Computer Systems Science & Engineering
DOI:10.32604/csse.2022.022365
Article

Tech Science Press

LogUAD: Log Unsupervised Anomaly Detection Based on Word2Vec

Jin Wang[1], Changqing Zhao[1], Shiming He[1,*], Yu Gu[2], Osama Alfarraj[3] and Ahed Abugabah[4]

[1]School of Computer and Communication Engineering, Changsha University of Science & Technology, Changsha, 410114, China
[2]Department of Chemistry, Institute of Inorganic and Analytical Chemistry, Goethe-University, Frankfurt, 60438, Germany
[3]Computer Science Department, Community College, King Saud University, Riyadh, 11437, Saudi Arabia
[4]Zayed University, CTI, Abu Dhabi, 144534, United Arab Emirates
*Corresponding Author: Shiming He. Email: smhe_cs@csust.edu.cn
Received: 05 August 2021; Accepted: 17 September 2021

Abstract: System logs record detailed information about system operation and are important for analyzing the system's operational status and performance. Rapid and accurate detection of system anomalies is of great significance to ensure system stability. However, large-scale distributed systems are becoming more and more complex, and the number of system logs gradually increases, which brings challenges to analyze system logs. Some recent studies show that logs can be unstable due to the evolution of log statements and noise introduced by log collection and parsing. Moreover, deep learning-based detection methods take a long time to train models. Therefore, to reduce the computational cost and avoid log instability we propose a new Word2Vec-based log unsupervised anomaly detection method (LogUAD). LogUAD does not require a log parsing step and takes original log messages as input to avoid the noise. LogUAD uses Word2Vec to generate word vectors and generates weighted log sequence feature vectors with TF-IDF to handle the evolution of log statements. At last, a computationally efficient unsupervised clustering is exploited to detect the anomaly. We conducted extensive experiments on the public dataset from Blue Gene/L (BGL). Experimental results show that the F1-score of LogUAD can be improved by 67.25% compared to LogCluster.

Keywords: Log anomaly detection; log instability; word2Vec; feature extraction

1 Introduction

Nowadays, information technology and the economy are growing closer. In order to ensure the sustainable development of the economy, the requirements of robust information systems keep getting higher. Any failure, including service interruption, and transmission quality degradation, may lead to application error, collapse, and even significant economic losses. For example, the securities industry is sensitive to the real-time and continuity of information systems. To enhance the ability of network security emergency response, securities companies have established an automatic emergency switching process to reduce system paralysis time. Once the cause is clearly located, the emergency switching time is controllable. Therefore, identifying the cause or anomaly detection has become a research hotspot [1,2].

At present, information systems have installed a great number of devices and software to monitor the system states, which generate large-scale system logs and key performance indicators (KPI) data [3], such as CPU utilization, query times per second, etc. Abnormal KPI data (e.g., spikes, sudden drops, jitters) can show the abnormal state of the system. However, KPI only can contain limited information. System logs record the detailed operation information. Generally, the cause of fault is also recorded in the system log. Analysis and detection of logs can provide various dimensional information to locate faults. The system log is a rich data source of anomaly detection. Therefore, log anomaly detection has been widely adopted in practice due to its directness and effectiveness.

System logs are generally semi-structured statements and have many different types and formats. Therefore, it is not easy to understand the causes of log anomalies. Initially, developers manually detect anomalies by keyword searching (such as "fail" or "exception") or regular expression matching according to their relevant field knowledge. However, it depends on manual inspection heavily by developers, which is inefficient and high error rate. Therefore, many machine learning methods are applied into log anomaly detection [4,5], such as Logistic Regression (LR), Support Vector Machine (SVM), Principal Component Analysis (PCA), etc. Machine learning methods can save time, reduce the possibility of error, and further improve detection accuracy. However, they do not consider the log instability and still have the following problems:

1) An ideal supervised classification model needs to be trained with a large amount of label data. Generally, the more the amount of label data, the better the classification model. In order to get a good classification model, a large number of label logs are needed, which is hard in practice.
2) The log is unstable. The update and evolution of log statements and the noise introduced by log parsing lead to log unstable, which has a great influence on anomaly detection results.
3) Since there are few label logs, unsupervised methods to detect anomalies in logs are the trend. The detection accuracy of the unsupervised model depends on the feature representation. The original log is often high-dimensional, including a lot of redundant information. Directly training original log data is often inefficient. A suitable feature representation is important for unsupervised model.

Therefore, to overcome the above issues, we propose Log Unsupervised Anomaly Detection (LogUAD), which is an unsupervised anomaly detection method based on Word2Vec. LogUAD does not require a log parsing step and takes original log messages as input to avoid the noise from log parsing. LogUAD uses Word2Vec to generate word vectors and generates weighted log feature vectors to handle the evolution of log statements. At last, a computationally efficient unsupervised clustering is exploited to detect anomalies based on the feature vectors of the log sequence. The contributions can be summarized as follows.

1) We propose a LogUAD anomaly detection model without log parsing, which uses the original log directly for feature extraction and avoids the noise from log parsing on anomaly detection.
2) A word-sequence weight matrix based on the TF-IDF (Term Frequency–Inverse Document Frequency) is used to indicate the relationship between words and log sequences. The feature vector of the log sequence is obtained from word vector generated by Word2Vec and the word-sequence weight matrix, which can handle the evolution of log statements because of the semantic similarity among evaluation of log. It effectively extracts suitable feature for clustering-based anomaly detection.
3) Compared with the LogCluster [6], the F1-score of LogUAD increases by 67.2%. LogUAD improves the accuracy of unsupervised log anomaly detection and can effectively carry out unsupervised anomaly detection.

The rest of this paper is organized as follows. The related work is introduced in Section 2. Section 3 introduces the necessary knowledge about log anomaly detection. Section 4 proposes the LogUAD method. Section 5 describes the experiment in detail. Finally, Section 6 summarizes the work of this paper and future work.

2 Related Work

In practice, system logs are widely used in anomaly detection by developers. There are many works on log anomaly detection. The current main technologies are machine learning and deep learning.

In order to pursue higher detection accuracy, log anomaly detection has applied many supervised learning methods. Liang et al. [7] trained Support Vector Machines (SVM) classifier to detect log events fault. Chen et al. [8] proposed a decision tree method for fault detection and diagnosis of website sites, by sorting and merging the correlation degree of faults to improve the detection accuracy. Farshchi et al. [9] proposed a novel regression-based analysis method. Their experiments on Amazon servers show that this method can solve the problem of operating cloud applications. Zhang et al. [10] proposed PreFix, which applied the random forest algorithm to the template sequence so that it could intervene and "fix" the potential failures. Bertero et al. [11] mapped the logfiles' words to a high-dimensional metric space and then used a standard classifier to check whether the log event is abnormal. However, a large number of label logs are necessary to train unsupervised models and detect anomalies.

Afterward, many unsupervised learning methods were proposed. The advantage of unsupervised learning is without labels. Lou et al. [12] presented Invariants Mining (IM). IM mines the invariants of sparse integer values from the message count vector. Experiments on Hadoop and CloudDB show that IM can detect more and more complete anomaly messages from the distributed systems. Xu et al. [13] used principal component analysis (PCA) to detect abnormal events and found that PCA with term weighting technology in information retrieval got good anomaly detection results without too much parameter adjustment. Yang et al. [14] proposed LogOHC, which had high extraction efficiency for multi-source log datasets. It is an online log template extraction method with an online hierarchical clustering algorithm.

The deep learning method is favored by more and more researchers, which provides a new direction for log anomaly detection. Zhang et al. [15] firstly employed LSTM for log system failure prediction. They gathered logs of similar format and content, and dealt with the "rarity" of labeled data in the training process by LSTM to capture the long-range dependency across log sequences. Due to the great similarity between log analysis and natural language processing, some works use natural language processing (NLP) related technologies for log anomaly detection. Du et al. [16] proposed DeepLog. DeepLog regards anomaly detection as a multi-classification problem of log keys. LSTM is used to model the log key sequence, and only the normal log key sequence is trained for LSTM. LSTM model needs to be trained with a large amount of normal data, but the original dataset cannot be all normal which may contain some abnormal log.

These works can not cope with the evolution of log statements, especially in the case of new log templates [17], which greatly limits their applicability in practice. Log instability is common, but there are few works on it. Brown et al. [18] proposed attention-based RNN, which improved the interpretability of the model without sacrificing the effect of anomaly detection. However, it focuses on log events and ignores the relationship of context in the log sequence. Zhang et al. [19] proposed LogRobust, which could handle the instability in log events and log sequences. LogRobust uses Bi-LSTM neural network based on attention to capturing context information. By assigning weights to different events, the model can automatically analyze the importance of events to improve the accuracy of anomaly detection. It can solve log instability problems, but the computational cost is large, and training is slower when the amount of log data is huge. Lin et al. [6] introduced the LogCluster algorithm based on the clustering method to realize data clustering and linear pattern mining of logs, but it still needed log parsing. Therefore, we propose LogUAD to resolve the problem of log instability and increase the accuracy and efficiency of anomaly detection with low computational cost.

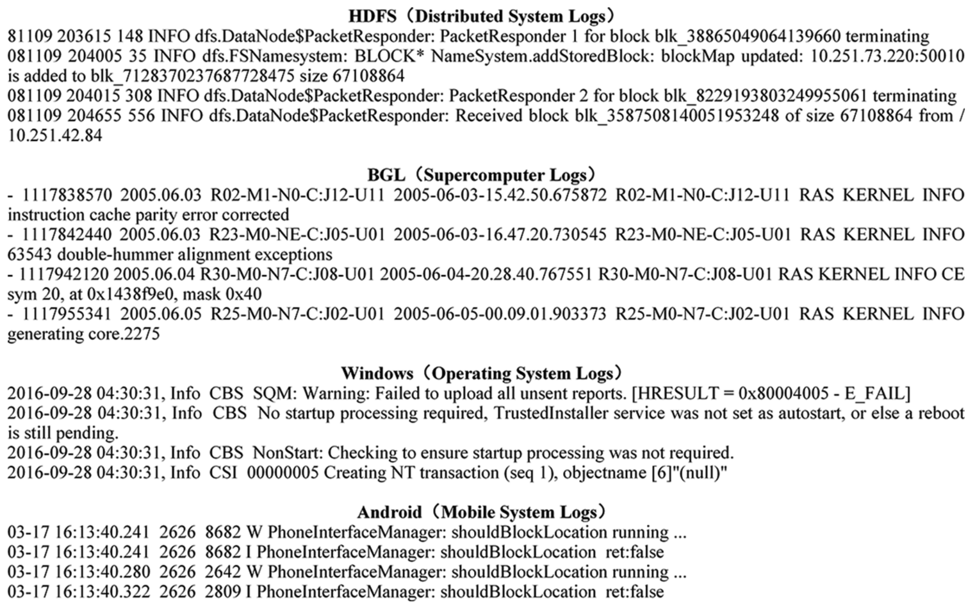
3 Preparatory Knowledge

In this session, we introduce the basic knowledge about different log types, general log anomaly detection framework, and Word2Vec.

3.1 Different Log Types

The system log records detailed information during system operation. Anomaly detection based on a system log can effectively maintain system security and reduce system failure.

Four different system logs are shown in Fig. 1. The log records the detailed information of the system operation, such as timestamp, message type, system operation status, dynamic changes of software and hardware, message level, date, and time of occurrence, etc. System logs are all semi-structured texts. The log specifications are not uniform, and different types of devices print different log formats.



HDFS（Distributed System Logs）

```
81109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_38865049064139660 terminating
081109 204005 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.73.220:50010
is added to blk_7128370237687728475 size 67108864
081109 204015 308 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_8229193803249955061 terminating
081109 204655 556 INFO dfs.DataNode$PacketResponder: Received block blk_3587508140051953248 of size 67108864 from /
10.251.42.84
```

BGL（Supercomputer Logs）

```
- 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.675872 R02-M1-N0-C:J12-U11 RAS KERNEL INFO
instruction cache parity error corrected
- 1117842440 2005.06.03 R23-M0-NE-C:J05-U01 2005-06-03-16.47.20.730545 R23-M0-NE-C:J05-U01 RAS KERNEL INFO
63543 double-hummer alignment exceptions
- 1117942120 2005.06.04 R30-M0-N7-C:J08-U01 2005-06-04-20.28.40.767551 R30-M0-N7-C:J08-U01 RAS KERNEL INFO CE
sym 20, at 0x1438f9e0, mask 0x40
- 1117955341 2005.06.05 R25-M0-N7-C:J02-U01 2005-06-05-00.09.01.903373 R25-M0-N7-C:J02-U01 RAS KERNEL INFO
generating core.2275
```

Windows（Operating System Logs）

```
2016-09-28 04:30:31, Info  CBS  SQM: Warning: Failed to upload all unsent reports. [HRESULT = 0x80004005 - E_FAIL]
2016-09-28 04:30:31, Info  CBS  No startup processing required, TrustedInstaller service was not set as autostart, or else a reboot
is still pending.
2016-09-28 04:30:31, Info  CBS  NonStart: Checking to ensure startup processing was not required.
2016-09-28 04:30:31, Info  CSI  00000005 Creating NT transaction (seq 1), objectname [6]"(null)"
```

Android（Mobile System Logs）

```
03-17 16:13:40.241  2626  8682 W PhoneInterfaceManager: shouldBlockLocation running ...
03-17 16:13:40.241  2626  8682 I PhoneInterfaceManager: shouldBlockLocation  ret:false
03-17 16:13:40.280  2626  2642 W PhoneInterfaceManager: shouldBlockLocation running ...
03-17 16:13:40.322  2626  2809 I PhoneInterfaceManager: shouldBlockLocation  ret:false
```

Figure 1: Log messages from four different systems

3.2 General Log Anomaly Detection Framework

Fig. 2 shows the overall framework of log anomaly detection, which contains the following four steps: log collection and processing, log parsing, feature extraction, and anomaly detection.

The system usually generates logs to record system status, operations, and changes. Therefore, logs are collected for preprocessing, such as log division and information extraction. Logs are unstructured, and the word count, and position in the text are not fixed. After preprocessing, this valuable information can be used for various purposes (for example, statistics, anomaly detection).

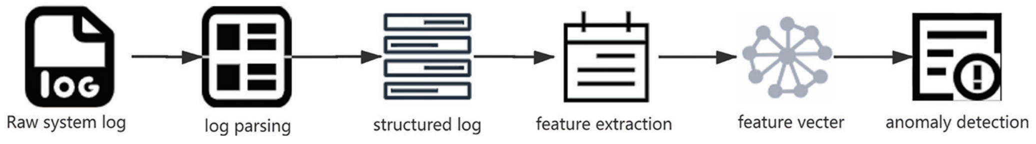


Figure 2: General log anomaly detection framework

The purpose of log parsing is to extract the log template from the original system log. Each log message can be seen as composed of a constant part and a variable part. Log parsing eliminates the variable part or replaces it with wildcards, and keeps only the constant part to form a log template.

After the logs are parsed into log templates, log templates are further encoded into digital feature vectors (such as log template count vectors, etc.). Then, a machine learning model can be used to detect the anomaly.

Finally, after obtaining the feature vector, the feature vector is input into the anomaly detection model. The trained model can identify whether the log is normal or abnormal.

3.3 Word2Vec

Word2Vec is a word vectorization tool proposed by Google in 2013, which calculates the relationship between words and explores the connections between words. According to the different types of input and output, it can be roughly divided into two completely different models, one is the continuous Skip-gram model, and the other is Continuous Bag of Words (CBOW). Word2Vec can convert words into word vectors, and vectorize texts while maintaining the semantic relationship between texts. Word2Vec can distinguish the semantic similarity of short texts, such as news headlines or similar bus and subway station analysis.

Fig. 3 summarizes the network structure of Word2Vec. The input and output are the One-Hot vectors of words. The first layer is a fully connected layer, and then there is no activation function. The output layer is a Softmax layer, which outputs a probability distribution that represents the probability of each word in the dictionary. Skip gram is a model that predicts context by giving input words. On the contrary, CBOW inputs the context word vector of the target word and outputs the word vector of the target word.

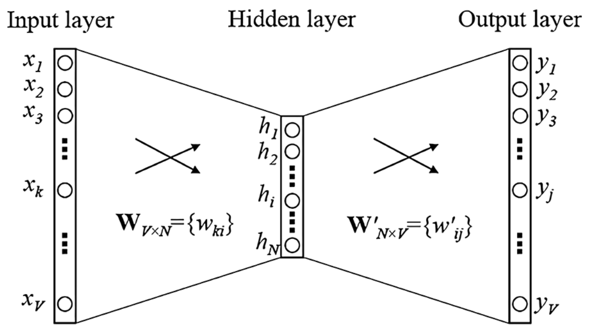


Figure 3: Network structure of Word2Vec model

4 Design of LogUAD

In this section, we first give the overview of LogUAD and the detailed steps of LogUAD, including log preprocessing, word vector matrix generation, word-sequence weight matrix generation, log sequence feature vector matrix generation, and unsupervised anomaly detection. Then, an example is given to illustrate our method.

4.1 The Overview of LogUAD

To unravel the issue of log instability, reduce computational overhead, and improve the accuracy and efficiency of log anomaly detection, we propose an unsupervised log anomaly detection method based on Word2Vec——LogUAD, as shown in Fig. 4. First of all, we download the relevant logs in the network system and divide and extract the collected system logs. Only the detailed contents are retained. There is no need to form structured text. By setting different window sizes, the detailed contents are divided into multiple log sequences. Then, the log contents are used as the input of Word2Vec to obtain the word vector. The feature vector of the log sequence can be obtained by combining TF-IDF of the log sequence. After obtaining the feature vector of the log sequence, log anomaly detection based on clustering can distinguish normal and abnormal logs. LogUAD mainly involves the following steps:

1) Data collection: We can download and collect a large number of logs from the network service system, such as distributed system logs, operating system logs, mobile system logs, etc. The collected logs include a great number of normal system logs and few anomaly system logs in the actual network environment.
2) Processing: After getting the original system log, delete the timestamp, node, information of the original system log, only extract the message content. Log messages are divided into multiple log sequences according to different window sizes (that is, the number of log entries).
3) Word vector matrix generation: Taking the words in the log sequence as the input of the Word2Vec, each word can be mapped to a vector space to generate a word vector. After training the vector of words, it is easy to be calculated in the next step. All words in the dictionary form a word vector matrix.
4) Word-sequence weight matrix generation: TF-IDF is a common weighting technique for information retrieval and data mining. The weight of a word increases proportionally with the number of occurrences in the log sequence, but if the frequency of occurrences in the file decreases, the weight of words also decreases. We use TF-IDF to calculate word-sequence weight matrix.
5) Log sequence feature vector generation: The word vector matrix is multiplied by the word-sequence weight matrix to get the log sequence feature vector.
6) Unsupervised clustering for anomaly detection: By the K-means clustering algorithm, the log sequence is divided into different clusters for anomaly detection according to the Euclidean distance from the feature vector of the log sequence to the cluster center.

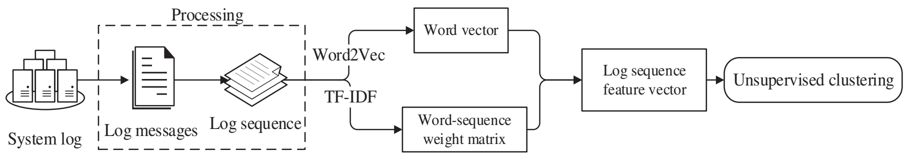


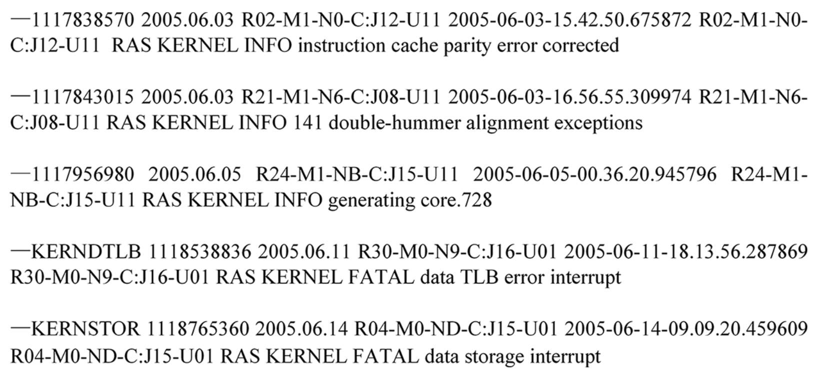
Figure 4: The overall framework of LogUAD

4.2 Log Processing

Most of the log files are original text and unstructured files. We take a dataset from the Blue Gene/L (BGL) [20] supercomputer system of Lawrence Livermore National Laboratory (LLNL) as an example. Fig. 5 lists some fragments of the BGL dataset.

As shown in Tab. 1, BGL data contains a lot of irrelevant information such as time, node, message type, kernel, etc. For example: "1118765360 2005.06.14 R04-M0-ND-C:J15-U01 RAS KERNEL FATAL data storage interrupt". The timestamp is 1118765360, the record date is 2005.06.14, the node is R04-M0-ND-C:J15-U01, the type of the log message is RAS, the location where the message is generated is

KERNEL, the corresponding level of the log message is FATAL, and the content is “data storage interrupt”. We remove irrelevant information and keep useful log messages, that is, “data storage interrupt”.



```
—1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.675872 R02-M1-N0-
C:J12-U11  RAS KERNEL INFO instruction cache parity error corrected

—1117843015 2005.06.03 R21-M1-N6-C:J08-U11 2005-06-03-16.56.55.309974 R21-M1-N6-
C:J08-U11 RAS KERNEL INFO 141 double-hummer alignment exceptions

—1117956980  2005.06.05  R24-M1-NB-C:J15-U11  2005-06-05-00.36.20.945796  R24-M1-
NB-C:J15-U11 RAS KERNEL INFO generating core.728

—KERNDTLB 1118538836 2005.06.11 R30-M0-N9-C:J16-U01 2005-06-11-18.13.56.287869
R30-M0-N9-C:J16-U01 RAS KERNEL FATAL data TLB error interrupt

—KERNSTOR 1118765360 2005.06.14 R04-M0-ND-C:J15-U01 2005-06-14-09.09.20.459609
R04-M0-ND-C:J15-U01 RAS KERNEL FATAL data storage interrupt
```

Figure 5: Examples of BGL original datasets

Table 1: Details of BGL log

Project	Data
Timestamp	1118765360
Date	2005.06.14
Node	R04-M0-ND-C:J15-U01
Information type	RAS
Generate location	KERNEL
Information level	FATAL
Content	Data storage interrupt

After extracting the log message contents, they are divided according to the fix window to generate log sequences. Assuming that the total number of log entries is N and the window size is t, the log contents are divided into m log sequences, where m = N/t. Each log sequence contains t consecutive logs. Seq_i represents the i-th log sequence.

4.3 Word Vector Matrix Generation

Because Word2Vec takes words as input, it is necessary to segment the log content first. Here, we can use space and other ways to segment words. For example, “141 double-hummer alignment exceptions data TLB error interrupt” are divided into nine words: “141”, “double”, “hummer”, “alignment”, “exceptions”, “data”, “TLB”, “error” and “interrupt”.

CBOW and Skip-gram are two training modes of Word2vec. We use CBOW to predict the current word through the context. CBOW calculates the probability of the current word through the first n or last n words of the current word.

As shown in Fig. 6, we suppose that a word is only associated with the first n and last n words of the word. The one-hot vector of the context word of the word $v_{(i-n)}, \ldots, v_{(i-1)}, v_{(i+1)} \ldots, v_{(i+n)}$ are entered, then the word vector of the $v_{(i)}$ through the Projection layer and Softmax layer is calculated. The word vector of word

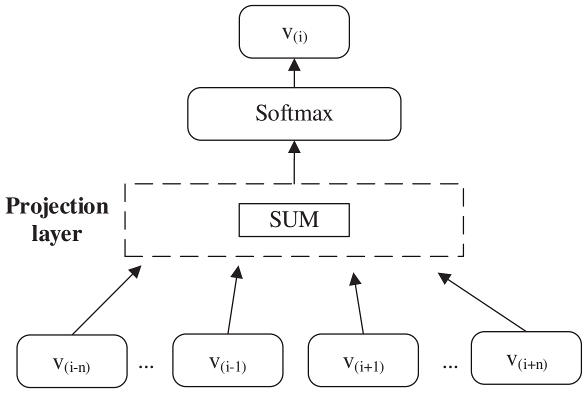
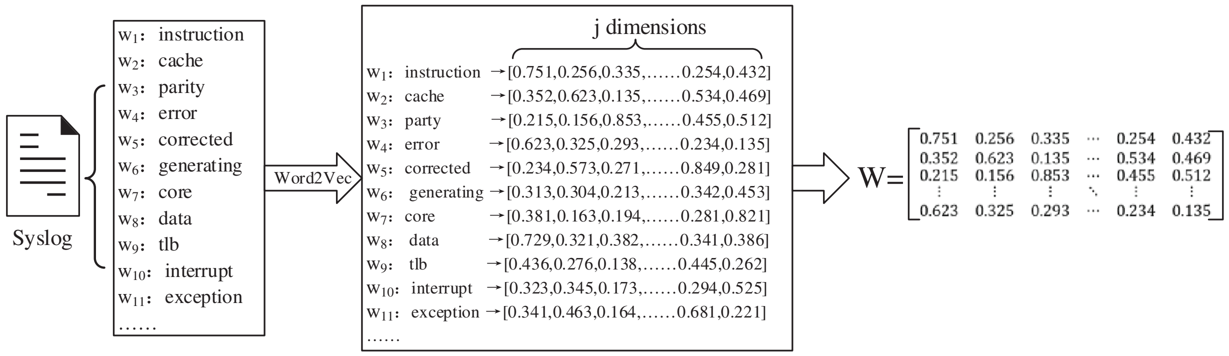


Figure 6: CBOW model structure

i is $w_i \in R^{1 \times j}$, where j is the dimension of the word. Assuming that the dictionary contains s words, the vectors of all words form a word vector matrix $\boldsymbol{W} \in R^{s \times j}$ according to the order of words appearing, as shown in Fig. 7.



Syslog
w1: instruction
w2: cache
w3: parity
w4: error
w5: corrected
w6: generating
w7: core
w8: data
w9: tlb
w10: interrupt
w11: exception
……
Word2Vec
j dimensions
w1: instruction →[0.751,0.256,0.335,……0.254,0.432]
w2: cache →[0.352,0.623,0.135,……0.534,0.469]
w3: party →[0.215,0.156,0.853,……0.455,0.512]
w4: error →[0.623,0.325,0.293,……0.234,0.135]
w5: corrected →[0.234,0.573,0.271,……0.849,0.281]
w6: generating →[0.313,0.304,0.213,……0.342,0.453]
w7: core →[0.381,0.163,0.194,……0.281,0.821]
w8: data →[0.729,0.321,0.382,……0.341,0.386]
w9: tlb →[0.436,0.276,0.138,……0.445,0.262]
w10: interrupt →[0.323,0.345,0.173,……0.294,0.525]
w11: exception →[0.341,0.463,0.164,……0.681,0.221]
……
W=

$$W=\begin{bmatrix} 0.751 & 0.256 & 0.335 & \cdots & 0.254 & 0.432 \\ 0.352 & 0.623 & 0.135 & \cdots & 0.534 & 0.469 \\ 0.215 & 0.156 & 0.853 & \cdots & 0.455 & 0.512 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0.623 & 0.325 & 0.293 & \cdots & 0.234 & 0.135 \end{bmatrix}$$

Figure 7: Generating schema of word vector matrix W

4.4 Word-Sequence Weight Matrix Generation

TF-IDF calculates the weight of words in all log sequences. If a word appears less frequently in all log sequences, it should be given higher weight. If it appears more frequently, its weight should be lower. If it does not appear in the current log sequence, its weight is 0. TF calculates the frequency of words, and IDF measures the occurrence of log events, which are formulated by Eq. (1) and Eq. (2).

$$\mathrm{TF}(\mathrm{word}_i) = \frac{Count(word_i)}{Total(word)} \tag{1}$$

$$\mathrm{IDF}(\mathrm{word}_i) = \log\frac{Total(seq_i)}{Total(seq) + 1} \tag{2}$$

where $Count(word_i)$ is the occurrence number of $word_i$ in all log sequences, $Total(word)$ is the total number of words in all log sequences, $Total(seq)$ is the total number of log sequences, and $Total(seq_i)$ is the total number of log sequences contain $word_i$. Finally, TF-IDF is the product of TF and IDF, according to Eq. (3):

$$\mathrm{TF} - IDF(word_i) = TF(word_i) \times IDF(word_i) \tag{3}$$

The weight vector of the i-th log sequence Seq_i is denoted by $T_{Seq_i} \in R^{1 \times s}$, where s is the total number of words. $T_{Seq_{iz}}$ represents the value of column z in the i-th log sequence. If $word_z$ belongs to log sequence Seq_i, the value of the z-th column of $T_{Seq_{iz}}$ is $TF - IDF(word_z)$. Otherwise, it is zero, as shown in Eq. (4):

$$\mathrm{T}_{Seq_{iz}} = \begin{cases} \mathrm{TF} - IDF(word_z), \ word_z \in Seq_i \\ \qquad\quad 0, \ word_z \notin Seq_i \end{cases} \tag{4}$$

There are m log sequences in total, and the weight vectors of all log sequences form a word-sequence weight matrix $T \in R^{m \times s}$.

4.5 Log Sequence Feature Matrix Generation

Obtaining the log sequence feature matrix is a key step in log anomaly detection. The log anomaly detection model takes the log sequence feature matrix as the input. The word-sequence weight matrix T is multiplied by the word vector matrix W to obtain the log sequence feature matrix vector $E_{seq} \in R^{m \times j}$, as shown in Eq. (5).

$$\mathrm{E}_{Seq} = T \times W \tag{5}$$

4.6 Unsupervised Anomaly Detection

An unsupervised clustering method based on K-means is exploited to detect anomaly. The K-means method has the advantages of simple principle, fast convergence speed, and good interpretability.

Algorithm 1: An Unsupervised Log Anomaly Detection Method Based on K-means

```
input: Log feature vector $D = \{E_{seq_1}, E_{seq_2}, E_{seq_3}, \ldots, E_{seq_m}\}$, the number of clusters k, and anomaly score threshold q
output: The anomaly detection results of all log sequence
1:k samples are randomly selected from D as the initial average vector $\{\mu_1, \mu_2, \mu_3, \ldots, \mu_k\}$
2:repeat
3:$C_i = \varnothing, \ (1 \le i \le k)$
4: for j = 1, 2, 3…m do
5:   Calculate the distance between $E_{seq_j}$ and $\mu_i (1 \le i \le k)$: $d_{ji} = \| E_{seq_j} - \mu_i \|_2$
6: Determine the cluster markup of $E_{seq_j}$ based on the nearest mean vector: $\lambda_j = \underset{i \in \{1,2,3,\ldots,k\}}{\arg\min} \ d_{ji}$
7:    Divide the sample $E_{seq_j}$ into the corresponding cluster: $C_{\lambda_j} = C_{\lambda_j} \cup \{E_{seq_j}\}$
8: end for
9: for i = 1, 2, 3…k do
10:   Computing the new mean vector: $\mu'_i = \frac{1}{|C_i|} \sum_{E_{seq_j} \in C_i} E_{seq_j}$
11: if $\mu'_i = \mu_i$ then
12:   Update current mean vector $\mu_i$ to $\mu'_i$
13: else
14:    Keep current mean unchanged
15:   end if
16:   end for
17:until cluster center vectors are not updated
18:for j = 1, 2, 3…m do
```

(Continued)

Algorithm 1: (Continued)

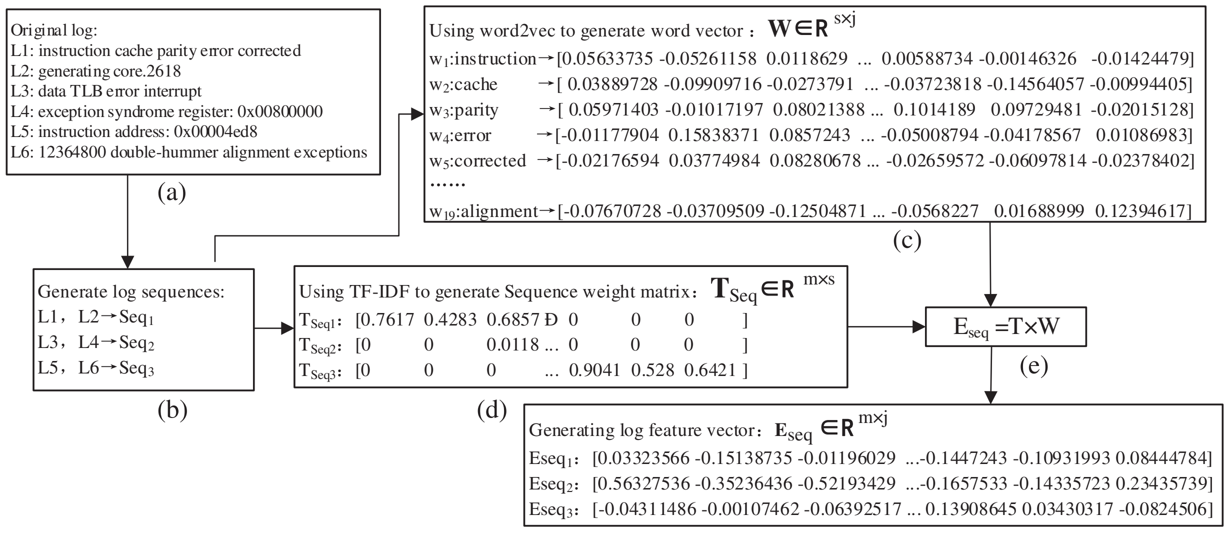
```
19:  if $d_{j\lambda_j} > q$ then
20:      Set the j log sequence to Anomaly
21: else
22:      Set the j log sequence to Normal
23:end if
24:end for
```

Algorithm 1 shows an unsupervised clustering method based on K-means. The input of this algorithm is the log sequence feature matrix, the number of cluster centers k, and the threshold of the abnormal fraction. The output of the algorithm is an anomaly label for each log sequence. The 1–3 row is initialization. The Euclidean distance between the log sequence vector and the center of each cluster is calculated in 4–8 rows and the log sequence is divided into the cluster with minimum distance. Line 9–17 updates the current cluster center until it is unchanged. Lines 18–24 mark the log that the distance between the log sequence vector and its cluster center is larger than the anomaly score threshold as an abnormal log, otherwise it is normal.

4.7 Example

An example is taken to briefly explain the main process of feature matrix generation. 6 lines of log messages are selected from the BGL dataset, which contains 19 (s = 19) unique words forming a dictionary.

As shown in Fig. 8a, the original system log is preprocessed to extract the log message contents. As shown in Fig. 8b, 6 log message is divided into 3 log sequences with a window size of 2. The 19 words in the dictionary are trained by Word2Vec to generate word feature vectors $w_{(i)} \in R^{1\times j}$, where j is the word vector dimension and is set in Word2Vec. Because there are 19 words in total, the word vector matrix W is $R^{19\times j}$, as shown in Fig. 8c.



```
Original log:
L1: instruction cache parity error corrected
L2: generating core.2618
L3: data TLB error interrupt
L4: exception syndrome register: 0x00800000
L5: instruction address: 0x00004ed8
L6: 12364800 double-hummer alignment exceptions
```

```
Generate log sequences:
L1，L2→Seq1
L3，L4→Seq2
L5，L6→Seq3
```

Using word2vec to generate word vector： $W \in R^{s\times j}$

```
w1:instruction→[0.05633735 -0.05261158  0.0118629  ...  0.00588734 -0.00146326  -0.01424479]
w2:cache      →[ 0.03889728 -0.09909716 -0.0273791  ... -0.03723818 -0.14564057 -0.00994405]
w3:parity     →[ 0.05971403 -0.01017197  0.08021388 ...  0.1014189   0.09729481 -0.02015128]
w4:error      →[-0.01177904  0.15838371  0.0857243  ... -0.05008794 -0.04178567  0.01086983]
w5:corrected  →[-0.02176594  0.03774984  0.08280678 ... -0.02659572 -0.06097814 -0.02378402]
......
w19:alignment→[-0.07670728 -0.03709509 -0.12504871 ... -0.0568227   0.01688999  0.12394617]
```

Using TF-IDF to generate Sequence weight matrix： $T_{Seq} \in R^{m\times s}$

```
TSeq1:  [0.7617  0.4283  0.6857 Đ  0        0       0      ]
TSeq2:  [0       0       0.0118 ...  0        0       0      ]
TSeq3:  [0       0       0      ...  0.9041   0.528   0.6421 ]
```

Generating log feature vector： $E_{seq} \in R^{m\times j}$

```
Eseq1:  [0.03323566 -0.15138735 -0.01196029  ...-0.1447243 -0.10931993 0.08444784]
Eseq2:  [0.56327536 -0.35236436 -0.52193429  ...-0.1657533 -0.14335723 0.23435739]
Eseq3:  [-0.04311486 -0.00107462 -0.06392517 ... 0.13908645 0.03430317 -0.0824506]
```

Figure 8: Example of generating feature vectors. (a) Original log. (b) Generating log sequences. (c) Generating word vector by Word2Vec. (d) Generating sequence weight matrix by TF-IDF. (e) Generating log feature vectors. (f) Details of log feature vectors

TF-IDF is used to calculate the weight of each word in all log sequences. As shown in Fig. 8d, the TF-IDF of the word is used to represent the word-sequence weight vector $T_{Seq_i} \in R^{1\times 19}$ of the i-th log sequence. All log sequence's weight vectors form a word-sequence weight matrix $T \in R^{3\times 19}$, where 3 is the number of log sequences.

Finally, as shown in Fig. 8e and 8f, the log sequence feature matrix $E_{seq} \in R^{3\times 19}$ is obtained by multiplying the word-sequence weight matrix T and the word vector matrix W. Then, the feature matrix is used unsupervised to detect the anomaly of the log.

5 Simulation and Evaluation

In this section, we first introduce the details of the data set used for the experiment, the experimental setup, and the experimental platform. Then we compare and evaluate our log anomaly detection method with LogCluster. Finally, we analyze the effects of different experimental parameters.

5.1 Datasets

We use the BGL dataset, which is published by the Blue Gene/L supercomputer system of LLNL. The BGL data contains 1,048,576 log messages. Tab. 2 shows the detailed information of the BGL dataset, including data size, number of log information, etc. BGL logs contains 6% Out-of-Vocabulary (OOV) words because of log evolution. After log parsing, the percent of OOV words can exceed 80% [21].

Table 2: Detailed information for BGL dataset

#System log	#Period	#Dataset size	#Number of messages	#Anomalies Log
BGL	24 days	1,048 M	1,048,576	224,921

5.2 Experimental Setup

We use 1,000,000 lines of log messages for the experiment from the BGL dataset. There are a total of 773 words (that is, s = 773). The word vector dimension of each word is 376 dimensions. The window size t of log sequences ranges from 5 to 100. Each line of log message has its label in the BGL dataset. Once there are one or more logs in the log sequence, this log sequence is abnormal. Otherwise, it is a normal log sequence. The default parameters of LogUAD are shown in Tab. 3.

Table 3: Default parameters of LogUAD

Dataset	N (Log lines)	j (Word vector dimension)	t (Window size)	k (Number of clusters)
BGL	1,000,000	376	20	2

We conduct an experimental comparison with LogCluster [6]. LogCluster processes and classifies the logs of large-scale online software systems, treats the log clustering problem as a pattern mining problem, and uses clustering methods to identify whether current failures have occurred before. LogCluster generates log templates by log parsing, then combines IDF-based and Contrast-based method to generate word vectors, and finally clusters them by similarity. Although the deep learning method has better performance, the computational overhead can be very large, when the data set is large. Moreover, the input log in the training stage must be correct so that a better model can be trained. However, most of the dataset contains anomaly logs. Therefore, we only compare with machine learning-based method.

We consider the impact of window sizes, the number of cluster and the dimension of word vectors on LogUAD. All experiments of LogUAD were run on AI Studio. AI Studio is an AI development platform based on Baidu deep learning platform, which equipped the server with a 4-core 32GB RAM and 100G memory CPU of Intel (R) Xeon (R) Gold 6148 and a Tesla V100 GPU with 32G memory.

5.3 Evaluation Metrics

Anomaly detection is usually a two-class classification problem. We use precision, recall, and F1-score, which are widely used in related studies, to evaluate the performance of our proposed method. The definition for precision, recall, and F1-score is shown in Eq. (6):

$$\begin{aligned} &\text{Precision} = \frac{TP}{TP+FP}, \ \text{Recall} = \frac{TP}{TP+FN} \\ &\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned} \tag{6}$$

A true positive (TP) is the number of the normal log sequences in practice, and the results predicted by the model are also normal. A false positive (FP) is the number of the abnormal log sequences in practice, and the results predicted by the model are normal. A false negative (FN) indicates the number of log sequences that the predictions are normal, but actually they are abnormal log sequences.

5.4 Experimental Result and Analysis

5.4.1 Comparison with Logcluster

In order to evaluate the performance of LogUAD, we compare it with the LogCluster proposed by Lin et al. [6]. The window size determines the division of the log sequences. To investigate the influence of window size, *t* is set to 5, 10, 20, and 32, and other parameters are set as default values.

The experimental results of LogUAD and LogCluster are shown in Fig. 9. As the window size increases, the F1-scores of LogUAD are 0.756, 0.757, 0.729, and 0.727, respectively, while the F1-scores of LogCluster are 0.457, 0.522, 0.625, and 0.545. Although the anomaly detection performance of LogCluster improves as the window size increases, the performance begins to decline when the window size increases to 20. LogCluster needs to parse the original log into a log template before clustering analysis, which introduces a lot of OOV words and results in a deviation in the log anomaly detection. As the window size gradually increases, LogUAD anomaly detection performance tends to decrease insignificantly and remains at a high level. Therefore, the overall performance of LogUAD is more accurate and more robust than LogCluster.

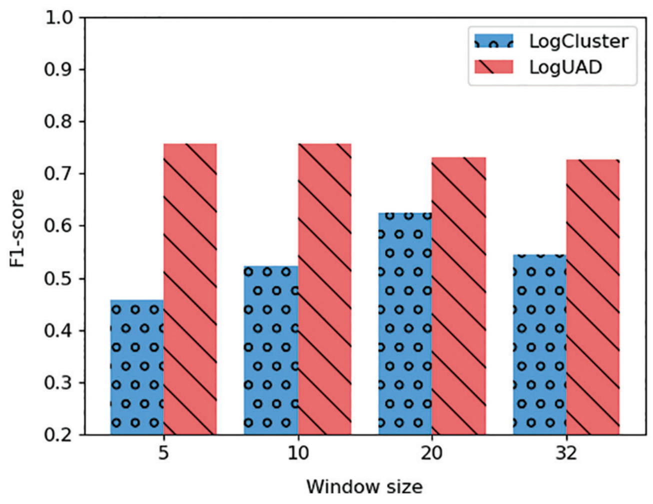


Figure 9: The F1-scores of LogUAD and LogCluster on BGL dataset

5.4.2 The Impact of Number of Clusters

The number of clusters plays an important role in LogUAD. Moreover, the number of clusters is set in advance according to experience. In order to observe the impact of the number of clusters on LogUAD, we set the number of cluster centers from 2 to 6. We use log sequences with four different window sizes t (5, 10, 20, and 32), and set other parameters, such as word vector dimensions, to the default value.

The F1-scores of LogUAD with the different number of clusters are shown in Fig. 10. The overall abnormality detection score tends to increase as the number of clusters decreases. For example, when the window size is 5 marked with the blue line, the F1-score of LogUAD gradually decreases from 0.756 to 0.637. When the number of clusters is less than 5, the red line with the window size of 32 maintains relatively stable anomaly detection performance. This shows that the LogUAD is stable and robust. When the number of clusters is 2 or 3, the F1-scores of LogUAD in different windows can reach 0.75, and can obtain better anomaly detection effects. Because the abnormal log vector is far from the cluster center, the fewer the number of cluster centers, the fewer the abnormal logs within the threshold range.

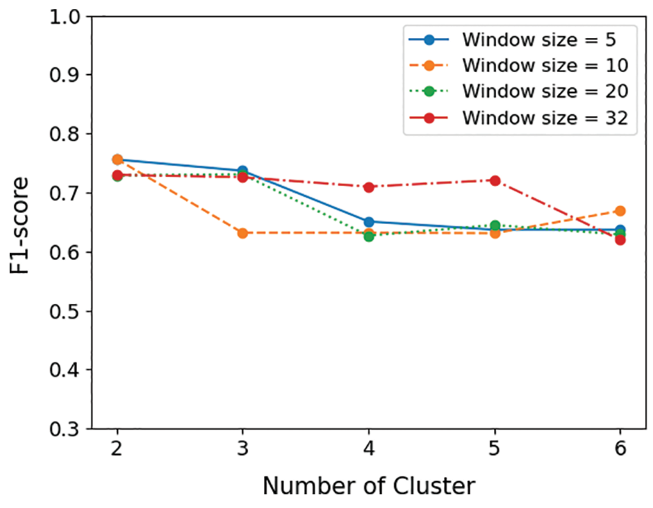


Figure 10: F1-score with different number of clusters

5.4.3 The Impact of Word Vector Dimensions

What word embedding does is map words into vector space, and the word vector dimension is a parameter that can be set. In order to evaluate the effect of word vector dimension on LogUAD, we select four types of word vector dimensions (200, 250, 300 and 376).

LogUAD with a high dimension of word vector can keep a stable anomaly detection effect when the window size changes. Fig. 11 shows the performance of 2 cluster centers in LogUAD and Fig. 12 demonstrate the performance of 2 cluster centers in LogUAD.

As shown in Fig. 11, when the word vector dimension is 200 and the window size is small, LogUAD can maintain a good anomaly detection effect. When the window size is larger than 32, the F1 scores begin to decline. The reason is that as the window size increases, a log sequence contains more and more log information. When the dimension of word vector is small, the cosine similarity of the word vectors of two log sequence is higher. It is difficult to distinguish and result in clustering errors.

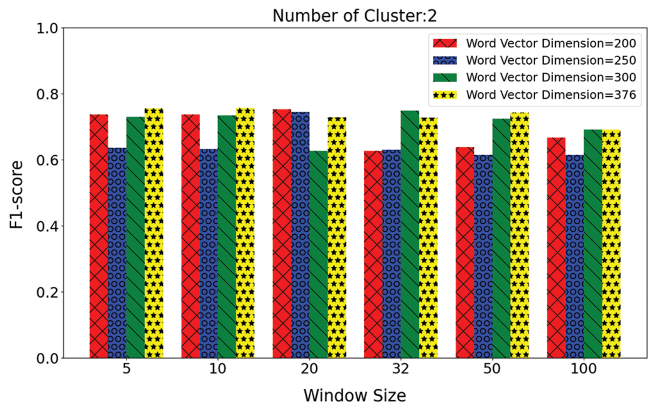


Figure 11: The performance of LogUAD with 2 cluster centers

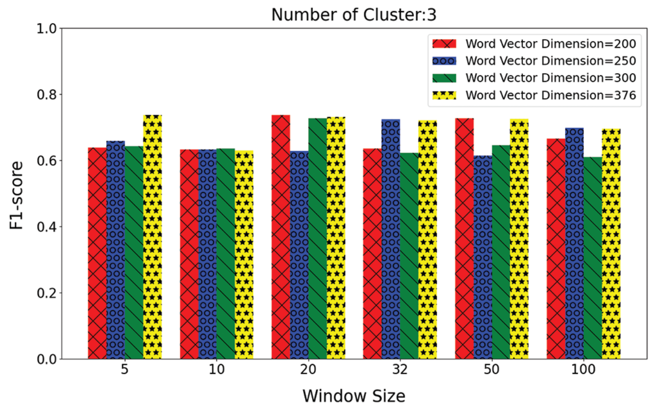


Figure 12: The performance of LogUAD with 3 cluster centers

As shown in Fig. 12, as the window size increases, LogUAD with high-dimensional word vector has a higher F1 score than that with low-dimensional word vector. LogUAD with high-dimensional word vector can maintain a stable anomaly detection effect. Therefore, LogUAD can effectively perform unsupervised anomaly detection.

From the above experiments, compared with LogCluster, LogUAD has better performance. The weight feature vector of LogUAD characterizes the log message efficiently. With different windows sizes and the different number of clusters, LogUAD can achieve robust performance. At the same time, LogUAD abandons the log parsing step and reduces computational overhead.

6 Conclusion

In this paper, we propose LogUAD, a log anomaly detection method based on Word2Vec. It is an off-line feature extraction method, which avoids the steps of log parsing for the original log messages, saves the time of the whole log anomaly detection. It can reduce the impact of the low accuracy of log parsing and also overcome the instability of system log. It uses K-means unsupervised clustering algorithm to detect anomaly. Simulation results show that our method is effective and superior to LogCluster. It is our next step to combine deep learning with LogUAD framework.

Acknowledgement: This work was funded by the Researchers Supporting Project No. (RSP. 2021/102) King Saud University, Riyadh, Saudi Arabia. The authors thank the National Natural Science Foundation of China, Natural Science Foundation of Hunan Province, and Research Foundation of Education Bureau of Hunan Province for their project financial support. In addition, the authors thank Baidu for the experimental platform and thank Tuo Deng and Meng Guo for the help of the experiment.

Funding Statement: This work was funded by the Researchers Supporting Project No. (RSP. 2021/102) King Saud University, Riyadh, Saudi Arabia. This work was supported in part by the National Natural Science Foundation of China under Grant 61802030, Natural Science Foundation of Hunan Province under Grant 2020JJ5602, the Research Foundation of Education Bureau of Hunan Province under Grant 19B005, and the International Cooperative Project for "Double First-Class", CSUST under Grant 2018IC24.

Conflicts of Interest: The authors declare there is no conflict of interest regarding the publication of this paper.

References

[1] B. Debnath, M. Solaimani, M. A. Gulzar, N. Arora, C. Lumezanu *et al.*, "Loglens: a real-time log analysis system," in *2018 IEEE 38th Int. Conf. on Distributed Computing Systems (ICDCS)*, Vienna, Austria, pp. 1052–1062, 2018.

[2] B. Zhu, J. Li, R. Gu and L. Wang, "An approach to cloud platform log anomaly detection based on natural language processing and LSTM," in *2020 3rd Int. Conf. on Algorithms, Computing and Artificial Intelligence*, New York, NY, USA, pp. 1–7, 2020.

[3] S. He, Z. Li, J. Wang and N. Xiong, "Intelligent detection for key performance indicators in industrial-based cyber-physical systems," *IEEE Trans. Ind. Inform*, vol. 17, no. 8, pp. 5799–5809, 2021.

[4] A. Farzad and T. A. Gulliver, "Unsupervised log message anomaly detection," *ICT Express*, vol. 6, no. 3, pp. 229–237, 2020.

[5] S. He, J. Zhu, P. He and M. R. Lyu, "Experience report: system log analysis for anomaly detection," in 2016 in *IEEE 27th Int. Symp. on Software Reliability Engineering (ISSRE)*, Ottawa, ON, Canada, pp. 207–218, 2016.

[6] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. of the 38th Int. Conf. on Software Engineering Companion*, Austin, Texas, USA, pp. 102–111, 2016.

[7] Y. Zhang and A. Sivasubramaniam, "Failure prediction in IBM blueGene/L event logs," in *2008 IEEE Int. Symp. on Parallel and Distributed Processing*, Miami, FL, USA, pp. 1–5, 2008.

[8] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan and E. Brewer, "Failure diagnosis using decision trees," in *Int. Conf. on Autonomic Computing, 2004. Proc.*, New York, NY, USA, pp. 36–43, 2004.

[9] M. Farshchi, J.-G. Schneider, I. Weber and J. Grundy, "Experience report: anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *2015 IEEE 26th Int. Symp. on Software Reliability Engineering (ISSRE)*, Gaithersbury, MD, USA, pp. 24–34, 2015.

[10] S. Zhang, Y. Zhang, Y. Chen, H. Dong, J. Xu *et al.*, "Prefix: switch failure prediction in datacenter networks," in *Abstracts of the 2018 ACM Int. Conf.*, Irvine, CA, USA, pp. 64–66, 2018.

[11] C. Bertero, M. Roy, C. Sauvanaud and G. Tredan, "Experience report: log mining using natural language processing and application to anomaly detection," in *2017 IEEE 28th Int. Symp. on Software Reliability Engineering (ISSRE)*, Toulouse, pp. 351–360, 2017.

[12] 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 Conf.*, Boston, MA, USA, pp. 1–14, 2010.

[13] W. Xu, L. Huang, A. Fox, D. Patterson and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. of the 27th Int. Conf. on Machine Learning*, Haifa, Israel, pp. 37–46, 2010.

[14] R. Yang, D. Qu, Y. Qian, Y. Dai and S. Zhu, "An online log template extraction method based on hierarchical clustering," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 135, 2019.

[15] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis *et al.*, "Automated IT system failure prediction: a deep learning approach," in *2016 IEEE Int. Conf. on Big Data (Big Data)*, Washington DC, USA, pp. 1291–1300, 2016.

[16] M. Du, F. Li, G. Zheng and V. Srikumar, "Deeplog: anomaly detection and diagnosis from system logs through deep learning," in *Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security*, Dallas, Texas, USA, pp. 1285–1298, 2017.

[17] J. Wang, Y. Tang, S. He, C. Zhao, P. K. Sharma *et al.*, "Logevent2vec: LogEvent-to-vector based anomaly detection for large-scale logs in internet of things," *Sensors*, vol. 20, no. 9, pp. 2451, 2020.

[18] A. Brown, A. Tuor, B. Hutchinson and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proc. of the First Workshop on Machine Learning for Computing Systems*, Tempe, AZ, USA, pp. 1–8, 2018.

[19] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proc. of the 2019 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*, Tallinn, Estonia, pp. 807–817, 2019.

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

[21] P. He, J. Zhu, S. He, J. Li and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2020.