Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.037392
Article

Unsupervised Log Anomaly Detection Method Based on Multi-Feature

Shiming He[1], Tuo Deng[1], Bowen Chen[1], R. Simon Sherratt[2] and Jin Wang[1,*]

[1]School of Computer & Communication Engineering, Changsha University of Science & Technology, Changsha, 410114, China
[2]School of Systems Engineering, The University of Reading, RG6 6AY, UK
*Corresponding Author: Jin Wang. Email: jinwang@csust.edu.cn
Received: 02 November 2022; Accepted: 08 February 2023; Published: 09 June 2023

Abstract: Log anomaly detection is an important paradigm for system troubleshooting. Existing log anomaly detection based on Long Short-Term Memory (LSTM) networks is time-consuming to handle long sequences. Transformer model is introduced to promote efficiency. However, most existing Transformer-based log anomaly detection methods convert unstructured log messages into structured templates by log parsing, which introduces parsing errors. They only extract simple semantic feature, which ignores other features, and are generally supervised, relying on the amount of labeled data. To overcome the limitations of existing methods, this paper proposes a novel unsupervised log anomaly detection method based on multi-feature (UMFLog). UMFLog includes two sub-models to consider two kinds of features: semantic feature and statistical feature, respectively. UMFLog applies the log original content with detailed parameters instead of templates or template IDs to avoid log parsing errors. In the first sub-model, UMFLog uses Bidirectional Encoder Representations from Transformers (BERT) instead of random initialization to extract effective semantic feature, and an unsupervised hypersphere-based Transformer model to learn compact log sequence representations and obtain anomaly candidates. In the second sub-model, UMFLog exploits a statistical feature-based Variational Autoencoder (VAE) about word occurrence times to identify the final anomaly from anomaly candidates. Extensive experiments and evaluations are conducted on three real public log datasets. The results show that UMFLog significantly improves F1-scores compared to the state-of-the-art (SOTA) methods because of the multi-feature.

Keywords: System log; anomaly detection; semantic features; statistical features; Transformer

1 Introduction

Modern large-scale systems usually serve millions of users, and as the number of users grows, the scale of the system becomes larger and larger, which is vulnerable to potential threats [1–3]. Any system failure caused by unstable factors leads to a large number of economic losses. Therefore, it is critical to maintain the stable and healthy operation of the system. Anomaly detection can identify failures and alert the operator in time to mitigate the loss of the system. Most anomaly detection methods focus on Key Performance Indicators (KPI) [4,5] to find outliers that deviate from normal indicator data. However, since KPI only contain numerical data, it is hard to know the specific reason for the anomaly. Instead, log messages record detailed information about the operation of a system, including operating status, and generate events, which are important data sources for anomaly detection. For example, "1118147576 2005.06.07 R17-M0-N1-C:J03-U01 2005-06-07-05.32.56.626092 RAS KERNEL FATAL machine check interrupt (bit=0x1d): L2 dcache unit data parity error" is a log message of the supercomputer, which shows a data parity error occurred on 07/06/2005 at 05:32:56 while machine checking. Log anomaly detection aims to automatically detect all abnormal logs from log data so that operators can quickly troubleshoot problems and ensure the reliability of the system, which has attracted extensive research [6–9].

With the popularity and development of Deep Learning (DL) [10,11] and Natural Language Processing (NLP) [12–15], Recurrent Neural Networks (RNNs) models, such as Long Short-Term Memory (LSTM) [16] networks, are widely applied to log sequence anomaly detection due to their memory property for sequence data [17–21]. DeepLog [17] and LogAnomaly [18] use LSTM to construct unsupervised models to detect anomalies that deviate from normal sequential patterns, whereas LogRobust [19] and SwissLog [20] build supervised models to detect log anomalies by combining attention mechanisms into Bidirectional Long Short-Term Memory (Bi-LSTM) networks. However, both LSTM and Bi-LSTM need to process sequence data in left-to-right or right-to-left order, which is time-consuming and unsuitable for long sequence data. Typically, the sequence of logs generated by the system can be relatively long (e.g., in a cloud computing environment such as OpenStack, the logs are generated by a particular VM instance during its lifecycle from startup to destruction). Therefore, RNNs-based log anomaly detection has low efficiency.

Recently, the Transformer [22] architecture instead of LSTM stands out in log anomaly detection, which greatly improves the effect of log anomaly detection [23–30]. Transformer can not only process the sequence in parallel to solve the long-range dependency problem but also has lower computational complexity and cost. However, existing works based on Transformer still face the following challenges:

Challenge 1: Most existing methods rely on log parsing to preprocess semi-structured log data. However, log parsing may introduce parsing errors and information loss because of log evolution, which affects the performance of anomaly detection [30].

Challenge 2: Existing methods usually exploit One-Hot encoding, Word2vec word embedding, or random vector initialization to represent the log messages. These methods have poor representation ability to extract the semantics of log messages, which also impacts the effectiveness of anomaly detection.

Challenge 3: System logs record the important operations and states of a running system, which provide rich information (log contents, timestamps, parameters, components, and other information) for diagnosing and maintaining the system. It is still a tricky problem to fully utilize the rich information in logs for anomaly analysis.

Challenge 4: In practice, the number of normal logs is much larger than that of abnormal logs, and the label cost of abnormal logs is huge. The inability of supervision-based anomaly detection methods to address imbalanced data issues and detect unknown anomalies is out of step with current industrial practice.

To overcome the above challenges, this paper proposes an unsupervised log anomaly detection method based on multi-feature named UMFLog. UMFLog includes two sub-models to consider two kinds of features: semantic feature and statistical feature, respectively. The first model uses a hypersphere-based Transformer model to learn log sequence representations and obtain anomaly candidates. Then, the second model uses a statistical feature-based Variational Autoencoder (VAE) model about word occurrence times in the log sequence to identify the final anomaly from anomaly candidates. The main contributions of this paper are summarized as follows:

- To avoid the log parsing error problems in Challenge 1, this paper directly processes the original content of the log message without log parsing, which preserves the parameters of the log content and the context order in the log sequence.
- To address the lack of log semantic problems in Challenge 2, this paper constructs a Transformer anomaly detection model based on hypersphere. The model uses the large pre-trained language model Bidirectional Encoder Representations from Transformers (BERT) [31] to extract the semantic information of logs and constructs a compact log sequence representation by the hypersphere loss to filter out anomaly candidates.
- To solve the lack of log multi-feature representation problem in Challenge 3, this paper constructs a statistical feature-based VAE model of word occurrence times. The model counts the occurrences of log sequence tokens to obtain statistical features so as to make full use of log information, which identifies the final anomaly from anomaly candidates and effectively reduces false positives.
- UMFLog is an unsupervised model which does not need labeled data for training and can cope with the label cost problem in challenge 4. Extensive experiments and evaluations are conducted on three real public log datasets, Blue Gene/L(BGL) [32], Thunderbird [32], and Hadoop Distributed File System (HDFS) [33]. The results show that UMFLog significantly improves evaluation scores compared to the state-of-the-art (SOTA) methods.

The remainder of this paper is organized as follows: Section 2 presents related work. Section 3 introduces preliminaries. Section 4 provides an overview and detailed steps of UMFLog. Section 5 conducts experimental evaluations and discusses the results. Finally, Section 6 summarizes the paper and looks forward to future work.

2 Related Works

This subsection first summarizes the related work on log anomaly detection methods based on RNNs and Transformers. Then, it summarizes the SOTA methods with hyperspheres and compares them with UMFLog.

2.1 RNNs-based Methods

DeepLog [17] treats log messages as sequential elements, that follow specific patterns and syntax rules and build an unsupervised log anomaly detection model by LSTM for the first time. DeepLog ignores the semantic information of log messages and represents log messages with log template IDs. Therefore, LogAnomaly [18] proposes Template2Vec to construct a collection of synonym-antonyms to extract the semantic information hidden in log templates and construct an unsupervised model to detect log sequential anomalies and quantity anomalies by LSTM. LogRobust [19] uses a text representation model called Fasttext to learn the semantic vector of a log template, which solves the log robustness problem, and constructs a supervised model by Bi-LSTM with attention for log anomaly detection. SwissLog [20] takes BERT as a sentence encoder to extract log template semantic embeddings by random initializing the words in the log template, and concatenates the semantic embeddings with the time interval embeddings. Based on the joint embedding, a supervised model is constructed by BiLSTM to detect anomalies of log sequence order changes and log interval changes. LogTAD [21] solves the problem of insufficient target domain data by transferring the source domain knowledge. Specifically, it uses the adversarial domain network to mix the source domain and target domain data and trains the LSTM model with the hypersphere loss.

The above RNNs-based method is time-consuming and not suitable for long sequence data.

2.2 *Transformer-based Methods*

Unsupervised methods based on Transform are proposed. Wibisono et al. [23] use Transformer to replace LSTM for the first time to solve the problem of long-sequence log anomaly detection. Chen et al. [24] reproduce the experiments based on Ref. [23] and carry out comparative experiments to demonstrate the superiority of Transformer in log anomaly detection. Logsy [25] initializes log template token IDs(refers to TokenID, and the token is what constitutes the content of the log message) with random vectors to extract the semantic embeddings by Transformer encoder with help of auxiliary dataset. Finally, it trains the model with a hypersphere loss to obtain compact vector representations of log events for individual log anomaly detection. LogBERT [26] initializes the log template IDs (refers to TemplateID) with random vectors and obtains the log sequence vectors by the Transformer encoder. Finally, it combines two self-supervised training tasks, masked event prediction and hypersphere volume minimization, to learn patterns of normal log sequences and detect anomalies. Trine [27] uses three Transformer encoders to construct unsupervised models for anomaly detection based on Generative Adversarial Networks (GAN). One of the Transformer encoders extracts feature representations of the system logs, while the other two Transformer encoders serve as generators and discriminators of the GAN, respectively, to alleviate the class imbalance of the data and detect log anomalies. CAT [28] proposes a self-attention-based encoder-decoder Transformer framework to predict the mask subsequences, combined with the hypersphere task for log sequence anomaly detection.

Some supervised methods based on Transformer are proposed. HitAnomaly [29] leverages Transformer encoders to build a hierarchy to extract latent representations of log sequences. NeuralLog [30] does not perform log parsing, directly uses BERT to extract the semantic features of the original log messages, and exploits the Transformer encoder to build a supervised model to detect log anomalies.

The above Transformer-based methods solve the long sequence dependency problem in LSTM, but still face the challenges mentioned in Section 1.

2.3 *The Differences between SOTA Methods Based on Hyperspheres and UMFLog*

This paper summarizes the aforementioned methods based on hyperspheres and contrasts them with UMFLog to clearly show the differences, as shown in Table 1. In terms of detection object, Logsy handles single log message, while the detection objects of other methods are log sequences. In terms of the input form, LogBERT uses the parsed log template ID (TemplateID) as input, while LogTAD uses the Template token, which contains more information, but they cannot avoid the log parsing errors problem. CAT and UMFLog use the raw log content as input. In terms of feature extraction model and feature embedding, Logsy and LogBERT do not use separate models and use random vectorization to represent feature embeddings, while other methods use a separate feature extraction model. But Word2vec, used by LogTAD, cannot extract context semantic features, and the LSTM model cannot handle the problem of long-sequence log anomaly detection. CAT is more similar to UMFLog in input form and embedding method, but CAT focuses on predicting the masked subsequences by the Transformer encoder and decoder. Finally, none of these methods consider multi-features and do not take full advantage of the rich information in logs.

Table 1: Comparison of methods based on hyperspheres

Method	Detection object	Input form	Feature extraction model	Feature embedding	Detection model	Loss function
Logsy	Single log message	Token ID	/	Random initialization	Transformer encoder	Hypersphere loss
LogBERT	Log sequence	Template ID	/	Random initialization	Transformer encoder	Masked event prediction loss + hypersphere loss
LogTAD	Log sequence	Template	Word2vec	Template embedding	LSTM	Domain adversarial loss + hypersphere loss
CAT	Log sequence	Content	Fine-tuning BERT	Content embedding	Transformer encoder Transformer decoder	Masked subsequence prediction loss + hypersphere loss
UMFLog	Log sequence	Content, Statistical data	BERT-Service	Content embedding	Transformer encoder +VAE	Hypersphere loss + reconstruct loss

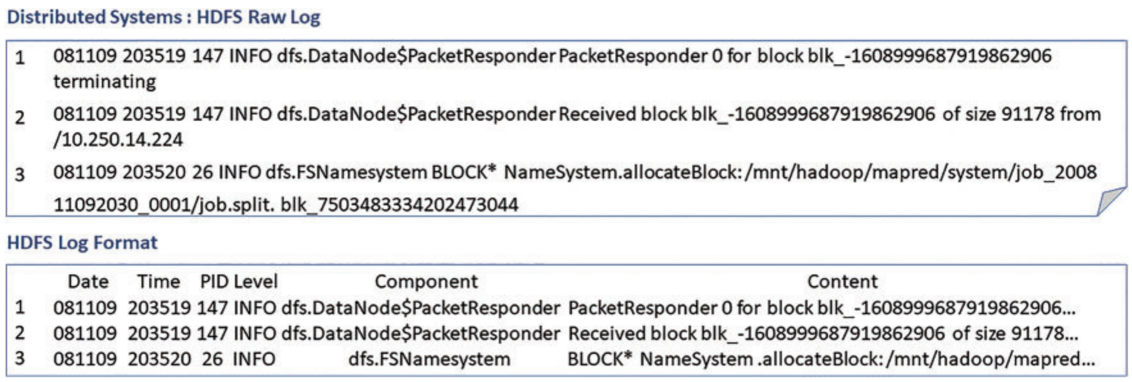
3 Preliminaries

This section introduces some basic concepts about log anomaly detection.

3.1 Log Data

Even when the system malfunctions, the system log still records its operational status. As a result, logs can help operators identify anomalies and locate the root cause of a problem, increasing the effectiveness of operation and maintenance.

As shown in Fig. 1, this paper gives an example of HDFS logs. There are three raw log messages. The raw system log records rich information and has special formats that include data, timestamps, Process Identification (PID), components, log contents, etc. The log content is semi-structured text consisting of constant and variable tokens. The constant tokens are the constant keywords. The variable tokens are the parameters with dynamic runtime information, which are classified into character parameters and non-character parameters according to the token category. For example, Table 2 shows the content, constant token, variable token, character parameters, and non-character parameters of the third HDFS log in Fig. 1.


Distributed Systems : HDFS Raw Log

```
1  081109 203519 147 INFO dfs.DataNode$PacketResponder PacketResponder 0 for block blk_-1608999687919862906
   terminating
2  081109 203519 147 INFO dfs.DataNode$PacketResponder Received block blk_-1608999687919862906 of size 91178 from
   /10.250.14.224
3  081109 203520 26 INFO dfs.FSNamesystem BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_2008
   11092030_0001/job.split. blk_7503483334202473044
```

HDFS Log Format

```
     Date    Time   PID Level          Component                             Content
1  081109  203519 147 INFO dfs.DataNode$PacketResponder  PacketResponder 0 for block blk_-1608999687919862906...
2  081109  203519 147 INFO dfs.DataNode$PacketResponder  Received block blk_-1608999687919862906 of size 91178...
3  081109  203520  26 INFO          dfs.FSNamesystem        BLOCK* NameSystem .allocateBlock: /mnt/hadoop/mapred...
```

Figure 1: HDFS raw log and log format

Table 2: HDFS raw log content and parsing results

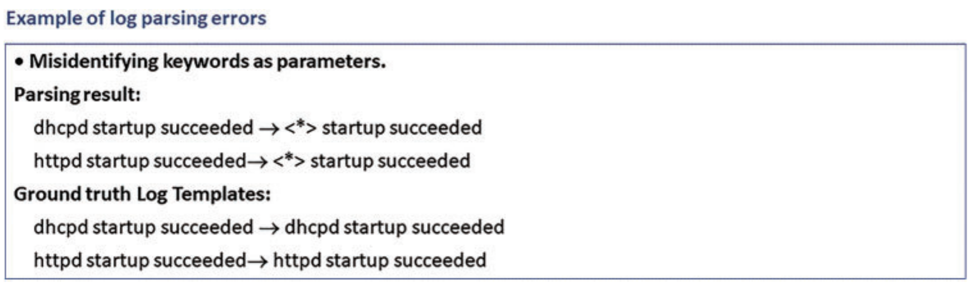
Content	BLOCK∗ NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_200811092030_0001/job.split. blk_7503483334202473044
Constant token	BLOCK∗ NameSystem.allocateBlock:
Variable token	/mnt/hadoop/mapred/system/job_200811092030_0001/job.split. blk_7503483334202473044
Character arguments	mnt, hadoop, mapred, system, job, split, blk
Non-character parameter	200811092030_0001,7503483334202473044
Template	BLOCK∗ NameSystem.allocateBlock: <*> <*>

3.2 *Log Parsing*

Since the log content is semi-structured, it is not conducive to automatic processing. Therefore, log parsing is introduced to get the structured log content. Generally, log parsing keeps the constant token and deletes the variable tokens, termed a log template. As shown in Table 2, the log template of the third log message is "BLOCK∗ NameSystem.allocateBlock: <*> <*>". It is obvious that the character parameters with semantics are also discarded, resulting in the loss of semantic information.

In addition, log parsing may cause parsing errors due to the Out-of-Vocabulary (OOV) of log evolution or misinterpretation of keywords and parameters. As shown in Fig. 2, the log parsing takes the keywords "dhcpd" and "httpd" as parameters and produces an incorrect log template "<∗> startup succeeded".

Some recent works detect log anomalies without log parsing because the information loss and parsing errors caused by log parsing degrade the anomaly detection performance [9,30,34]. This paper also does not perform log parsing to avoid this problem.


Example of log parsing errors

- **Misidentifying keywords as parameters.**

Parsing result:

dhcpd startup succeeded → <*> startup succeeded

httpd startup succeeded→ <*> startup succeeded

Ground truth Log Templates:

dhcpd startup succeeded → dhcpd startup succeeded

httpd startup succeeded→ httpd startup succeeded

Figure 2: Example of log parsing error

4 Unsupervised Log Anomaly Detection Method Based on Multi-feature

This section first gives problem definitions of log anomaly detection, and Table 3 summarizes the notations that appear in this paper. Then it introduces the overview and steps of the proposed method in detail.

Table 3: Notations

Symbol	Description
tok	token. A token is what makes up the content of a log message.
ℓ	$\ell = \left\{tok_1, tok_2, \ldots, tok_{\lvert\ell\rvert}\right\}$ A log content ℓ is a sequence of tokens.
$\mathcal{S}$	$\mathcal{S} = \left\{\ell_1, \ell_2, \ldots, \ell_{\lvert\mathcal{S}\rvert}\right\}$ A log sequence $\mathcal{S}$ is a sequence of chronologically ordered log content.
$\mathfrak{s}$	$\left\{tok_1^{\ell_1}, \ldots, tok_{\lvert\ell_1\rvert}^{\ell_1}, \ldots, tok_1^{\ell_{\lvert\mathcal{S}\rvert}}, \ldots, tok_{\lvert\ell_{\lvert\mathcal{S}\rvert}\rvert}^{\ell_{\lvert\mathcal{S}\rvert}}\right\}$ A log token sequence $\mathfrak{s}$ is the token form of the log sequence.
$\mathcal{T}$	$\mathcal{T} = \left\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{\lvert\mathcal{T}\rvert}\right\}$ Set of log sequences, including training set $\mathcal{T}_1$, validation set $\mathcal{T}_2$, and test set $\mathcal{T}_3$
$\mathfrak{e}sem$	The initial embedding vector of the log token generated by the embedding layer of BERT.
$\mathfrak{w}sem$	Log token semantic vector
ℓsem	Log content semantic vector
$\mathcal{P}$	$\mathcal{P} = \{\mathfrak{p}os_1, \mathfrak{p}os_2, \ldots, \mathfrak{p}os_{\lvert\mathcal{S}\rvert}\}$ Sequence of positional embeddings of all log messages in a window.
$\mathfrak{X}sem$	$\mathfrak{X}sem = \left\{x_1, \ldots, x_t, \ldots, x_{\lvert\mathcal{S}\rvert}\right\}$ Sequence of log vectors, where $x_t = \ell sem_t + \mathfrak{p}os_t$

(Continued)

Table 3: Continued

Symbol	Description
$\mathcal{Z}sem$	$\mathcal{Z}sem = \left\{z_1, \ldots, z_i, \ldots, z_{\lvert\mathcal{S}\rvert}\right\}$ A sequence of latent representations of log messages output by Transformer
$R_{\mathcal{S}}$	The compact log sequence vector representation
R_c	The hypersphere center vector representation
Sta^{tok}	The normalized occurrences times for tokens after the maximum and minimum normalization
$D_{\mathcal{T}}$,$D_{\mathcal{V}}$	$D_{\mathcal{T}} = \left\{tok_1\colon Sta^{tok_1}, \ldots, tok_i\colon Sta^{tok_i}, \ldots, tok_M\colon Sta^{tok_{M_{train}}}\right\}, D_{\mathcal{V}} = \left\{tok_1\colon Sta^{tok_1}, \ldots, tok_i\colon Sta^{tok_i}, \ldots, tok_M\colon Sta^{tok_{M_{val}}}\right\}$The standard statistical value Dictionary of all normal sequences in $\mathcal{T}_1$ and $\mathcal{T}_2$, M_{train} and M_{val} is the total number of all tokens in $D_{\mathcal{T}}$ and $D_{\mathcal{V}}$, respectively.
$\mathcal{X}sta$	Log sequence statistical feature vector
G	Candidate anomaly log sequence set
$L_{OO\mathcal{V}}$	$L_{OO\mathcal{V}} = \{oov_1, \ldots, oov_i, \ldots\}$ The list of OOV token

4.1 Problem Definition

Definition 4.1 (Log content). Log content is the detailed content of a log message which consists of a sequence of tokens. Formally, a log content is denoted by $\mathfrak{l} = \left\{tok_1, tok_2, \ldots, tok_{\lvert\mathfrak{l}\rvert}\right\}$, where tok$_i$ represents the i-th tokens, $\lvert\mathfrak{l}\rvert$ is the total number of tokens.

Definition 4.2 (Log sequence). A log sequence can be described as a sequence of consecutive logs in chronological order within an observation window. Formally, a log sequence is denoted by $\mathcal{S} = \left\{\mathfrak{l}_1, \mathfrak{l}_2, \ldots, \mathfrak{l}_{\lvert\mathcal{S}\rvert}\right\}$, where $\mathfrak{l}_i$ represents the i-th log content, $\lvert\mathcal{S}\rvert$ is the total number of log content within a window.

Definition 4.3 (Log token sequence). A log token sequence is the token form of the log sequence. Formally, a log token sequence is denoted by $\mathfrak{s} = \left\{tok_1^{\mathfrak{l}_1}, \ldots, tok_{\lvert\mathfrak{l}_1\rvert}^{\mathfrak{l}_1}, \ldots, tok_1^{\mathfrak{l}_{\lvert\mathcal{S}\rvert}}, \ldots, tok_{\left\lvert\mathfrak{l}_{\lvert\mathcal{S}\rvert}\right\rvert}^{\mathfrak{l}_{\lvert\mathcal{S}\rvert}}\right\}$, where $\lvert\mathfrak{l}_i\rvert$ represents the total number of tokens for the i-th log content, $\lvert\mathfrak{s}\rvert$ is the total number of tokens of the log token sequence, $\lvert\mathfrak{s}\rvert = \lvert\mathfrak{l}_1\rvert + \lvert\mathfrak{l}_2\rvert + \ldots + \left\lvert\mathfrak{l}_{\lvert\mathcal{S}\rvert}\right\rvert$.

Definition 4.4 (Log sequence anomaly detection). Given three disjoint set of log sequences as training set $\mathcal{T}_1 = \left\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{\lvert\mathcal{T}_1\rvert}\right\}$, validation set $\mathcal{T}_2 = \left\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{\lvert\mathcal{T}_2\rvert}\right\}$, and test set $\mathcal{T}_3 = \left\{\mathcal{S}, \mathcal{S}_2, \ldots, \mathcal{S}_{\lvert\mathcal{T}_3\rvert}\right\}$. Using the training set to train an anomaly detection model, the trained model can indicate whether the sequence $\mathcal{S}$ in the test set is normal or anomalous. The training set $\mathcal{T}_1$ of an unsupervised model only contains normal sequences.

4.2 Overview

To overcome the limitations of existing methods, this paper designs UMFLog, a novel unsupervised multi-feature-based log anomaly detection method, whose framework is shown in Fig. 3. UMFLog is a two-stage model, including an offline training stage and an online detection stage, and each stage is composed of two sub-models, namely a hypersphere-based Transformer model and a statistical feature-based VAE model.

The process of the offline training stage is as follows:

1. **Preprocessing.** In order to tackle log parsing errors or information loss problems in Challenge 1, this paper preprocesses the historical raw log message data without log parsing to delete non-character parameters and obtain the log content $\mathfrak{l}$;
2. **Log sequence partition.** The second step is to use a sliding window or a session window to divide the log contents into the log sequence $\mathcal{S}$ and the log token sequence $\mathfrak{s}$;
3. **Transformer model based on hypersphere.** To fully mine semantic features in Challenge 2, this paper uses Bert to extract semantic features to represent log sequence and builds a Transformer model based on hypersphere;
4. **VAE model based on statistical features.** In order to make full use of multi-feature to improve anomaly detection accuracy and reduce false positives in Challenge 3, this paper builds a VAE model based on statistical features about word occurrence times in the log sequence for statistical feature extraction.

In the online detection stage, the new logs perform the same process as the offline training stage in data preprocessing and log sequence partition. After that, this paper uses the trained Transformer model with hypersphere to filter out candidate anomaly log sequences. The trained VAE model further detects candidate anomaly log sequences to determine the final abnormal log sequence.

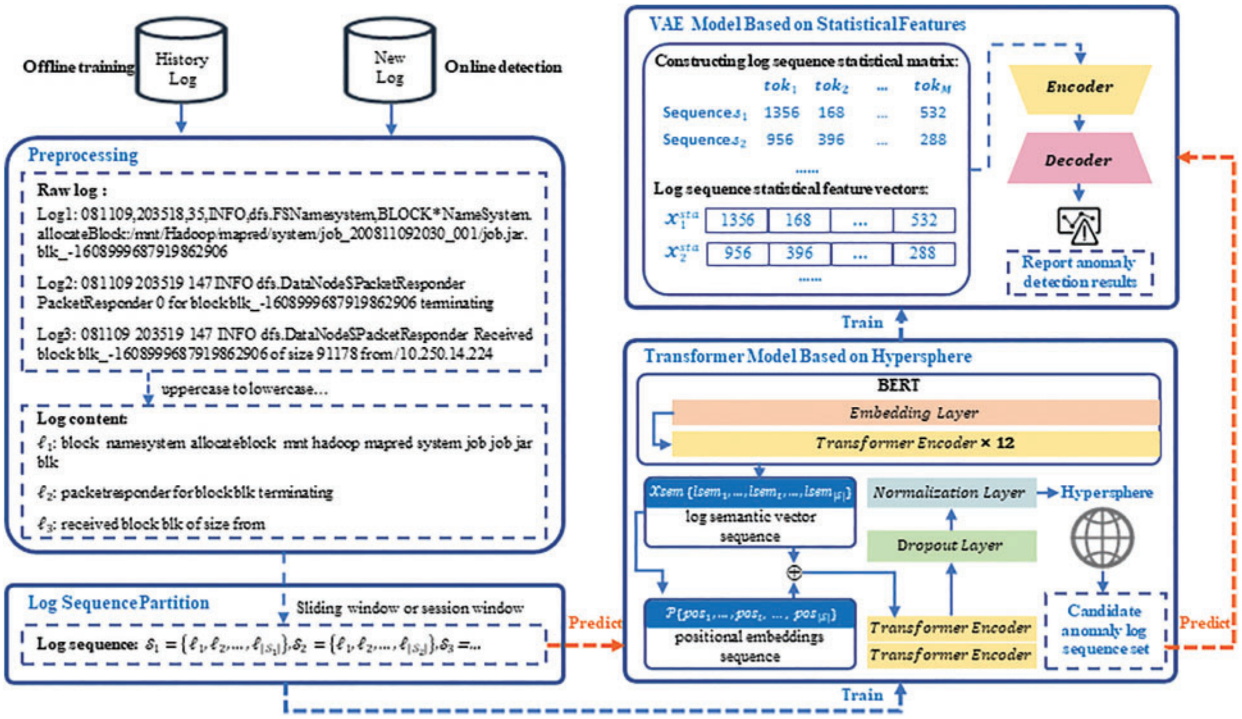

```
Raw log :
Log1: 081109,203518,35,INFO,dfs.FSNamesystem,BLOCK*NameSystem.
allocateBlock:/mnt/Hadoop/mapred/system/job_200811092030_001/job.jar.
blk_-1608999687919862906
Log2: 081109 203519 147 INFO dfs.DataNode$PacketResponder
PacketResponder 0 for block blk_-1608999687919862906 terminating
Log3: 081109 203519 147 INFO dfs.DataNode$PacketResponder Received
block blk_-1608999687919862906 of size 91178 from /10.250.14.224
```
```
Log content:
ℓ1: block namesystem allocateblock mnt hadoop mapred system job job jar
blk
ℓ2: packetresponder for block blk terminating
ℓ3: received block blk of size from
```
Offline training
History Log
New Log
Online detection
Preprocessing
uppercase to lowercase...
Log Sequence Partition
Sliding window or session window
Log sequence: $\mathcal{S}_1 = \{\ell_1, \ell_2, \ldots, \ell_{|S_1|}\}, \mathcal{S}_2 = \{\ell_1, \ell_2, \ldots, \ell_{|S_2|}\}, \mathcal{S}_3 = \ldots$
Predict
VAE Model Based on Statistical Features
Constructing log sequence statistical matrix:
tok_1 tok_2 ... tok_M
Sequence s_1 1356 168 ... 532
Sequence s_2 956 396 ... 288
......
Log sequence statistical feature vectors:
x_1^{sta} 1356 168 ... 532
x_2^{sta} 956 396 ... 288
......
Encoder
Decoder
Report anomaly detection results
Train
Transformer Model Based on Hypersphere
BERT
Embedding Layer
Transformer Encoder × 12
$Xsem\ \{lsem_1, \ldots, lsem_i, \ldots, lsem_{|S|}\}$
log semantic vector sequence
$P\{pos_1, \ldots, pos_i, \ldots, pos_{|S|}\}$
positional embeddings sequence
Normalization Layer
Hypersphere
Dropout Layer
Transformer Encoder
Transformer Encoder
Candidate anomaly log sequence set
Predict
Train

Figure 3: The framework of UMFLog

4.3 Preprocessing

In order to preserve parameter information as much as possible and avoid log parsing errors, this paper replaces the log parsing with preprocessing and obtains the log content consisting of a set of word tokens. Specifically, each uppercase letter is first converted into lowercase letters. This paper then uses common delimiters in logging systems (i.e., spaces, colons, commas, etc.) to split the content into a set of symbolic tokens and remove all non-character tokens such as operators, punctuation, and numbers from the symbolic tokens, to get log content. It is worth noting that all character parameters are preserved for extracting the multi-feature. This is because non-character tokens usually represent variables and do not provide semantic information. Instead, character parameters always provide semantic information.

Fig. 3 shows the example of preprocessing the raw log content "BLOCK: NameSystem.allocate Block:/mnt/Hadoop/mapred/system/job_200811092030_001/job.jar.blk_-1608999687919862906" convert to word tokens "block namesystem allocateblock mnt hadoop mapred system job job jar blk", where the non-character tokens "200811092030_001, 1608999687919862906" are deleted, and the character parameters "mnt hadoop mapred system job job jar blk" are preserved.

4.4 Log Sequence Partition

The order of log messages provides the necessary information for anomaly detection. The order of log messages represents the execution path of a program, and the wrong execution order may mean an exception occurred. A single log message without the order and context information cannot provide enough information or features to detect order anomalies. Therefore, all log contents can be divided into several log sequences by windows to capture the sequential features. The partition methods include sliding windows, fixed windows, and session windows. For the session window, all log contents that belong to a user constitute a log sequence. As a result, the length of log contents within a session window varies. For fixed and sliding windows, all log contents are divided in chronological order by the window size, which determines the length of the log contents within the window.

4.5 Transformer Model Based on Hypersphere

To address the lack of log semantic problems in challenge 2, this paper designs the hypersphere-based Transformer model, which is specialized for extracting semantic information from log content and mining compact latent representations of log sequences.

4.5.1 Semantic Feature Extraction

The feature representations of log sequences are significant because poor log sequence representations often affect the effect of anomaly detection models. An ideal representation of a log sequence should be able to represent the differences between different log sequences, and the representation of log sequences with the same or similar semantics should be close.

Word embedding methods such as Word2vec have been widely applied to convert words in log content into vectors representing log messages. However, Word2vec cannot extract the context semantics. It may encode a word with different contexts and semantics into an identical embedding vector and two different words with the same or similar semantics into two orthogonal vectors, which would confuse downstream tasks.

Instead of Word2vec or simply random initialization, this paper utilizes the BERT encoder of the off-the-shelf online service (BERT-as-service) to perform word embeddings. The reason is that BERT can extract context semantics and exploit subword tokenization to handle OOV tokens. Specifically, subword tokenization splits unknown compound tokens into basic words. For example, the rare word "publickey" is split into more frequent subwords: {"public", "key"}. In this way, the number of OOV tokens is reduced. The embedding of the OOV tokens is expressed by all the base words in its sub-word set.

The simplified structure of BERT is shown in Fig. 4, which contains 12 layers of Transformer encoders. Because of the maximum input length limitation in BERT, this paper encodes the log content for each sequence separately. Firstly, BERT adds two extra tokens for each log content $\mathfrak{l}\left\{tok_1, tok_2, \ldots, tok_{|\mathfrak{l}|}\right\}$ to get a new sequence $\left\{CLS, tok_1, tok_2, \ldots, tok_{|\mathfrak{l}|}, SEP\right\}$, where Classification Token (CLS) is the log content start token, and Special Token (SEP) is the log content end token. Then, the embedding layer generates an initial embedding vector $\mathfrak{e}sem$ for each token and feeds them into the Transformer encoder. Next, the Transformer encoder adds the position information of each log token and learns the tokens' embedding representation by self-attention. This paper takes the embedding representation of the last layer as the token semantic vector $\mathfrak{w}sem$. The log content semantic vector $\mathfrak{l}sem$ is defined by the average of its corresponding log token semantic vector, as shown in Eq. (1),

$$\mathfrak{l}sem = \frac{\mathfrak{w}sem_{cls} + \ldots + \mathfrak{w}sem_i + \ldots + \mathfrak{w}sem_{sep}}{|\mathfrak{l}| + 2} \tag{1}$$

where $\mathfrak{w}sem_{cls}$ is the token semantic vector of the start token CLS, $\mathfrak{w}sem_{sep}$ is the token semantic vector of the end token SEP, $\mathfrak{w}sem_i$ is the i-th token semantic vector, and $|\mathfrak{l}|$ is the number of word tokens in the log content $\mathfrak{l}$.

In this way, each log message can be represented by the log content semantic vector $\mathfrak{l}sem$.

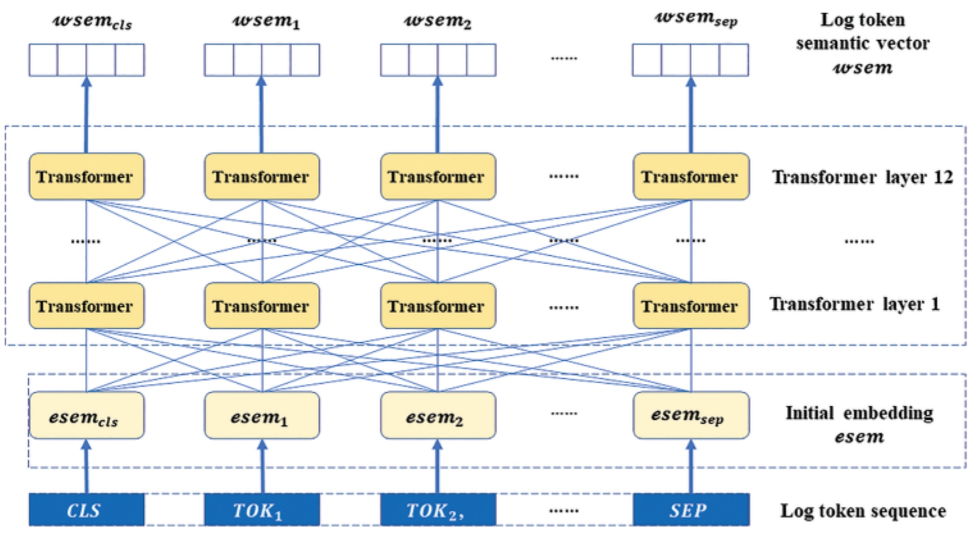


Figure 4: The simplified structure of BERT

4.5.2 Transformer Model

Transformer is designed for sequence-to-sequence encoding-decoding problems such as language translation and can process sequences of arbitrary length in parallel. As shown in Fig. 5, the Transformer model includes a position embedding module and a Transformer encoder module.

A. Position embedding module

The order in the log sequence implies the execution path. According to feature extraction, BERT represents a log message as a fixed-dimensional semantic vector ℓsem, where log messages with similar semantics are closer. However, these semantic vectors do not contain the order information in the log sequence.

Therefore, to represent the order information, the position encoding module encodes the position embedding $\wp os_t$ of the log message based on the sine and cosine functions, as shown in Eq. (2),

$$\wp os_t^{(i)} = \begin{cases} \sin(w_i t), & t = 2i \\ \cos(w_i t), & t = 2i+1 \end{cases}, w_i = \frac{1}{10000^{\frac{2i}{d}}}, i = 0, 1, \ldots, \frac{d}{2} - 1 \tag{2}$$

where $\wp os_t^i$ represents the i-th element in this position vector $\wp os$, t is the position of the log content ℓ_t in the log sequence $\mathbb{S}$, d is the dimension of the log content semantic vector ℓsem.

The positional embeddings of all log messages within a window constitute the sequence of positional embeddings $\mathcal{P}\{\wp os_1, \wp os_2, \ldots, \wp os_{|\mathbb{S}|}\}$,where $|\mathbb{S}|$ is the total number of log messages in the window. Then, a new log vector x_t is obtained by adding the log semantic vector ℓsem_t and the position embedding $\wp os_t$, that is, $x_t = \ell sem_t + \wp os_t$. The new sequence of log vectors $\mathfrak{X}sem \left\{x_1, \ldots, x_t, \ldots, x_{|\mathbb{S}|}\right\}$ is used to extract the representation of the log sequence by the Transformer encoder in the following Transformer encoder module.

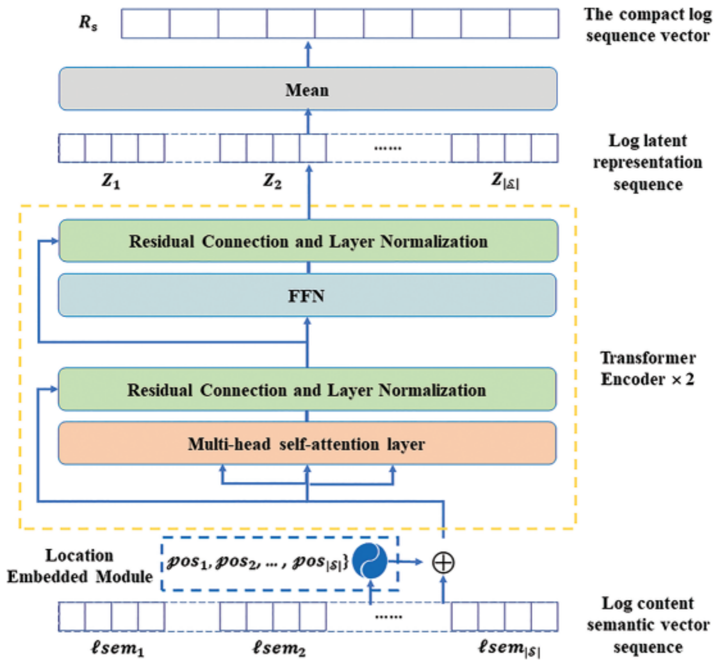

R_s
The compact log sequence vector
Mean
Log latent representation sequence
Z_1
Z_2
$Z_{|\mathbb{S}|}$
Residual Connection and Layer Normalization
FFN
Transformer Encoder × 2
Residual Connection and Layer Normalization
Multi-head self-attention layer
Location Embedded Module
$\{\wp os_1, \wp os_2, \ldots, \wp os_{|\mathbb{S}|}\}$
Log content semantic vector sequence
ℓsem_1
ℓsem_2
$\ell sem_{|\mathbb{S}|}$

Figure 5: Transformer model

B. Transformer encoder module

Transformer encoder is exploited to extract the features of log sequences. The Transformer encoder consists of a multi-head self-attention layer and a feed-forward network layer. The multi-head attention layer computes the attention score matrix for each log message with different attention patterns by training the query matrix and key-value matrix of the attention layer. Then, the output of the self-attention layer is fed back to a feed-forward network (FFN) layer, which contains two fully connected layers to achieve a combination of different attention scores. Finally, the output of the Transformer encoder is fed to the Dropout layer and the Normalization layer to obtain a sequence of latent representations $\mathfrak{Z}sem\left\{z_1, \ldots, z_i, \ldots, z_{|\mathfrak{S}|}\right\}$ for the log message.

Then the compact log sequence vector representation $R_{\mathfrak{S}}$ is obtained by the mean of the latent representations of all log messages in the sequence, as shown in Eq. (3),

$$R_{\mathfrak{S}} = \frac{1}{|\mathfrak{S}|}\sum\nolimits_{i=1}^{|\mathfrak{S}|} z_i \tag{3}$$

where z_i represents the latent representation of the i-th log message in a window sequence and $|\mathfrak{S}|$ is the total number of log messages in the sequence.

4.5.3 Hypersphere Constraint

As an unsupervised model, this paper takes hypersphere as the loss function. Within the constraint of the hypersphere, an ideal log message sequence representation should ensure that normal log sequence vector representations are closer to each other than abnormal log sequence vector representations. This paper constrains the normal log sequences for training inside a hypersphere to learn compact log sequence representations.

The center representation R_c of the hypersphere is the mean of the log sequence vectors of all training log sequences, as shown in Eq. (4),

$$R_c = \frac{1}{|\mathfrak{T}|}\sum\nolimits_{i=1}^{|\mathfrak{T}|} R_{\mathfrak{S}i} \tag{4}$$

where $|\mathfrak{T}|$ represents the total number of log sequences in the dataset and $R_{\mathfrak{S}_i}$ represents the i-th log sequence vector.

The training objective is to reduce the distances between R_s and R_c as much as possible. The distances are defined by the mean squared error loss $Loss_{mse}$, as shown in Eq. (5), and the training objective is to minimize $Loss_{mse}$.

$$Loss_{mse} = \frac{1}{|\mathfrak{T}|}\sum\nolimits_{i=1}^{|\mathfrak{T}|} \left\| R_{s_i} - R_c \right\|^2 \tag{5}$$

The hypersphere loss can capture the intrinsic differences between normal and abnormal log sequences. The trained model keeps the normal log sequence vector inside the hypersphere as much as possible, while the abnormal log sequence should be outside the hypersphere and away from the center. The distance to the center indicates whether the log sequence is abnormal.

4.6 VAE Model Based on Statistical Features

To address challenge 3, which is to make full use of the rich information in logs to construct multi-features, this paper counts the occurrence times of each token in the log sequences and builds the VAE model to further improve the accuracy of anomaly detection.

4.6.1 Statistical Feature Extraction

The occurrences times of each ***tok*** in the log sequence directly imply the regularity of normal and abnormal patterns in the log sequence. Usually, some ***tok***s only appear in the abnormal sequences. As a result, the statistical feature of the log sequence can be extracted to detect anomalies.

For the training set $\mathfrak{T}_1$, this paper constructs a standard statistic dictionary $\boldsymbol{D}_{\mathfrak{T}}$ as shown in Eq. (6).

$$\mathrm{D}_{\mathfrak{T}} = \left\{\mathrm{tok}_1\text{: } \mathrm{Sta}^{\mathrm{tok}_1}, \ldots, \mathrm{tok}_i\text{: } \mathrm{Sta}^{\mathrm{tok}_i}, \ldots, \mathrm{tok}_M\text{: } \mathrm{Sta}^{\mathrm{tok}_{M_{\mathrm{train}}}}\right\} \tag{6}$$

where tok_i represents the token, M_{train} is the total number of tokens, Sta^{tok_i} represents the normalized occurrences times of tok_i in all log sequences of $\mathfrak{T}_1$. The normalization method is max-min normalization which scale the occurrences times in [0,1].

For a given log token sequence $s\left\{tok_1^{\ell_1}, \ldots, tok_{|\ell_1|}^{\ell_1}, \ldots, tok_1^{\ell_{|s|}}, \ldots, tok_{\left|\ell_{|s|}\right|}^{\ell_{|s|}}\right\}$, each token looks up its standard statistical times according to the standard statistic dictionary $D_{\mathfrak{T}}$, and constructs a statistics vector $\mathfrak{X}sta$, as shown in Eq. (7). The dimension is $M_{train} + N$.

$$\mathfrak{X}sta[j] = \begin{cases} D_{\mathfrak{T}}\left[tok_j\right], & if \quad tok_j \in s, 0 < j \le M_{train} \\ 0, & if \quad tok_j \notin s, 0 < j \le M_{train} \\ 0, & if \quad M_{train} < j \le M_{train} + N \end{cases} \tag{7}$$

where j indicates the *index* of token in $D_{\mathfrak{T}}$, M_{train} is the number of all tokens of $D_{\mathfrak{T}}$. Obviously, the first M_{train} dimension of the statistics vector corresponds to the M_{train} tokens in $D_{\mathfrak{T}}$. To handle the tokens which do not appear in $\mathfrak{T}_1$, that is, OOV tokens, this paper reserves the latter N dimensions for the OOV tokens.

Therefore, each log sequence can be represented by a statistical vector $\mathfrak{X}sta$, which helps improve anomaly detection accuracy.

4.6.2 Variational Autoencoder Model

Variational autoencoder is exploited to learn the patterns implicit in the statistical features of log sequences. The VAE consists of two Multilayer Perceptron (MLP) neural networks, an encoder (denoted by $\mathrm{E}\phi\ (z|x)$) and a decoder (denoted by $D\theta\ (x|z)$), as shown in Fig. 6, where x represents the input data, z represents the latent variable, ϕ and θ represent the parameters of the encoder and decoder respectively. The mean μ and the covariance σ are generated by the encoder, and the encoder converts x into the probability distribution $\wp_\theta\ (z|x)$ of the latent variables, and then randomly samples the latent variable z with a standard normal distribution $\in$, which is decoded to reconstruct the output x' by the decoder.

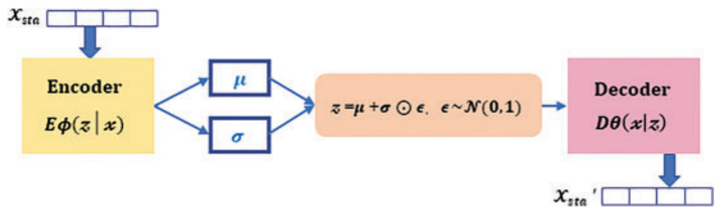


Figure 6: Variational Autoencoder model

The loss function consists of the reconstruction term for optimizing the encoder and decoder and the regularization term for regularizing the hidden space, in Eq. (8). The reconstruction term is the expectation of the decoder output $\mathcal{P}_\theta(x|z)$ given the encoder output $\mathcal{Q}_\phi(z|x)$. The larger the expectation, the smaller the reconstruction loss. The regularization term is the Kullback-Leibler divergence(KL-divergence) of $\mathcal{Q}_\phi(z|x)$ and $\mathcal{P}_\theta(z|x)$. The smaller the KL-divergence, the smaller the difference between the two distributions.

$$\max_{\phi,\theta}\left(E_{z\sim\mathcal{Q}_\phi(z|x)}\left(log\left(\mathcal{P}_\theta(x|z)\right)\right) - D_{KL}(\mathcal{Q}_\phi(z|x)||\mathcal{P}_\theta(z))\right) \tag{8}$$

where $\mathcal{P}_\theta(x|z)$ is the Gaussian distribution randomly sampled from the latent variable z, $\mathcal{P}_\theta(z)$ is the prior distribution randomly sampled from the hidden variable, and $\mathcal{Q}_\phi(z|x)$ is the Gaussian distribution used to approximate $\mathcal{P}_\theta(z|x)$ during variational inference.

4.7 Anomaly Detection

After the offline training, this paper obtains a trained hypersphere-based Transformer model and a statistical feature-based VAE model for log sequence anomaly detection. When new log messages arrive, the same preprocessing and log sequence partition steps are performed. Then, the trained Transformer model with hypersphere extracts compact log sequence representations and filters candidate abnormal log sequences. Finally, the candidate abnormal log sequence is further detected by the trained VAE.

4.7.1 Constructing the Candidate Anomaly Log Sequence Set

When the distance between a new log sequence and the center of the hypersphere is larger than the radius threshold, the log sequence is considered abnormal and added to the candidate set G. This paper takes the validation set to set the radius threshold. The mean distance between the hypersphere center R_c and all normal log sequences in the validation set is set as the initial radius threshold, which is then fine-tuned by all abnormal log sequences in the validation set.

4.7.2 Anomaly Detection Using the Trained VAE Model

Similar to the training set $\mathcal{T}_1$, this paper constructs a standard statistical dictionary $D_\mathcal{V}$ for the validation set, denoted by Eq. (9).

$$D_\mathcal{V} = \left\{tok_1\colon Sta^{tok_1}, \ldots, tok_i\colon Sta^{tok_i}, \ldots, tok_M\colon Sta^{tok_{M_{val}}}\right\} \tag{9}$$

where tok_i represents the token, which appears in normal log sequences of the validation set and not in the training set, M_{val} is the total number of this type of token, Sta^{tok_i} represents the normalized value of occurrences times of tok_i in all normal log sequences of $\mathcal{T}_2$. This paper creates an OOV token list $L_{OO\mathcal{V}}$ for reserved OOV tokens, which is empty on initialization and then updated.

For each token in a given log token sequence $\mathfrak{s}\left\{tok_1^{\ell_1}, \ldots, tok_{|\ell_1|}^{\ell_1}, \ldots, tok_1^{\ell_{|\mathfrak{s}|}}, \ldots, tok_{\left|\ell_{|\mathfrak{s}|}\right|}^{\ell_{|\mathfrak{s}|}}\right\}$ in $\mathcal{T}_2$ or G, if it does not belong to $D_\mathcal{T}$ or $D_\mathcal{V}$, it is an OOV token and appended into the list $L_{OO\mathcal{V}}$, denoted by Eq. (10).

$$L_{OO\mathcal{V}} = \{oov_1, \ldots, oov_i, \ldots\} \tag{10}$$

Next, the sequence statistics vector $\mathfrak{X}sta$ is constructed according to $D_{\mathfrak{T}}$, $D_{\mathfrak{V}}$ and $L_{OO\mathfrak{V}}$, and the dimension of the vector is $M_{train} + N$, as shown in Eq. (11).

$$\mathfrak{X}sta[j] = \begin{cases} D_{\mathfrak{T}}\left[tok_j\right], & if \quad tok_j \in \mathfrak{s}, 0 < j \le M_{train} \\ D_{\mathfrak{V}}\left[tok_{j-M_{train}}\right], & if \quad tok_{j-M_{train}} \in \mathfrak{s}, M_{train} < j \le M_{train} + M_{val} \\ 1.5, & if \quad oov_{j-M_{train}-M_{val}} \in \mathfrak{s}, M_{train} + M_{val} < j \le M_{train} + +M_{val} + |L_{OO\mathfrak{V}}| \\ 0, & if \quad tok_j \in \mathfrak{s}, M_{train} + M_{val} + |L_{OO\mathfrak{V}}| < j \le M_{train} + N \\ 0, & if \quad tok_j \notin \mathfrak{s}, 0 < j \le M_{train} + M_{val} + |L_{OO\mathfrak{V}}| \end{cases} \tag{11}$$

where j is the index, M_{train} is the number of all tokens in $D_{\mathfrak{T}}$, M_{val} is the number of all tokens in $D_{\mathfrak{V}}$, and N is the number of the reserved OOV tokens defined during the training phase.

The VAE model obtains the reconstruction error of the validation set $\mathfrak{T}_2$, and the ninetieth percentile of all reconstruction errors is set as the threshold for anomaly detection. For each log sequence in the candidate abnormal log sequence set G, if the reconstruction error from VAE is greater than the threshold, the log sequence is abnormal.

5 Experimental Evaluation

This section demonstrates the effectiveness and superiority of UMFLog for log sequence anomaly detection by answering the following questions:

- RQ1: How effective is UMFLog?
- RQ2: How sensitive is UMFLog to the parameters?
- RQ3: How effective is log multi-feature on log anomaly detection?
- RQ4: How effective are the log data input form, embedding method, and log statistical feature method of UMFLog?

5.1 Dataset and Experimental Environment

To evaluate the performance of UMFLog, this paper conducts extensive experiments on three public datasets: HDFS, BGL, and Thunderbird, which are also extensively studied in existing work.

The HDFS dataset contains 11,175,629 log messages collected from the Hadoop distributed file system on the Amazon EC2 platform. The BGL dataset contains 4,747,963 log messages collected from the BGL supercomputer at Lawrence Livermore National Laboratory, of which 348,460 logs were flagged as anomalous. Thunderbird contains 211,212,192 logs collected from real-world supercomputers at Sandia National Laboratories, and this paper uses the first 5 million logs. In addition, log messages with tokens of less than five are excluded for both the BGL and Thunderbird datasets. For the HDFS dataset, this paper uses the session window marked by the Block ID to divide the log sequence, while for the BGL and Thunderbird datasets, it uses the sliding window. The details of the three datasets are shown in Table 4.

All experiments in this paper are run on Google COLAB, an artificial intelligence development platform for machine learning and data analysis developed by the Google Research team, equipped with an NVIDIA TESLA P100 GPU server with 16 G memory. In addition, this paper uses the Python 3.7 environment to build the Pytorch model.

Table 4: Details of the dataset

Dataset	System category	Duration	# Log messages	#Normal log sequence	#Abnormal log sequence
HDFS	Distributed Systems	38.7 h	11175629	558223	16838
BGL	Supercomputer	214.7 days	1181552	264258	31126
Thunderbird	Supercomputer	244 days	4010726	632447	370230

5.2 Evaluation Metrics

The widely used metrics of precision, recall, and F1-Score are used to evaluate the performance.

Precision: The percentage of log sequences that are truly abnormal among all log sequences judged by the model to be abnormal, as shown in Eq. (12).

Recall: The percentage of abnormal log sequences correctly identified by the model among all abnormal log sequences, as shown in Eq. (13).

F1-Score: The harmonic mean of precision and recall, as shown in Eq. (14).

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$\text{F1}-\text{score} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{14}$$

whereas TN is the number of normal log sequences correctly detected by the model, FP is the number of abnormal log sequences incorrectly identified by the model as normal, FN is the number of anomalous log sequences not detected by the model, and TP is the number of anomalous log sequences correctly detected by the model.

5.3 Baseline

To verify the rationality and advancement of UMFLog, this paper selects various SOTA unsupervised methods as baselines. A brief introduction to the baseline method is as follows:

PCA [33] is a Principal Component Analysis (PCA)-based approach that extracts major components of input sequence features and reports test sequences above the threshold as anomalies.

OC4Seq [35] is a Gated Recurrent Neural Networks (GRU)-based approach with hypersphere objective loss functions of both global and local perspectives.

DeepLog [17] is an LSTM-based model that predicts the next log template index.

LogAnomaly [18] is an LSTM-based model that extracts semantic information hidden in log templates by a collection of synonyms and antonyms.

LogBERT [26] is a BERT-like model that trains log template index sequences with both masked template index prediction loss and hypersphere objective loss.

CAT [28] is a Transformer-based model that fine-tunes BERT to extract semantic information from log sequences and trains variable-length log semantic sequences with both masked subsequence prediction loss and hypersphere objective loss.

5.4 *Experimental Settings*

This paper uses a sliding window with a window size of 20 and a stride of 4. The training set size is 160,000 normal log sequences, and the validation set includes 9,000 normal log sequences and 1,000 abnormal log sequences. All the remaining normal and abnormal log sequences are used as test sets.

Other experimental details include that the number of layers L of the Transformer encoder is 2, the number of multi-head self-attention heads is 12, the hidden layer dimension is 2048, the semantic embedding dimension D is 768, the encoder encoding dimension d_e of VAE is 512, and the latent variable dimension d_h is 256.

5.5 *RQ1: The Effectiveness of UMFLog*

To answer RQ1, this paper first compares UMFLog with all the above baseline unsupervised methods, and Table 5 shows the best experimental results for the baseline and UMFLog.

In general, most of the baseline methods perform unevenly on the three datasets. Although some methods perform well on some datasets, they perform poorly on others. For example, PCA achieved an F1 score of 0.9497 on the HDFS dataset, but performed poorly on both the BGL and Thunderbird datasets, thus achieving the lowest mean F1 score. Some methods obtain relatively high precision, such as LogBERT in HDFS and PCA in the Thunderbird dataset, but their recall is relatively low because many undetected abnormal logs are missed. Some methods also achieve high recall, such as DeepLog and LogAnomaly in the Thunderbird dataset, but their precision is relatively low, which means that many false anomaly alerts are generated. In addition, the F1 scores of semantic feature-based methods (CAT, LogAnomaly) on the three datasets are generally higher than those of other methods without semantic features (PCA, OC4Seq, LogBERT, DeepLog). This shows the superiority of log sequence anomaly detection with semantics.

Overall, it can be observed that UMFLog surpasses all baseline methods in F1 scores on all three datasets with robust performance advantages, demonstrating the superior ability of the proposed multi-feature method on the task of log sequence anomaly detection.

Table 5: Overall performance evaluation results

Dataset	HDFS			BGL			Thunderbird			Average		
Method	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
PCA	0.91	**0.993**	0.9497	0.352	0.1709	0.2301	0.8981	0.5846	0.7082	0.720	0.583	0.629
OC4Seq	**0.998**	0.955	0.976	0.704	0.795	0.747	0.8783	0.8748	0.8771	0.860	0.875	0.867
LogBERT	0.8702	0.781	0.8232	0.894	0.9232	0.9083	0.9675	0.9652	0.9664	0.911	0.890	0.899
DeepLog	0.95	0.96	0.96	0.88	**1**	0.93	0.7849	0.9996	0.8795	0.872	0.987	0.923
LogAnomaly	0.96	0.94	0.95	0.97	0.94	0.96	0.7996	**1**	0.8884	0.910	0.960	0.933
CAT	0.8837	0.9063	0.8947	0.8893	0.9554	0.9233	0.9935	0.9897	0.9916	0.922	0.950	0.937
UMFLog	0.99373	**1**	**0.99685**	**0.99533**	0.99071	**0.99301**	**0.99726**	0.99677	**0.99701**	**0.995**	**0.996**	**0.996**

5.6 RQ2: The Parameter Sensitivity of UMFLog

In this part, this paper evaluates the parameter stability of UMFLog by conducting experimental evaluations on the HDFS dataset with different parameters, including Transformer encoder layers L and VAE encoding dimension d_e and hidden dimension d_h.

The Transformer encoder layer L is set from 1 to 4. As shown in Fig. 7A, UMFLog can achieve good performance even with only one layer of the Transformer encoder. In addition, as the number of layers increases, the F1 score increases but then decreases, and the best results are obtained when the number of layers L is 2. The increase in the number of layers of the Transformer encoder allows the model to fit more complex log sequence features but may also introduce unwanted training noise, as well as longer training and prediction times. d_e and d_h are set to 64–128, 128–25, and 256–512 respectively. Fig. 7B shows that UMFLog is not very sensitive to these two dimension parameters on HDFS data and can achieve stable performance in different situations.

In general, UMFLog is not sensitive to parameters and can obtain stable performance. The parameters of UMFLog are set to L = 2, d_e = 512, and d_h = 256.

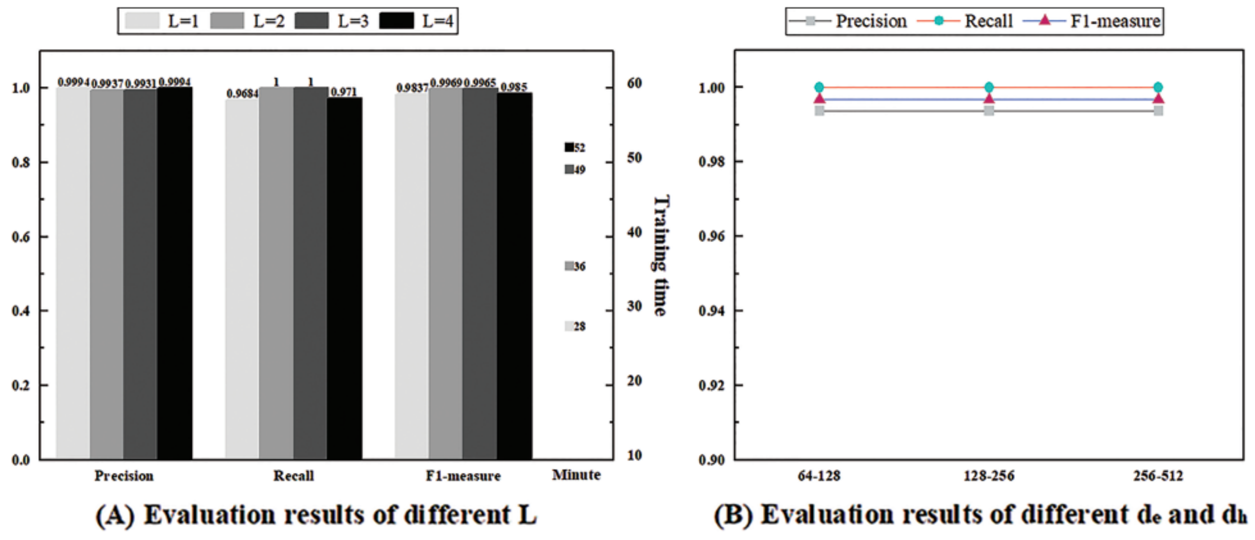


Figure 7: The result of parameter sensitivity experiment

5.7 RQ3: The Effectiveness of Log Multi-feature

This subsection conducts ablation experiments to evaluate the effectiveness of multi-feature. UMFLog without VAE follows UMFLog and removes the statistical feature-based VAE. As shown in Figs. 8A–8C, the performance of anomaly detection on all three datasets reduces without the statistical feature-based VAE model. For Thunderbird logs, the F1 score is reduced by 7%, while precision is reduced by 13%, which means more false alarms are generated. This proves that the statistical feature-based VAE model can effectively improve the accuracy of log sequence anomaly detection and reduce false positives.

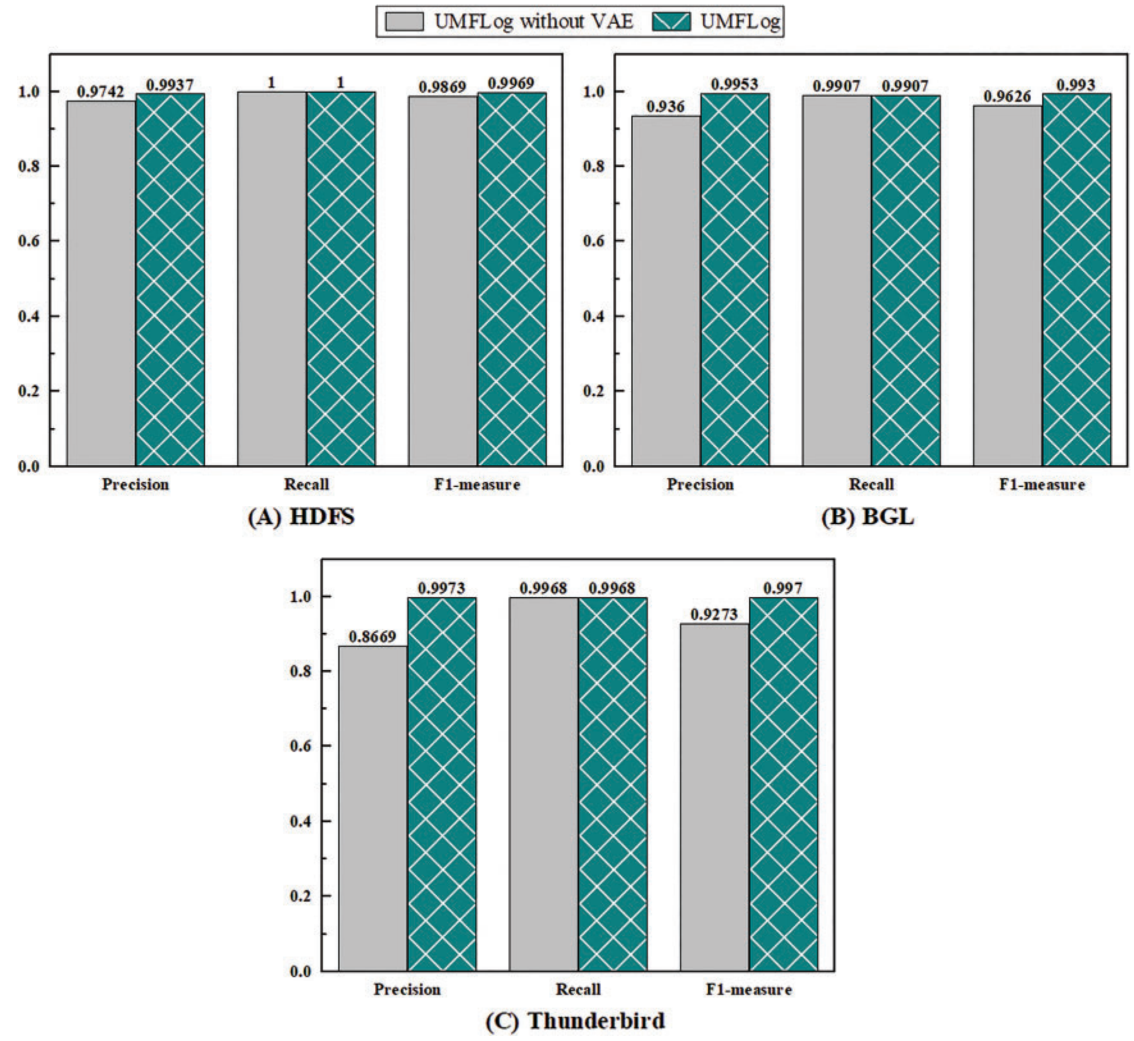


Figure 8: Multi-feature model ablation experiment on three datasets

5.8 RQ4: The Impact of Input Form, Embedding Method, and Log Statistical Feature Method

Section 2 summarizes the existing hypersphere-based log anomaly detection methods, and makes a detailed comparison between UMFLog and them in terms of input form and feature embedding method. In order to answer question 4, this subsection further carries out ablation experiments to discuss the impact of different input forms, embedding methods, and statistical feature methods on UMFLog.

In term of input form and embedding method, UMFLog uses log content and BERT. There are other optional methods: log template for input form and Word2vec for embedding. Therefore, there are four different schemes: Template with Wod2vec, Template with BERT, Content with Word2vec, and Content with BERT. Only three of these schemes are compared because Content with Word2vec exceeds the computational capabilities.

Figs. 9A–9C shows the impact of different input forms and log content embedding methods. The results show that log content with BERT has the best performance on the three datasets, which confirms the importance of preserving the character parameters and extracting context semantics.

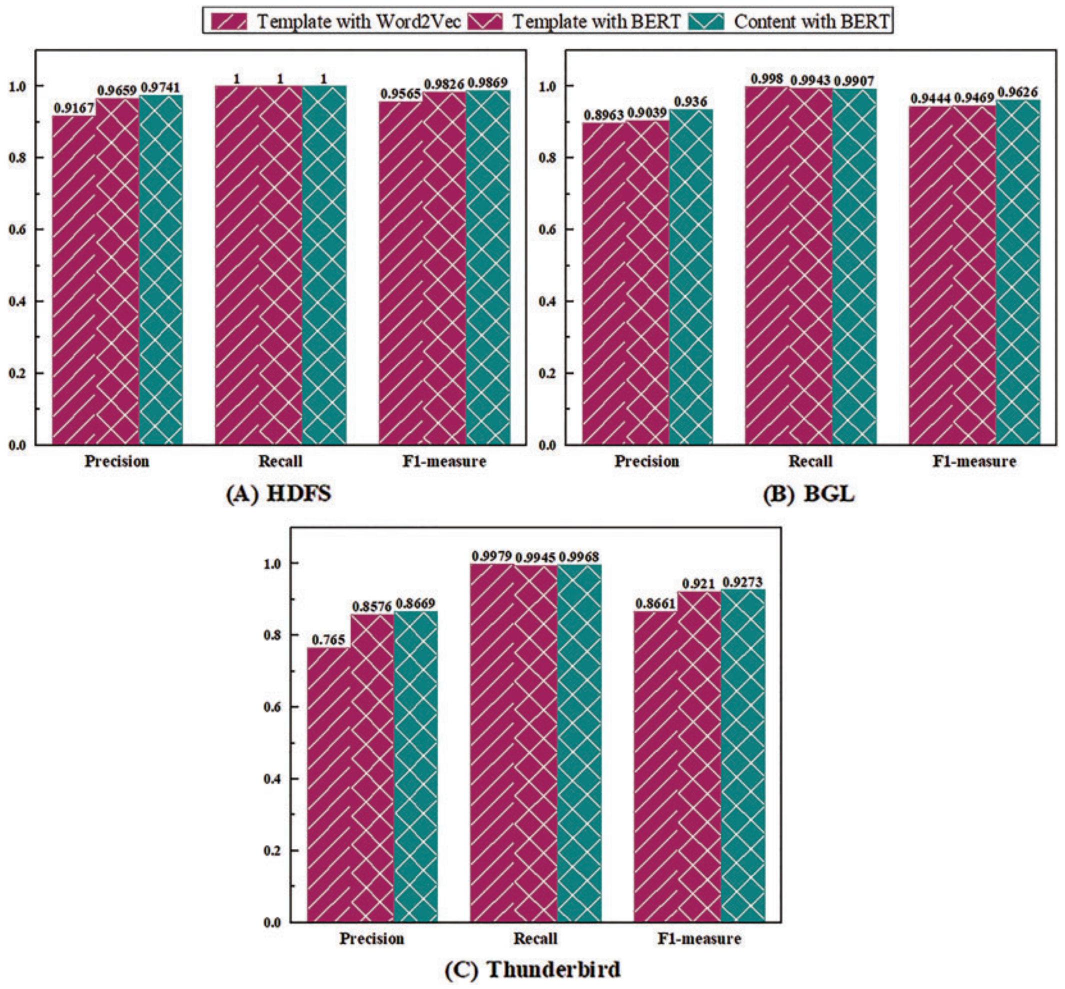


Figure 9: The performance evaluations with different input forms and log content embedding methods on three datasets

Next, different ways of statistical features are compared. In Section 4.6.1, this paper constructs a dictionary-based statistical feature utilizing a standard statistic dictionary and reserved OOV tokens. It is inspired by the fact that some tokens often appear in abnormal sequences but rarely in normal sequences. However, the appearance order and co-occurrence pattern of the tokens constituting the log sequence also contain certain regularities. Based on this intuition, this paper proposes the order-based statistical feature and compares it with the dictionary-based statistical feature.

For a given log token sequence $s \left\{ tok_1^{\ell_1}, \ldots, tok_{|\ell_1|}^{\ell_1}, \ldots, tok_1^{\ell_{|s|}}, \ldots, tok_{\left|\ell_{|s|}\right|}^{\ell_{|s|}} \right\}$, the order-based statistical feature constructs the statistical vector according to the appearance order of the tokens.

According to the training set statistical times dictionary $D_{\mathfrak{T}}$, the temporary order statistical vector $\mathfrak{s}^{sta}$ is as shown in Eq. (15).

$$\mathfrak{s}^{sta} = \left[Sta^{tok_1^{\ell 1}}, Sta^{tok_2^{\ell 1}}, \ldots, Sta^{tok_1^{\ell |\mathfrak{s}|}}, \ldots, Sta^{tok_{\left|\ell_{|\mathfrak{s}|}\right|}^{\ell |\mathfrak{s}|}} \right] \tag{15}$$

where $Sta^{tok_n^{\ell m}}$ is the standard statistical time value of the n-th token of the m-th log content in the log token sequence $\mathfrak{s}$, and the dimension of the vector is the length $|\mathfrak{s}|$ of the log token sequence $\mathfrak{s}$.

However, the number of tokens in each log sequence varies. The dimension of the temporary order statistics vector $\mathfrak{s}^{sta}$ is not uniform and cannot be fed into the VAE model. Therefore, this paper sets a unified vector dimension d_{sta} by the average or maximum number of log sequence tokens in all training sets $\mathfrak{T}_1$. When the dimension $|\mathfrak{s}|$ is larger than the unified vector dimension d_{sta}, it is truncated directly, and only the first d_{sta} dimensions are retained; otherwise, the last d_{sta}-$|\mathfrak{s}|$ dimensions are padded with 0.

MEAN sets the unified vector dimension as the average value of the token numbers of all log sequences in the training set $\mathfrak{T}_1$, while MAX uses the maximum value, and vocabulary (VOCAL) means dictionary-based statistical features of UMFLog. Figs. 10A–10C shows the results with different ways of statistical features. The three methods can achieve good performance on the three datasets, especially the HDFS and BGL datasets. In contrast, on the Thunderbird dataset, the performance of the three methods is not much different. Dictionary-based statistical features achieve the best performance compared with order-based statistical features. Order-based statistical features can provide information about the order of token occurrences, but at the same time, it is redundant because the same token may appear multiple times. The average method weakens this effect compared with the maximum method. Obviously, the stable performance results indicate that all three approaches can provide valid and sufficient statistical features.

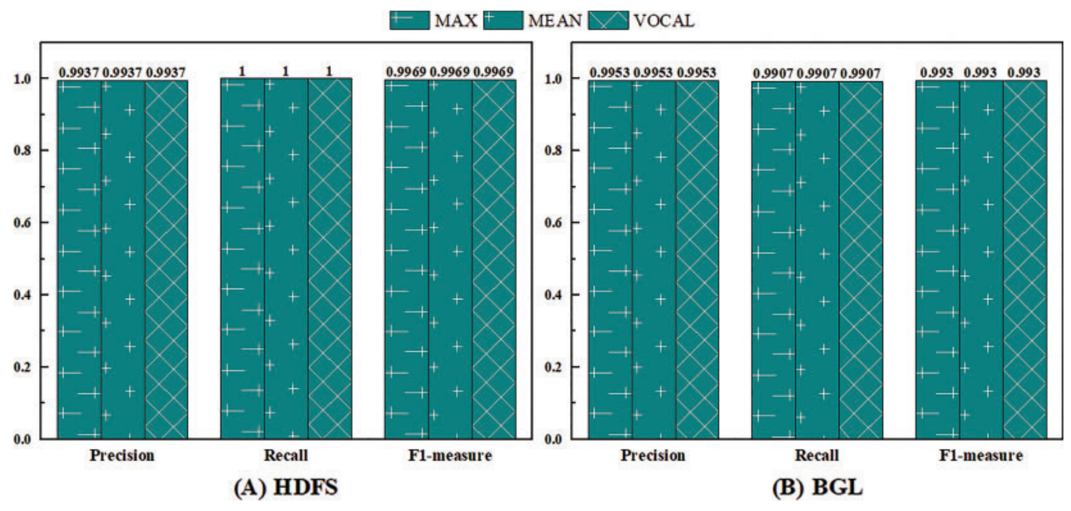
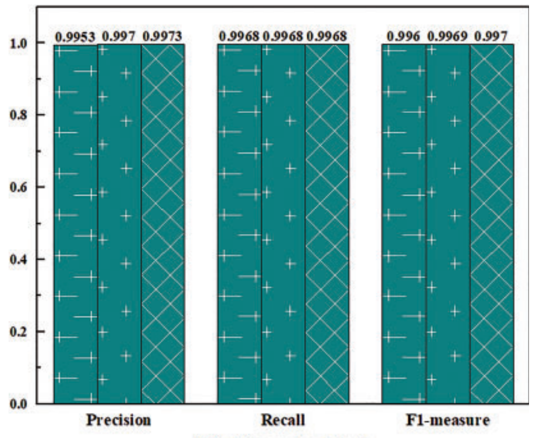


Figure 10: (Continued)



(C) Thunderbird

Figure 10: The performance of UMFLog with different statistical features on three datasets

6 Conclusion

Log anomaly detection methods help systems with troubleshooting and problem analysis. Existing methods still have some challenges, and this paper proposes UMFLog, an unsupervised log anomaly detection method based on multi-feature, to address these challenges. UMFLog uses a hypersphere-based Transformer model to learn log sequence representations from the unparsed log and obtain candidate abnormal sequences, and then uses a statistical feature-based VAE model to identify abnormal sequences among candidate abnormal sequences. Extensive experiments have been conducted on real-world datasets to evaluate the effectiveness and robustness of UMFLog. The results show that our approach outperforms the SOTA methods. In the future, we plan to consider other log features for log anomaly detection to take full advantage of the rich information in logs.

Acknowledgement: The authors would like to thank the anonymous reviewers for their valuable comments and suggestions that improve the presentation of this paper.

Funding Statement: This work was supported in part by the National Natural Science Foundation of China under Grant 62272062, the Scientific Research Fund of Hunan Provincial Transportation Department (No. 202143), the Open Fund of Key Laboratory of Safety Control of Bridge Engineering, Ministry of Education (Changsha University of Science Technology) under Grant 21KB07.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] B. Xiong, K. Yang, J. Zhao and K. Li, “Robust dynamic network traffic partitioning against malicious attacks,” *Journal of Network and Computer Applications*, vol. 87, no. 7, pp. 20–31, 2017.
[2] Z. Xia, J. Tan, K. Gu and W. Jia, “Detection resource allocation scheme for two-layer cooperative IDSs in smart grids,” *Journal of Parallel and Distributed Computing*, vol. 147, no. 3, pp. 236–247, 2021.
[3] Z. Xia, G. Long and B. Yin, “Confidence-aware collaborative detection mechanism for false data attacks in smart grids,” *Soft Computing*, vol. 25, no. 7, pp. 5607–5618, 2021.

[4] H. Gao, B. Qiu, R. J. D. Barroso, W. Hussain, Y. Xu *et al.,* "TSMAE: A novel anomaly detection approach for internet of things time series data using memory-augmented autoencoder," *IEEE Transactions on Network Science and Engineering*, vol. 99, pp. 1–14, 2022. https://doi.org/10.1109/TNSE.2022.3163144

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

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

[7] C. Bertero, M. Roy, C. Sauvanaud and G. Tredan, "Experience report: Log mining using natural language processing and application to anomaly detection," in *Int. Symp. on Software Reliability Engineering*, Toulouse, France, pp. 351–360, 2017.

[8] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang *et al.,* "An empirical investigation of practical log anomaly detection for online service systems," in *Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*, Athens, Greece, pp. 1404–1415, 2021.

[9] V. H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *2022 IEEE/ACM 44th Int. Conf. on Software Engineering*, Pittsburgh, PA, USA, pp. 1356–1367, 2022.

[10] S. He, Z. Li, Y. Tang, Z. Liao, F. Li *et al.,* "Parameters compressing in deep learning," *Computers, Materials & Continua*, vol. 62, no. 1, pp. 321–336, 2020.

[11] H. Gao, J. Xiao, Y. Yin, T. Liu and J. Shi, "A mutually supervised graph attention network for few-shot segmentation: The perspective of fully utilizing limited samples," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 99, pp. 1–13, 2022. https://doi.org/10.1109/TNNLS.2022.3155486

[12] H. Gao, J. Huang, Y. Tao, W. Hussain and Y. Huang, "The joint method of triple attention and novel loss function for entity relation extraction in small data-driven computational social systems," *IEEE Transactions on Computational Social Systems*, vol. 9, no. 6, pp. 1725–1735, 2022.

[13] L. Xiang, Y. Li, W. Hao, P. Yang and X. Shen, "Reversible natural language watermarking using synonym substitution and arithmetic coding," *Computers, Materials & Continua*, vol. 55, no. 3, pp. 541–559, 2018.

[14] L. Xiang, X. Wang, C. Yang and P. Liu, "A novel linguistic steganography based on synonym run-length encoding," *IEICE Transactions on Information and Systems*, vol. 100, no. 2, pp. 313–322, 2017.

[15] J. Zhang, X. Jin, J. Sun, J. Wang and A. K. Sangaiah, "Spatial and semantic convolutional features for robust visual object tracking," *Multimedia Tools and Applications*, vol. 79, no. 21, pp. 15095–15115, 2020.

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

[17] 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*, New York, NY, USA, pp. 1285–1298, 2017.

[18] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei *et al.,* "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Int. Joint Conf. on Artificial Intelligence*, Macao, China, pp. 4739–4745, 2019.

[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*, New York, NY, USA, pp. 807–817, 2019.

[20] X. Li, P. Chen, L. Jing, Z. He and G. Yu, "SwissLog: Robust anomaly detection and localization for interleaved unstructured logs," *IEEE Transactions on Dependable and Secure Computing*, vol. 99, pp. 1–18, 2022. https://doi.org/10.1109/TDSC.2022.3162857

[21] X. Han and S. Yuan, "Unsupervised cross-system log anomaly detection via domain adaptation," in *Proc. of the 30th ACM Int. Conf. on Information & Knowledge Management*, New York, NY, USA, pp. 3068–3072, 2021.

[22] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones *et al.,* "Attention is all you need," in *Advances in Neural Information Processing Systems*, Red Hook, NY, USA, pp. 5998–6008, 2017.

[23] S. R. Wibisono and A. I. Kistijantoro, “Log anomaly detection using adaptive universal transformer,” in *2019 Int. Conf. of Advanced Informatics: Concepts, Theory and Applications*, Yogyakarta, Indonesia, pp. 1–6, 2019.

[24] Z. Chen, J. Liu, W. Gu, Y. Su and M. Lyu, “Experience report: Deep learning-based system log analysis for anomaly detection,” Arxiv Preprint Arxiv: 210705908, 2021.

[25] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso and O. Kao, “Self-attentive classification-based anomaly detection in unstructured logs,” in *2020 IEEE Int. Conf. on Data Mining*, Sorrento, Italy, pp. 1196–1201, 2020.

[26] H. Guo, S. Yuan and X. Wu, “LogBERT: Log anomaly detection via BERT,” in *2021 Int. Joint Conf. on Neural Networks*, Shenzhen, China, pp. 1–8, 2021.

[27] Z. Zhao, W. Niu, X. Zhang, R. Zhang, Z. Yu *et al.,* “Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network,” *Applied Intelligence*, vol. 52, no. 8, pp. 8810–8819, 2022.

[28] S. Zhang, Y. Liu, X. Zhang, W. Cheng, H. Chen *et al.,* “CAT: Beyond efficient transformer for content-aware anomaly detection in event sequences,” in *Proc. of the 28th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, New York, NY, USA, pp. 4541–4550, 2022.

[29] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao *et al.,* “HitAnomaly: Hierarchical transformers for anomaly detection in system log,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[30] V. H. Le and H. Zhang, “Log-based anomaly detection without log parsing,” in *36th IEEE/ACM Int. Conf. on Automated Software Engineering*, Melbourne, Australia, pp. 492–504, 2021, 2021

[31] J. Devlin, M. -W. Chang, K. Lee and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” ArXiv Preprint ArXiv: 181004805, 2019.

[32] A. J. Oliner and J. Stearley, “What supercomputers say: A study of five system logs,” in *Dependable Systems and Networks*, Edinburgh, UK, pp. 575–584, 2007.

[33] W. Xu, L. Huang, A. Fox, D. A. Patterson and M. I. Jordan, “Detecting large-scale system problems by mining console logs,” in *Proc. of the ACM SIGOPS 22nd Symp. on Operating Systems Principles*, New York, NY, USA, pp. 117–132, 2009.

[34] J. Wang, C. Zhao, S. He, Y. Gu, O. Alfarraj *et al.,* “LogUAD: Log unsupervised anomaly detection based on word2Vec,” *Computer Systems Science and Engineering*, vol. 41, no. 3, pp. 1207–1222, 2022.

[35] Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen *et al.,* “Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection,” in *Proc. of the 27th ACM SIGKDD Conf. on Knowledge Discovery & Data Mining*, New York, NY, USA, pp. 3726–3734, 2021.