approach on both stable(fixed format) and unstable(unseen, unfixed format) logs.**

***Index Terms*—anomaly detection, log analysis, deep learning**

I. Introduction

Large-scale systems, such as cloud-based online service systems, play a core role in a various critical industries. However, as the large-scale systems get increasingly larger and more complex than ever before, they face more security threats. Small system anomalies could lead to huge losses. In order to build a secure system, accurate and timely anomaly detection is needed.

System log is an important resource for troubleshooting and diagnosis of systems. Since system log contains rich and descriptive information, such as timestamp, system states and events, engineers can examine recorded logs for anomaly detection. Traditionally, engineers search keyword in system logs to find indicative anomalies, such as 'error', 'fail', etc. However, in lager-scale system, a great number of logs are generated every day. It is impracticable and time-consuming to diagnose problems through logs manually. Furthermore, both normal and abnormal logs contain words such as 'fail' and 'error'. Therefore, using only keywords search will cause many false positives.

In recent years, many automated log-based anomaly detection approaches have been proposed [1]–[5] , some methods [2], [3], [6] treat anomaly detection as a binary classification. In general, these approaches convert logkey(log event indexes) sequences into log count vectors, and then apply data mining methods to detect anomalies. With the prevalence of deep learning, those methods [1], [7] have widely used to detect anomalies. However, the above methods ignore the semantic meaning in the log sequences, and not consider the instability in real-word log. In recent studies, some semantic-based methods have been published [8], [8]–[10]. LogAnomaly [9] proposes the template2vec method, which encodes log template to a semantic vector, and utilizes LSTM to train anomaly detection model. Even though they get good results in certain scenarios, the performance of their methods could be further improved.

The log based anomaly detection faces the following challenges.

1) Semantic meaning of system logs is not utilized well in the detection. Some existing detection techniques do not take semantic information into consideration, while others only calculate semantic information based on all words in the log entry, ignoring that some words have no semantic meanings or even reduce real meanings to some extends. For example, an entry in HDFS logs is *"10.250.11.100:50010 Served block blk_-3544583377289625738 to /10.250.19.10"*. It contains words 'served block to', where the word 'to' has little meanings in fact.

2) Log is unstable in real-word [10], in other words, many unseen logs are existing in the information systems. In the process of software developers insert or delete certain words to update the logging statements, lots of unseen log events will be generated. Normally the existing techniques only use known logkeys and ignore unseen log events, where unseen log events is only treated as new events. Therefore, they fail to work with unseen log events and cause many false positives.

To address the above challenges, we propose an unsupervised anomaly detection method, which can detect anomalies automatically and achieve high detection accuracy. The proposed method encodes logs into fixed-length semantic vectors. Two logs with similar meanings have similar semantic vectors. Note that a LSTM itself cannot extract enough features to make a good detection, the proposed method employs a CNN and LSTM combined model to detect anomalies. It has the advantages of automatically extracting log short sequence relationship and log quantitative features simultaneously.

We evaluated our proposed method on stable and unstable log dataset. The experimental results show that the proposed approach outperforms the state-of-art techniques with 98% F-measure and only uses 1% normal datasets for training. We also evaluated the proposed method on unstable log dataset. The experimental results demonstrate the effectiveness of the proposed method.

DOI reference number: 10.18293/SEKE2021-126

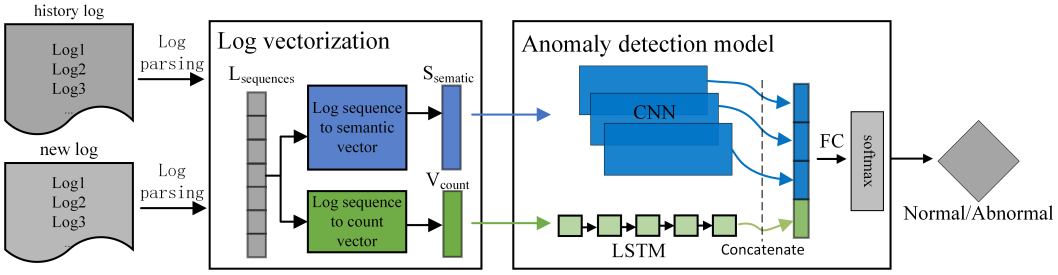

Fig. 1. The overview of the proposed method

The main contributions of this paper are as follows:

- A novel semantic information embedding technique is proposed to detect anomalies in system logs. The basic idea of extracting semantic information in the detection comes from that system administrators usually use keyword search to find anomalies. Thus, some keywords in the log entries may represent the meanings of the entire log entries.
- A CNN and LSTM combined detection approach is proposed. It has the capabilities of not only learning semantics from system logs, but also learns the quantitative feature from log count vector. It can achieve high detection rates on both stable and unstable logs.

The rest of this paper is organized as follows: Sec. II introduces related work. Then, Sec. III describes the proposed method, and the experiment results are discussed in section IV. Finally, in Sec. VI we conclude our work.

II. Related Work

Logs contain abundant information, such as computer status, making them a valuable resource for anomaly detection. A significant amount of studies have been published in log-based anomaly detection [4], [5]. The current methods are mainly divided into four categories: rule-based methods, data mining-based methods, deep learning-based methods and NLP-based methods.

Rule-based methods use keywords or regular expressions to find anomalies in logs. Marcello [5] proposes a rule-based methods to analyze software failures from logs. Rule-based methods have high accuracy, but they require domain expertise and are very time-consuming in production environment. In the past few years, data mining and deep learning methods have been proposed.

There are many methods based on data mining [2], [3], [6]. These methods generally first parse log messages into logkey. Then they convert the logkey sequences into a log count vectors, and finally apply data mining methods to detect anomalies. Data mining methods can be divided into unsupervised methods and supervised methods. Compared with unsupervised methods, supervised methods have better results. However, supervised method requires a large quantity of labeled data, which need massive manual effort in large system. Furthermore, in real environments, abnormal data is rare compared with normal datasets, which make it impractical for supervised method training. Therefore, our method only uses normal logs for training.

In recent years, deep learning-based methods have been widely studied [1], [7]. Deeplog [1] uses normal datasets to train an anomaly detection model. It uses LSTM to predict the next logkey and compares it with the actual logekey to detect anomalies. These methods do not consider the semantic information in log entries, and they have poor performance on the unseen log data. In real-word log data, logs are unstable because the evolution of logging statements and noise generated during log data pre-processing [10]. For example, developers often add or delete certain words when they update a logging statement. Many unseen log events will be produced. Existing approaches have poor performance on this issue due to the wrong classification. DeepLog uses a user feedback mechanism to update anomaly detection models. However, it needs lots of manual feedback which is infeasible in real-time systems. We propose a method to handle with unseen log events without the need of manual feedback.

The latest studies [8]–[10] use NLP techniques to analyze log-based anomaly detection. LogAnoamly [9] extracts the semantic information form log, and then use LSTM to train the anomaly detection model. LogRobust [10] can also represent log message as semantic vectors, and utilizes the attention-based Bi-LSTM model to detect anomalies. However, these methods only use LSTM to train anomaly detection models. Our method combines CNN and LSTM, which can extract the richer features.

III. Method

A. Overview

The overview of the proposed method is shown in Fig. 1, which mainly includes two parts, log vectorization and anomaly detection model. First, we leverage Drain [11] to parse the log into logkey and group logkey by identifiers(such as block_id in HDFS dataset). After that, the proposed method does not rely on logkey sequences($L_{sequences}$) for anomaly detection like exiting methods. Instead, it encodes logkey sequences into semantic vector sequences($S_{semantic}$) and log count vector(V_{count}). Then, we propose an anomaly detection model that can detect semantic vector sequences and logkey count vectors simultaneously. The model has the ability to extract short sequence relationship among log entries, which is suitable for analyzing log sequences data. In the end, an

alarm message could be sent to system administrator if an anomaly is detected.

B. Semantic Embedding

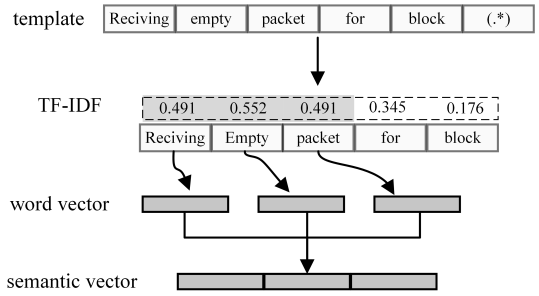

Fig. 2. The workflow of semantic vectorization

The proposed method can extract semantic information from log entries and encode each logkey into a fixed-dimension vector. The design of converting logkey into a semantic vector is based on the assumption that some keywords in a log entry can represent the meaning of whole log entry. For example, a log entry *"081109 203521 145 INFO dfs.DataNode$DataXceiver: Receiving block blk_-3544583377289625738 src: /10.250.19.102:39325 dest: /10.250.19.102:50010"* from HDFS dataset, which contains rich information, such as timestamp, pid, IP address , block_id etc. The meaning of this log entry can be represented as two keywords "receiving block". The manual effort to find keywords is unpractical in large system, so we need an automatic method to determine which words in a log entry are keywords and how to convert the keywords to a semantic vector. As shown in Fig. 2, the workflow of semantic embedding consists of three phases, keyword search, word vectorization and concatenation.

1) Keywords Search: Get keywords from the log template. First we need pre-processing. In this step we filter non-words(such as '*' ,':') in the template entry, and split the template into individual word. Then, we use TF-IDF [12] to calculate the importance of each word in the template entry, which can effectively measure importance of words in a document. For each word in template entry, its TF-IDF weight is calculated by TF*IDF. The term frequency(TF) represents the frequency of words in the template entry. For example, the word 'block' appears multiple times in the template, which means that word 'block' has a high TF weight. However, if the word 'block' appears in all template entries, it means that it can not distinguish between those template entries, so its weight should be reduced. Therefore, we also calculate the inverse document frequency(IDF). If the word appears multiple times in the template entry, it has low IDF weight. We calculate words importance by the following formula.

$$TF = \frac{N_{word}}{N_{word\_total}} \tag{1}$$

$$IDF = log(\frac{N_{template}}{N_{template\_toatl}}) \tag{2}$$

$$TF - IDF = TF * IDF \tag{3}$$

N_{word} represents the number of target word in the log template. N_{word_total} is the total number of word in a log template. $N_{template}$ is total the number of log template containing target words. $N_{template\_toatl}$ is the total number of log template. After getting the TF-IDF weight of each word, we sort the word importance according to their TF-IDF weight. And we set the parameter *g*, which represents the number of keywords used. As the Fig. 2 shows, *g=3* and we can get top 3 keywords in the template.

2) Word Vectorization: In this phase, we convert each word in log into a semantic vector. We use Glove [13] algorithm encodes each word to a word semantic vectorwhich map each word to a fixed dimension vector. Two words that are semantically close have similar word semantic vector.

3) Concatenation: As we get keywords and word semantic vector, we can convert log into a semantic vector. As the Fig. 2 shows, we convert the keywords into word vectors according to the Glove model, and then we concatenate the word vectors to get semantic vector. The dimension of semantic vector is *w*g*, in which *w* is the word semantic dimension and *g* is the number of keywords.

C. Model

Our anomaly detection model combines CNN and LSTM. CNN is used for capture patterns from semantic vector sequences(a list of semantic vector), because our CNN model has strong ability to extract the short sequence relationships. As for LSTM [14], it can learn the quantitative patterns from log count vector. The combination of CNN and LSTM model can improve the accuracy, which is demonstrated in Sec. VI.

Due to the fact log is a kind of text, it can take a benefit of natural language processing (NLP). Our CNN model refers to the sentence classification model in the NLP field [15]. The CNN neural network contains several layers. The input layer we use semantic vector sequence as input, which is a *m*n* matrix, where *m* represents the size of the sliding window, and *n* represents the dimension of the semantic vector. The next layer is CNN convolutional layers, which is the core layer of CNN. It uses three different one-layer filters to convolute over the input layer. These filters have same width but different heights. The filters width is the same as the dimension of semantic vector, so these filters can only move in the height direction. And then we use 1-max-pooling layers to obtain the maximum feature from feature vector. In this way, we get the semantic vector sequence feature.

LSTM is a kind of recurrent neural networks that designed for sequential data. Thus, we use LSTM neural network to capture quantitative patterns from log count vector. The LSTM neural network consists of an input layer, a hidden layer and an output layer. The LSTM unit to calculates the new state and output uses the input data and the previous unit state. A series of LSTM unit form an LSTM neural network.

Finally we concatenate the CNN and LSTM feature map, and add a softmax function in the output layer. The softmax function outputs the probability of the next logkey.

D. Detection

The proposed method uses the normal execution path to train the anomaly detection model. If the log execution path deviates from the model prediction, we can send an alarm message. The Logkey sequence represents the execution path of log. Let k=*(k_1,k_2,k_3 ... k_n)* as the whole set of distinct logkey and there are *n* different logkeys. The main idea is using the most recent *m* logkey to predict the next *m+1* logkey, so we treat anomaly detection as a multi-classification problem, where each logkey is a class. We use a sliding window of *m* to split the logkey sequence. A subsequence is obtained as *(k_j,k_{j+1} ... k_{j+m-1})*, The next logkey is k_{j+m}. For example, there has a logkey sequence *(1,2,3,4,5)*. First we set a sliding window size as 3. Dividing the log sequence according to the sliding window, we can get logkey subsequences as *(1,2,3),(2,3,4)* and their next logkey is *4,5* respectively.

The proposed method converts logkey sequence to semantic vector sequence and log count vector. The log count vector is wildly used in anomaly detection. It represents the number of occurrences of each logkey in a sliding window. The entire logkey can be expressed as k=*(k_1,k_2,k_3 ... k_n)* and there have *n* distinct logkey. Thus, the log count vector is *n* dimension. Log count vector is denoted as *(c_1,c_2 ... c_j ... c_n)*, where c_j is the number of *j*-th logkey in the sliding window. Finally, the proposed method inputs the semantic vector sequence and log count vector into the anomaly detection model.

In anomaly detection, taking the most recent logkey as input value, the anomaly detection model returns a prediction result, that is, the probability of next logkey. The proposed method select top *g* probabilities as candidates. If the next logkey is not in the top *g* candidates, it can be considered abnormal.

IV. EXPERIMENT

A. Dataset

1) Stable Dataset: We conduct our experiment on stable log dataset HDFS [2] and BGL [16]. Tab. I shows summary of stable log data set.

The HDFS dataset is a benchmark dataset for log anomaly detection. It is generated by Hadoop-based map-reduce jobs on Amazons EC2 with more than 200 nodes. In total, 11,197,954 log messages are collected. Since HDFS data set is labeled by block_id, we use block_id as identifier to group log entries and get 558,221 normal sessions and 16,838 abnormal sessions. The data set is unbalanced, only 2.9% of the datasets are abnormal sessions. The proposed method only uses less than 1% normal sessions for training, which is a grouping of the first 100,000 log entries of original dataset.

BGL is a open source dataset used in Blue Gene/L super-computer system at Lawrence Livermore National Labs. The BGL dataset contains 4,747,963 logs, in which each log is labeled as abnormal or normal, and 348,460 logs are labeled as abnormal. We use a window size of 10 to slice logs into log sequences, and randomly take 20% for training the others for testing.

2) Unstable Dataset: In order to evaluate the robustness of the proposed method. We create unstable log event datasets based on the BGL dataset, with randomly adding or deleting words in the BGL dataset as shown in Fig. 3.

TABLE I
SUMMARY OF STABLE LOG DATA SET

dataset	Train data	Test data
HDFS dataset	4,855 normal 0 abnorma	553,366 normal 16,838 abnormal
BGL dataset	949,592 normal 0 abnormal	3,7983,704 normal 348,460 abnormal

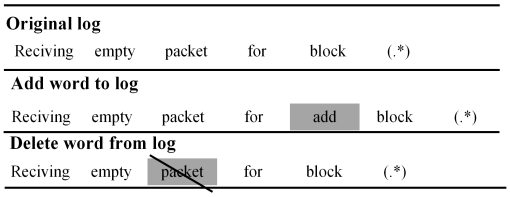

Fig. 3. Synthetic log events

B. Parameter Setting

Our proposed method uses the following parameters: *w*=10, *g*=8, *k*=(2,3,4), *t*=4, *l*=2, *m*=64, *n*=150. *w* is the window size, and *g* is the number of candidates. If the next logkey is in the top *g* of the prediction candidates, it is considered normal. *k* is the convolution kernel of CNN. For example, *k=(2,3,4)* means that there have three different convolution kernels and their heights are 2, 3, 4 respectively. *n* and *t* denote the number of feature maps of CNN and the number of keywords to represent a log entry. *l* and *m* is the number of layers and number of neurons in LSTM. For other methods, we use the parameters with their best results.

C. Evaluation Metrics

In order to measure the prediction accuracy and recall rate, we introduce four indicators: true positive (TP) is anomalous block predicted to be anomalous , false positive (FP) is normal block predicted to be abnormal, true negative(TN) is normal block predicted as normal, and false negative (FN) means a abnormal block predicted to be normal.

The calculation formulas of precision, recall and F-measure are as follow.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F - measure = \frac{2 * Precision}{Precision + Recall} \tag{6}$$

D. Comparison

We compare the proposed method with four unsupervised baseline methods PCA [2], LogCluster [6], DeepLog [1] and LogAnomaly [9].

1) Compared The Proposed Method with Exiting Methods on stable dataset: Fig. 4 shows the comparison of results

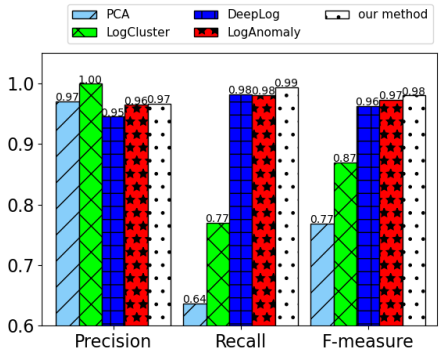

Fig. 4. Evaluation on HDFS

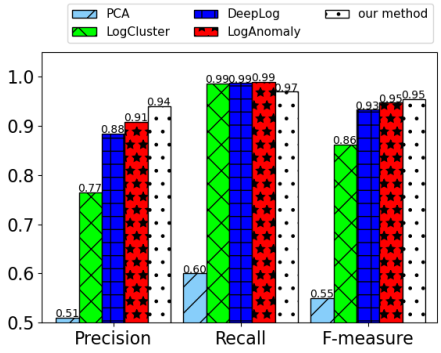

Fig. 5. Evaluation on BGL

of the proposed method and the other four baseline methods on HDFS dataset. Apparently, the proposed method achieves best result, with F-meaure 0.98. Both PCA and LogCluster methods achieves good results on precision, with the price of low recall. The state-of-art method LogAnomaly encodes log template to semantic vector for anomaly detection, and has better performance compared with DeepLog. However, the performance of LogAnomaly could be further improved. The proposed method gets F-measure 0.98, which higher than LogAnomaly with F-measure 0.97.

Fig. 5 shows the performance of our method and other four baseline method on BGL dataset. Among those methods, our method achieves the best F-measure 0.954, and LogAnomaly has an F-measure 0.945 as second. The reason why our method has better performance than LogAnoamly, is as follow. First, LogAnomaly extracts the semantic vector based on all words in the log entry, but our method only considers the keywords in log entry to calculate semantic vector. In addition, LogAnomaly uses LSTM model to extract patterns from logkey sequence, which can not extract log short sequence relationship well. Our proposed method uses a LSTM and CNN combined model that has ability to extract log short sequence relationship.

2) Compared The Proposed Method with Exiting Methods on Unstable Log Events: As the log system evolve and log parsing error, many new log events will be produced. These

TABLE II
EXPERIMENT RESULTS ON BGL DATASET OF UNSTABLE LOG EVENT

Injection Ratio	Metric	LogCluster	DeepLog	the proposed method
0%	Precision	0.77	0.88	0.94
	Recall	0.99	0.99	0.97
	F-measure	0.86	0.93	0.95
5%	Precision	0.68	0.59	0.94
	Recall	0.99	0.97	0.97
	F-measure	0.80	0.74	0.95
10%	Precision	0.62	0.44	0.94
	Recall	0.99	0.99	0.96
	F-measure	0.76	0.61	0.95
15%	Precision	0.63	0.38	0.94
	Recall	0.99	0.95	0.96
	F-measure	0.77	0.54	0.95
25%	Precision	0.62	0.27	0.94
	Recall	0.99	0.96	0.94
	F-measure	0.76	0.42	0.94

methods such as DeepLog and LogCluster cannot effectively handle these unseen log events.

Our method is based on the assumption that most of the new logs are variants of the original logs and will not change the meaning of the original logs, so the proposed method can match the new logs with the original logs significantly. First the proposed method use Drain extract the new log to the template, then search for keywords in the log based on TF-IDF, and finally calculate its similarity by the following formula:

$$similarity = \frac{2 * N_{same}}{N_{new} + N_{exist}} \tag{7}$$

N_{same} means the number of keyword both in new template and exist template. N_{new} means the number of word in new template and N_{exist} means the number of word in exist template. After obtaining the similarity between new template and exist template, the proposed method can match new template to an existing one.

We evaluated the proposed method on unstable BGL log dataset. The experimental results on the unstable log event datasets are shown in Tab. II. The injection ratio represents the ratio of the number of randomly adding and deleting logs to the total number of logs. Note that PCA is omitted from this table because of its very poor performance. Clearly, our method has achieved the best performance. As the injection ratio increases, the performance of our method decreases slowly (F-measure from 0.95 to 0.94). And the performance of the DeepLog method has declined a lot (F-measure from 0.93 to 0.42). LogCluster achieves better F-measure than DeepLog with F-measure declined from 0.86 to 0.76 as the injection ratio increases. The reason is that LogCluster and DeepLog treat unseen log events as a new log events, which may cause false alarms.

E. Discussion

1):Impact of CNN and LSTM in Proposed Method: Our CNN and LSTM combined model can extract patterns from semantic vector sequences and log count vectors simultaneously. Tab. III, demonstrates the impact of CNN and LSTM

TABLE III
THE IMPACT OF CNN AND LSTM IN PROPOSED METHOD

method	Precision	Recall	F-measure
without (w/o) CNN	0.91	0.99	0.95
without(w/o) LSTM	0.96	0.98	0.97
CNN and LSTM	0.97	0.99	0.98

TABLE IV
THE IMPACT OF KEYWORD NUMBER

Number of keywords	Precision	Recall	F-measure
1	0.963	0.986	0.973
2	0.965	0.991	0.978
3	0.963	0.993	0.978
4	0.966	0.993	0.980
5	0.959	0.993	0.975
6	0.960	0.990	0.975

in proposed method. We calculate the precision, recall and F-measure of the proposed method without (w/o) LSTM, and the proposed method without(w/o) CNN. The proposed method without CNN has a much low F-measure, which demonstrates semantic vector sequences are important for the anomaly detection model. The proposed method without LSTM has a lower F-measure. By combining CNN and LSTM model, the proposed method obtains best results.

2):Impact of Keyword Numbers in Proposed Method: In Sec. IV, we describe the semantic vectorization, which can capture the semantic information from log. We want to verify the impact of keyword numbers on the accuracy of the proposed method. We consider 6 distinct numbers from 1 to 6. We follow the same parameters and utilize the same training and testing dataset.

Intuitively, we think that the more keywords are used, the more semantic information will be captured from the log. However, Tab. III shows as the number of keywords increases. The value of F-measure first increases and then decreases, the value of F-measure is the highest when the number of keywords is 4. This is mainly because as the number of keywords increases, some unimportant words(such as 'of', 'the') will also be converted into semantic vectors, which may add noise to semantic vector and reduce the F-measure.

V. CONCLUSION

To address the challenges caused by unstable system logs, this paper proposes a CNN and LSTM based anomaly detection approach. The proposed approach can automatically learn the semantic information among system log sequences and embed the semantic information in the detection. As a result, the proposed approach obtains high detection rates on both stable and unstable logs, in comparison to the existing methods in the experiments.

However, our present method has a major limitations. The proposed approach gets better results than exiting methods, with the price of it takes more time for anomaly detection. One of the future directions of our work is to reduce the time for model anomaly detection.

ACKNOWLEDGMENTS

This work is supported by the Key Research and Development Program for Guangdong Province (Grant No. 2019B010136001) and the National Natural Science Foundation of China (Grant No. 61672494). The corresponding author is Shuyuan Jin.

REFERENCES

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[2] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[3] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection." in *USENIX Annual Technical Conference*, 2010, pp. 1–14.

[4] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 111–124.

[5] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2012.

[6] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 102–111.

[7] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 151–158.

[8] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[9] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 7, 2019, pp. 4739–4745.

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

[11] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[12] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[13] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820*, 2015.

[16] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 575–584.