LogEncoder: Log-Based Contrastive Representation Learning for Anomaly Detection

Jiaxing Qi, Zhongzhi Luan, *Member, IEEE*, Shaohan Huang, Carol Fung, Hailong Yang, *Member, IEEE*, Hanlu Li, Danfeng Zhu, and Depei Qian

***Abstract*—In recent years, cloud computing centers have grown rapidly in size. Analyzing system logs is an important way for the quality of service monitoring. However, systems produce massive amounts of logs, and it is impractical to analyze them manually. Automatic and accurate log analysis to detect abnormal events in systems has become extremely important. However, due to the nature of the log analysis problem, such as discrete property, class imbalance, and quality of log, log-based anomaly detection remains a difficult problem. To address these challenges, we propose LogEncoder, a framework of log sequence encoding for semi-supervised anomaly detection. LogEncoder utilizes a pre-trained model to obtain a semantic vector for each log event. To separate normal and abnormal log event sequences and preserve their contextual information, we integrate one-class and contrastive learning objectives training into the representation model. Finally, we propose two methods, one for offline and one for online, to detect system anomalies. Compared to six state-of-the-art baselines on three benchmark datasets, LogEncoder outperforms five unsupervised and semi-supervised methods, and the performance is comparable to the supervised method LogRobust.**

***Index Terms*—Anomaly detection, system logs, deep learning, semi-supervised.**

I. Introduction

RECENTLY, with the rapid development of online services and mobile devices, Internet service providers (ISPs) have established cloud computing systems to improve the performance of online services. However, these systems are vulnerable to malicious attacks, resulting in service unavailability. This may cause substantial economic losses for ISPs. For example, Amazon Web service (AWS) suffered a critical interruption on December 8, 2021, at approximately 10:45 a.m. EST, resulting in significant economic losses [1]. Consequently, timely and effective detection of system abnormality is critical.

Manuscript received 29 April 2022; revised 18 August 2022, 2 November 2022, and 17 January 2023; accepted 19 January 2023. Date of publication 25 January 2023; date of current version 6 July 2023. This work is supported by National Natural Science Foundation of China (Grant No. 61732002 and No. 62072018). The associate editor coordinating the review of this article and approving it for publication was N. Zincir-Heywood. *(Corresponding author: Zhongzhi Luan.)*

Jiaxing Qi, Zhongzhi Luan, Shaohan Huang, Hailong Yang, Hanlu Li, Danfeng Zhu, and Depei Qian are with the School of Computer Science and Engineering, Sino-German Joint Software Institute, Beihang University, Beijing 100191, China (e-mail: jiaxingqi@buaa.edu.cn; luan.zhongzhi@buaa.edu.cn).

Carol Fung is with the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC H3G 1M8, Canada.

Digital Object Identifier 10.1109/TNSM.2023.3239522

System logs record the elaborate status of the system. We can timely and effectively discover abnormalities in systems by analyzing logs. However, systems typically generate massive logs due to their complex services. It is a big challenge to detect the potential abnormality through log analysis. Over the years, many methods have been proposed to automatically detect system log anomalies [2], [3], [4], [5]. For example, [6] evaluated six (Machine learning) ML-based anomaly detection methods on system logs. These ML-based methods share some limitations, requiring more careful feature engineering, inefficiency, and poor generalization. To address these problems, deep learning (DL-based) anomaly detection methods have been proposed [3], [7], [8], [9]. Generally, the existing DL-based log anomaly detection methods are superior to ML-based methods. However, several important inherent challenges remain in log-based anomaly detection. We summarize the major challenges as follows:

- *Discrete property of logs:* Unlike continuous sequence data (such as sensor data), which have physical meanings, log data is discrete log event sequences. As such, the relations among events are hard to capture.
- *Imbalanced classes:* This is a common challenge in the field of anomaly detection. As systems are running under normal status most of the time, abnormal sequences in the raw logs after preprocessing are infrequent [8], leading to the situation that the size of normal logs is much larger than abnormal logs. To evaluate the effect of imbalance ratio (IR) on performance, where the IR is defined in (1), we randomly selected 6000 normal logs in hadoop distributed file system (HDFS) and applied a simple two-layer long short-term memory (LSTM) model to test the performance. The results are shown in Fig. 1, which demonstrates that the performance degrades dramatically when IR is greater than 5. However, the IR in regular log data sets is typically greater than 5, thus traditional classification methods may not be suitable for log-based anomaly detection.

$$\mathrm{IR} = \frac{n_{\mathrm{normal}}}{n_{\mathrm{abnormal}}} = \frac{|\mathcal{N}|}{|\mathcal{P}|} \tag{1}$$

- *Unstable Log:* Since the log is defined by system developers, the source code may be modified on-the-fly because the developers add some new log events. For example, we give some real cases of evolving log events from HDFS logs, as shown in Fig. 2. In case 1, two new keywords ("of size") are appended to the original log event, and one

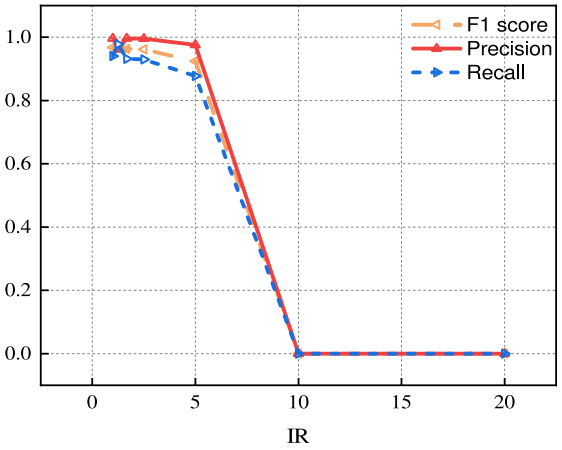

Fig. 1. Different imbalance ratio impact on HDFS example.

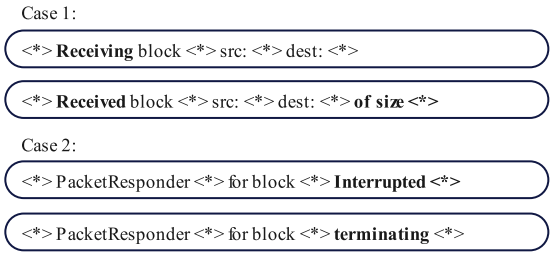

Fig. 2. Examples of Evolving Log Events.

keyword ("Receiving") is replaced with a new keyword ("Received") to record the detail of the original log. In case 2, developers replaced the keyword ("Interrupted") with a new keyword ("terminating") to record the new system event. Therefore, log-based anomaly detection must be able to cope with log unstability effectively.

- *Noise in log data:* While collecting and processing log data, some noises may be added. For example, when a cloud system uploads data to data centers, data loss, duplication, and disorder may happen due to network problems. Moreover, the log parsing method also yields noises. The noises mentioned above have a great impact on the performance of anomaly detection [7].

Based on the techniques used, log-based anomaly detection methods can be divided into supervised methods, unsupervised methods, and semi-supervised methods. Although these methods are effective in their corresponding scenario, they still suffer from the following problems:

- Supervised methods require annotation information of logs (e.g., whether an anomaly has occurred), which is a time consuming process. Moreover, it is affected by the imbalanced-class problem mentioned previously. As a result, most research focused on semi-supervised and unsupervised methods.
- Unsupervised and semi-supervised methods do not require annotation information of logs, which avoids being affected by class imbalance. However, the detection accuracy is often compromised due to the lack of prior information. Furthermore, the performance degrades significantly when new log events appear in the testing logs [7]. Therefore, these methods lack robustness.

TABLE I
COMPARISON WITH PRIOR WORK

Challenges	Discrete	Imbalanced	Unstable	Noise
DeepLog [14]	✗	✓	✗	✗
LogAnomaly [12]	✓	✓	✓	✗
LogBert [11]	✓	✓	✗	✗
LogCNN [15]	✓	✗	✗	✗
LogRobust [7]	✓	✗	✓	✓
Ours	✓	✓	✓	✓

We summarized the comparison of our work to existing DL-based methods in Table I. Except for DeepLog, other prior studies are solving the challenge of discrete property of logs by leveraging word2vec [10] or event embedding technology [11]. However, the event embedding can not capture the semantic information and suffers from the unstable log challenge. Others applied the word2vec and leveraged the weighted average of word vectors of the words in the events, which may be lost useful semantic information, resulting in poor detection accuracy [12]. In addition, the evolution of logging statements and noise in logs also significantly affect model performance [13]. Existing methods can not overcome these challenges besides LogRobust, which is suffered from the class imbalanced problem.

To address the above challenges, we propose LogEncoder, a log representation framework with semi-supervised anomaly detection. The framework consists of three parts: Log to embedding (*Log2Emb*), Embedding to representation (*Emb2Rep*), and Anomaly detection. For *Log2Emb*, we first use a pre-trained model to parse a natural language-based log into a fixed dimension. After that, we transform discrete log events into continuous semantic vectors, which can preserve the relations among events. For *Emb2Rep*, we build a deep learning model to learn the representation of log event sequences, which combines the anomaly detection objective and the sequence pattern to represent the log event sequence. Specifically, we employ a one-class objective [16] such that all normal log event sequences are distributed on a hyper-sphere, and the abnormal log event sequences are distributed away from the hyper-sphere in latent space.

Furthermore, our approach is to learn sequences pattern (also called context information) based on contrastive objective [17], which is a self-supervised learning method. Finally, we use an attention-based model to capture the global and local information about log event sequences. For the anomaly detection part, we propose two types of methods: offline-based and online-based. Evaluation results demonstrate that the offline-based and online-based methods obtained competitive results and can overcome the unstableness problem. The main contributions of this work are summarized as follows:

- We propose a framework called LogEncoder for log-based anomaly detection, which contains only normal logs in the training data. Therefore, it is not influenced by the class imbalance problem in supervised learning. We also employ a pre-trained model to extract continuous semantic vectors for discrete log events, which

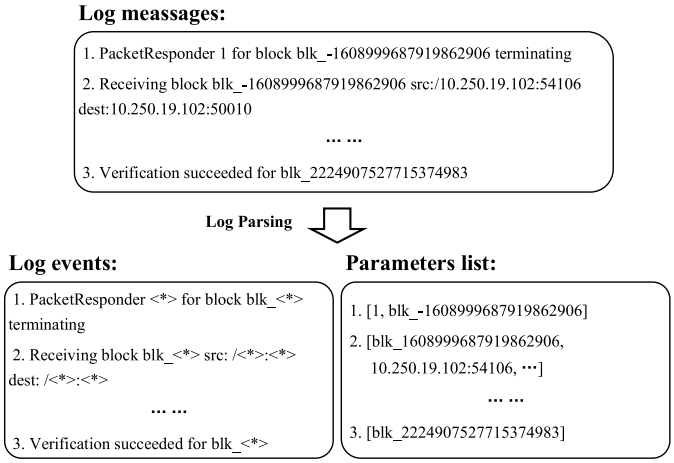

Fig. 3. The example of log parsing.

overcomes the challenge that discrete events do not capture correlations among events.

- To the best of our knowledge, we are the first to propose adopting one-class classification and using contrastive learning as the objective of the model in log analysis, which allows the model to learn separable features while preserving the contextual information of the sequences.
- We conducted an extensive performance evaluation to demonstrate that LogEnocoder outperforms five state-of-the-art unsupervised and semi-supervised learning methods. The the results also demonstrate the competitiveness of LogEnocoder compared to the supervised learning model LogRobust.

The rest of this paper is organized as follows. In Section II, we introduce the preliminary for LogEncoder. In Section III, we describe the details of LogEncoder. In Section IV, we present the experimental results and analyses. In Section V, we briefly review the related work in log-based anomaly detection. At last, we conclude our work in Section VI.

II. Preliminaries

In this section, we go over some background techniques that are used by LogEncoder, including log parsing, one-class classifier, and contrastive learning.

A. Log Parsing

As shown n Fig. 3, each raw log message consists of a constant part (also called "log event" or "log key") and a variable part (also called "parameter"). Log events and parameters can be extracted by log parsing methods, such as Spell [18], IPLoM [19], Drain [20], and MoLFI [21]. According to the previous study [22], weak log parsing methods can negatively affect the performance of the following anomaly detection task. As a result, Drain has become a popular method used by researchers due to its superior accuracy and efficiency [22], [23]. In this work, we use Drain as our log parsing method.

B. One-Class Classifier

One class method [16] is commonly used in the field of anomaly detection, such as **O**ne **C**lass **S**upport **V**ector **M**achines (OCSVM) [24] and **S**upport **V**ector **D**ata **D**escription (SVDD) [25]. They typically use kernel tricks to map normal data to a hypersphere in the latent space. However, these kernel-based methods are not suitable for high-dimensional data and are inefficient. In recent years, one class methods have been extended to deep learning. Existing methods include DeepSVDD [26] and **O**ne **C**lass **N**euron **N**etwork (OCNN) [27] which use neural networks instead of kernel tricks to find the hyper-sphere. Here we use OCSVM [16] as an example to illustrate a one class classifier. The optimization objective can be written as follows:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi},b} \ & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu N}\sum_i \xi_i - b \\ \text{s.t.} \ & \langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle \geq b - \xi_i, \xi_i \geq 0 \end{aligned} \tag{2}$$

where ξ_i is the slack variable corresponding to the i^{th} training sample; Φ is a mapping function that maps x_i to a latent space; b is the bias term; ν is a trade-off parameter, and N is number of training examples. After solving this optimization, we can use the condition $sgn(\langle \mathrm{w}, \phi(\mathrm{x})\rangle - b)$ for $\mathrm{x} \in \mathrm{X}_{test}$.

C. Contrastive Learning

Contrastive learning is a self-supervised learning method in deep learning that does not require annotation information to learn the representation of data [17]. According to previous research [28], it can significantly improve downstream tasks, including anomaly detection. Specifically, given the data x, contrastive learning learns an encoder f that maximizes the following objective:

$$\text{score}\big(f(x), f\big(x^+\big)\big) >> \text{score}\big(f(x), f\big(x^-\big)\big) \tag{3}$$

where x^+ is a positive sample that is similar to x, x^- is a negative sample that is dissimilar to x, and score is a metric function to measure the similarity between samples. For the definition of the metric function, there have been many studies and more surveys can be found in [17], [28], [29].

III. Framework

In this section, we introduce LogEncoder, a semi-supervised log representation framework for anomaly detection. An overview of the proposed framework of **LogEncoder** is shown in Fig. 4. It consists of three major components, *Log2Emb*, *Emb2Rep*, and *Anomaly detection*. The details of each component are elaborated in this section.

A. Overview

As mentioned in Section I, logs are often modified by developers, and the log anomaly detection model needs to be able to deal with this challenge. In this work, we use a pre-trained model, namely **B**idirectional **E**ncoder **R**epresentation from **T**ransformers (BERT) [30], to extract the semantic information of log events and transform them into semantic vectors. This process is called *Log2Emb*. The second component is *Emb2Rep*. In this process, we extract features that separate abnormal and normal log sequences, which is the core of LogEncoder. To do so, we built an attention-based model which combines one-class objective and contrastive objective

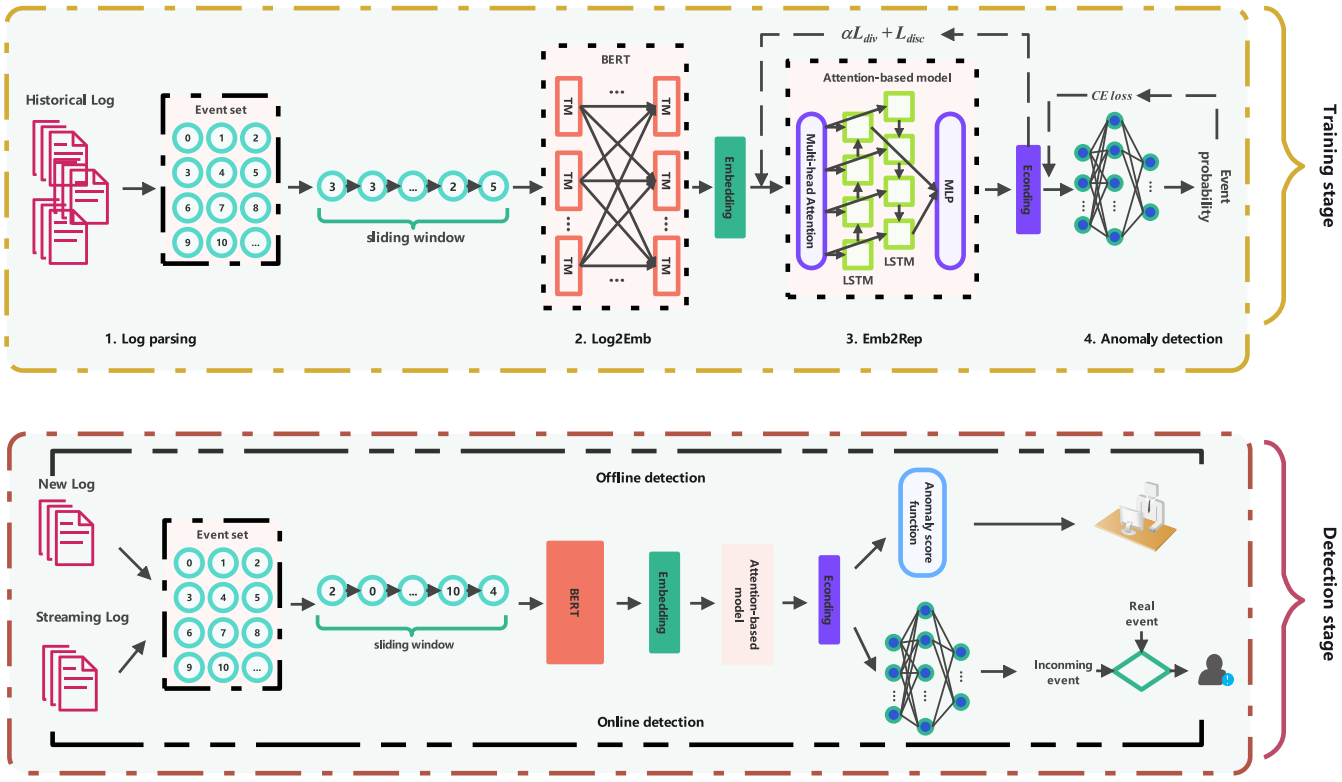

Fig. 4. Overview of LogEncoder. The directed solid black line indicates the data flow and the directed black dashed line indicates the gradient delivery.

to extract features. It mainly addresses how to extract separable features with contextual information, which means that normal and abnormal sequences have a boundary in feature space. Finally, the anomaly detection stage is decoupled from the extracted feature stage. Hence, the anomaly detection component is flexible and diverse. We describe each component in detail in the following subsections.

B. Log2Emb

Unlike existing methods that use one-hot encoding to represent log events [14], we use a pre-trained model that extracts a semantic vector of fixed dimensions for each log event. The model converts discrete sequences into continuous sequences and mitigates the effects of unstable logs. As mentioned in Section II, the raw logs need to be transformed into events and parameters through log parsing methods. In this paper, we choose the Drain [20] method to transform logs to log events and parameters. Next, we extract the semantic vectors of each log event using the pre-trained BERT [30]. Existing methods use the word embedding technique (e.g., Word2Vec [10] and FastText [31]) to convert keywords in log events into vectors, and the weighted sum (e.g., TF-IDF [32]) of all vectors in log events to represent log events. However, using the weighted sum of keyword vectors to represent log events may be lose useful semantic information.

The BERT[1] pre-trained model has a good semantic representation because of its rich prior information. The architecture of the BERT model is shown in Fig. 5. We consider log

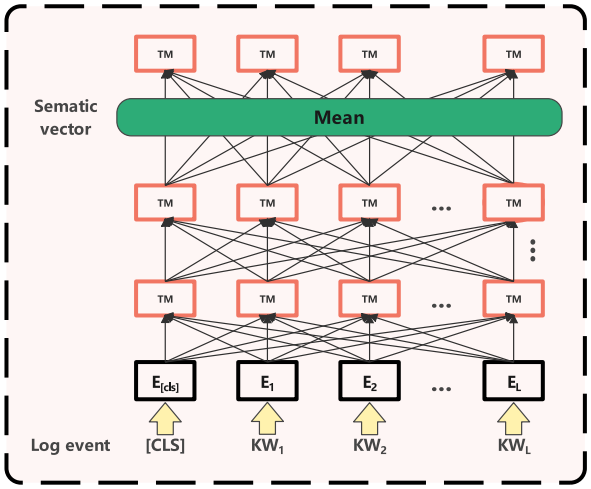

Fig. 5. The architecture of the BERT model.

events as sentences in natural language to extract semantic vectors of log events. Given a normal event sequence, i.e., $S^{(i)} = (e_1^{(i)}, e_2^{(i)}, \ldots, e_l^{(i)}, \ldots, e_L^i)$, where L is the sequence length. Firstly, the keywords (KW_i in Fig. 5) in the log events $e_l^{(i)}$ are obtained by using the tokenizer. Secondly, it is used as input to transformer encoders (TM in Fig. 5) to obtain a d dimensional semantic vector $z_l^{(i)}$, where d is set to 768 in BERT. Note that the output $z_l^{(i)}$ is the last hidden state. Finally, we obtained a semantic vector sequence after *Log2Emb*, written as $Z^{(i)} = (z_1^{(i)}, z_2^{(i)}, \ldots, z_l^{(i)}, \ldots, z_L^{(i)})$.

In summary, *Log2Emb* represents log events with semantic vectors, preserving the semantic information of events so

[1]As we only use the feature extraction part of BERT, the rest of BERT will not be shown in this paper.

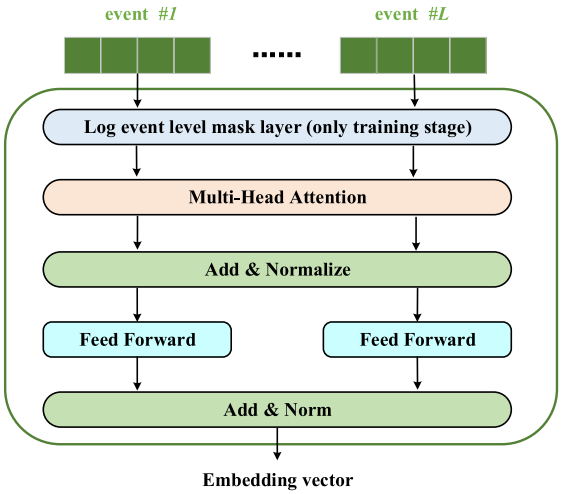

Fig. 6. The architecture of multi-head attention block.

that similar log events also have similar semantic vectors, mitigating the impact of unstable logs and noises.

C. Emb2Rep

After *Log2Emb*, we present each event as a semantic vector. And then splitting the log events into different sequences with a sliding window or session window (same *'blockID'* in HDFS). In order to detect abnormal sequences, it is important to learn the feature representation of the whole sequence. To this end, we propose an attention-based model, which includes a Multi-head Attention block, RNN block, and MLP block, to learn the whole sequence representation, and the details are described as follows.

Given a semantic vector sequence after *Log2Emb*, $Z^{(i)} = (z_1^{(i)}, z_2^{(i)}, \ldots, z_l^{(i)}, \ldots, z_L^{(i)})$, the model learns a representation of the sequence in an attention mechanism. Note that we only keep the last step hidden state $x^{(i)}$ as output, written as:

$$x^{(i)} = \sigma(wh_L + b) \tag{4}$$

where w, b is the trainable parameters of the input h_L and $\sigma(\cdot)$ is a LeakyRelu function [33].

Moreover, $h_L^{(i)}$ in (4) is the last hidden state of R, which is computed as follows:

$$h_L^{(i)} = R_L\left(a^{(i)}, \theta_{rnn}\right) \tag{5}$$

where θ_{rnn} is the trainable parameters. R can be any Recurrent Neural Network (RNN), such as Gated Recurrent Neural Networks (GRU) or Long Short-Term Memory (LSTM). According to previous studies [14], [34], we use LSTM in our method since it can effectively learn the location information of the input sequence and use the last step hidden state of the output as representative of the whole sequence.

$a^{(i)}$ in (5) is the multi-head attention block output. This block captures the global and local information of the input sequence and takes into account the computational efficiency, The multi-head attention block architecture is shown in Fig. 6.

1) Learning Separated Features by One-Class Objective: Inspired by one-class classification [35], we would like all input sequences to be distributed within a hyper-sphere of the latent space with center $\mathbf{c}$ and with a smaller radius. To do this, we propose the following objective function:

$$\mathcal{L}_{disc} = \min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \left\| x^{(i)} - \mathbf{c} \right\|^2 + \lambda \|\Theta\|_F^2 \tag{6}$$

where $\mathbf{c}$ is the pre-defined center of all input sequences, and Θ is the trainable parameters of the model. The first term in the objective makes all input sequences close to the center. The second term is used as regulation controlled by hyper-parameter λ. Therefore, $\mathcal{L}_{disc}$ forces the model to map the input sequences to the hyper-sphere of the latent space with center $\mathbf{c}$.

2) Learning Contextual and Diversity Features by Contrastive Objective: Based on the one-class objective, the model learns features with discriminative for anomaly detection tasks. However, it ignores the critical contextual information of the log event sequences. Moreover, the one-class objective may lead to different normal log event sequences being too close together or overlapping in the latent space, resulting in a lack of diversity in features so that it cannot provide useful information for anomaly detection tasks. To address these limitations, we introduce an objective based on contrastive learning, which is defined as follows:

$$\mathcal{L}_{div} = \min_{\Theta} -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\mathcal{D}\left(x^{(i)}, x_{mask}^{(i)}\right)}{\sum_{j=1}^{N} \mathcal{D}\left(x^{(j)}, x_{mask}^{(i)}\right)} \tag{7}$$

where $\mathcal{D}(x, y)$ is the similarity function of x and y. $x_{mask}^{(i)}$ is the output of model which input $S^{(i)}$ is masked through the mask layer. In the mask layer, we random set $e_l^{(i)}$ to zero for the input $S^{(i)}$, and the ratio of the mask is a hyper-parameter. More detailed analysis can be found in Section IV-E.

The numerator of $\mathcal{L}_{div}$ forces $x^{(i)}$ and $x_{mask}^{(i)}$ to be similar, and the denominator makes dissimilar with $x^{(j)}$ and $x_{mask}^{(i)}$, thus it can learn instance-wise discriminativeness features, preserve the diversity of features. Based on probabilistic perspective, given $x_{mask}^{(i)}$, we have the following probabilities:

$$\begin{aligned} p\left(\kappa = i \mid X, x_{mask}^{(i)}\right) &= \frac{p\left(x^{(i)} \mid x_{mask}^{(i)}\right) \Pi_{l \neq i} p\left(x^{(l)}\right)}{\sum_{j=1}^{M} p\left(x^{(j)} \mid x_{mask}^{(i)}\right) \Pi_{l \neq j} p\left(x^{(l)}\right)} \\ &= \frac{\frac{p\left(x^{(i)} | x_{mask}^{(i)}\right)}{p\left(x^{(i)}\right)}}{\sum_{j=1}^{M} \frac{p\left(x^{(j)} | x_{mask}^{(i)}\right)}{p\left(x^{(j)}\right)}} \end{aligned} \tag{8}$$

where $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(M)}\}$ means M log sequences; $\kappa = i$ is the indicator that $x^{(i)}$ corresponding to $x_{mask}^{(i)}$. Compared to (7), it is obvious that $\mathcal{D}(x^{(i)}, x_{mask}^{(i)}) \propto \frac{p(x^{(i)}|x_{mask}^{(i)})}{p(x^{(i)})}$. Followed by [28], we can proof that minimizing (7) is equivalent to maximizing the mutual information of $x^{(i)}$ and $x_{mask}^{(i)}$.

Algorithm 1 Mini-Batch Stochastic Gradient Descent Training of LogEncoder

Input:
The training normal event sequences: $\{S^{(i)}\}_{i=1}^{M}$;
The model of parameters $\Theta^{(f)}$, hyper-parameters α, the iterative number t, and the batch size M;
Output: The model of parameters $\Theta^{(f)}$;

for 1 to t **do**
  for $i = 1$ to M **do**
    Get the semantic vector sequence of $S^{(i)}$ via *Log2Emb* block:

$$Z^{(i)} = (z_1^{(i)}, z_2^{(i)}, \ldots, z_l^{(i)}, \ldots, z_L^{(i)})$$

    Input $Z^{(i)}$ to the model f, and the output is $x^{(i)}$;
    Input $Z^{(i)}$ to mask layer, and the output is $Z_{mask}^{(i)}$
    Input $Z_{mask}^{(i)}$ to the model f, and the output is $x_{mask}^{(i)}$;
  end for
  Update the model f by descending its stochastic gradient:

$$\nabla_{\theta^{(f)}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_{disc}(x^{(i)}) + \alpha \mathcal{L}_{div}(x^{(i)}, x_{mask}^{(i)})$$

end for
return $\Theta^{(f)}$;

3) Integrate One-Class and Contrastive Learning Objectives: As aforementioned, we defined (6) and (7) based on the log anomaly detection task, which ensures the discriminative, diversity,and contextual information of the learned features. Specifically, given $\mathcal{L}_{disc}$ and $\mathcal{L}_{div}$, the overall objective function of LogEncoder is defined as follows:

$$\min_{\Theta} \mathcal{L} = \mathcal{L}_{disc} + \alpha \mathcal{L}_{div} \tag{9}$$

where α is the hyperparameter that controls the discriminative and diversity of features, the specific optimization procedure is described in Algorithm 1.

D. Anomaly Detection

The features learned in the previous stage keep the contextual information, and the normal sequences are distributed in the hyper-sphere in latent space, which is decoupled from the anomaly detection stage and can be combined with various detection methods. In this subsection, we propose two approaches for log anomaly detection: **Offline detection** and **Online detection**. Offline detection is the detection of anomalies for log event sequences that have been generated. The distance between the representation of the log event sequence and **c** is used as the anomaly score to evaluate whether it is an anomaly sequence or not. Online detection predicts upcoming events based on historical log event sequences and compares them with the occurrence events. More details of the two approaches are described as follows.

- *Offline detection:* After previous subsection, we have center **c** of the normal sequences in training data. For sequences $S^{(i)}$ in test data, we can compute anomaly score as follows:

$$A\left(x^{(i)}\right) = \|x^{(i)} - \mathbf{c}\|^2 \tag{10}$$

  After that, we evaluate whether $x^{(i)}$ is an abnormal sequence as follows:

$$label = \begin{cases} Normal & \text{if } A\left(x^{(i)}\right) < \varepsilon \\ Abnormal & \text{else} \end{cases} \tag{11}$$

  where ε is a hyperparameter indicating the threshold of the anomaly score.
- *Online detection:* As mentioned in the previous subsection, *Log2Emb* not only improves the accuracy of LogEncoder by leveraging the semantic information of log events but also overcomes the challenges of the unstable log by matching new events to existing events (See Table V). In addition, LogEncoder extracted features to preserve contextual information by leveraging contrastive learning. Therefore, we can use it to predict possible upcoming log events. To do this, we build a predictive model $P(x)$. Specifically, Given a new sequence feature $x^{(i)}$, the output of $P(x)$ is defined as follows:

$$p^{(i)} = P\left(x^{(i)}, \omega\right) \tag{12}$$

  where ω is the trainable parameters and $p^{(i)}$ is a K dimensional vector and $p_k^{(i)}$ indicates the probability of the k-th event happens after the log event sequence $S^{(i)}$. when the service system is running, our model predicts the next event occurrence probability by historical logs. If the actual event does not include in the results, we consider the system occur anomalies and touch alarms.

IV. Evaluation

In this section, we compare LogEncoder performance with six baseline methods, including five unsupervised and semi-supervised methods and one supervised method. For the five unsupervised and semi-supervised methods, we compare them to evaluate LogEncoder's effectiveness. For the one supervised method, we compare the gaps between LogEncoder and evaluate the competitiveness of our approach.

A. Datasets

- *HDFS [36]:* This dataset was generated by map-reduce jobs on the Amazon cloud environment. It contains 11,175,629 log records; about 2.9% of them are manually labeled anomalies. Each log record includes a unique block ID, containing 575061 blocks in total, of which 16838 blocks are labeled anomalies.
- *BGL [23]:* It is a public log dataset generated by BlueGene/L supercomputer system at Lawrence Livermore National Labs, which contains 4,747,963 log messages, and 348,469 log messages were labeled as anomalies. Unlike HDFS data, BGL logs have no identifier recorded for each job execution. Thus, we use time

TABLE II
STATISTICS OF THREE PUBLIC DATASETS

Datasets	Duration	# of logs	# of anomalies	# of events
HDFS	38.7hours	1,175,629	16,838(blocks)	47
BGL	7months	4,747,963	348,469(logs)	377
Thunderbird	244days	2,000,000	6,590(logs)	1758

TABLE III
CONFUSION MATRIX

Confusion Matrix		Prediction Labels	
		Positive	Negative
True Labels	Positive	TP(True Positive)	FN(False Negative)
	Negative	FP(False Positive)	TN(True Negative)

sliding windows of thirty minutes in width to generate log sequences.

- *Thunderbird [37]:* Thunderbird is a public log dataset generated by the Thunderbird supercomputer system at Sandia National Labs (SNL) in Albuquerque, with 9,024 processors and 27,072GB of memory. We selected the first 2,000,000 log messages as our dataset, among which 13,150 logs were labeled anomalies. Like BGL, we also adopt a one-minute sliding window to generate log sequences.

We show the statistics of the three log datasets in Table II. This work is a semi-supervised method for representing log event sequences, where the training data only contains normal log event sequences. We use *chronological* strategy [13] to split the training and testing set. Specifically, the data is split by the timestamp of logs. We select the first 50% of the normal logs as the training set, 10% of the normal logs as the validation set to select the threshold score, 40% of the normal logs and all the abnormal logs as the testing set.

B. Baselines

We compare the results of LogEncoder and state-of-the-art methods, including supervised, semi-supervised and unsupervised. The details of these methods are as follows:

- *Principle Component Analysis (PCA) [2]:* It is a commonly used unsupervised dimension reduction technique in machine learning. In recent years, it has been used for anomaly detection. Firstly, it constructs a counting matrix *CM*, where $CM_{i,j}$ indicates the count of j^{th} log event in the i^{th} sequence. Then, the matrix *CM* is reduced to a low dimensional space using PCA to detect upcoming sequences.
- *Invariant Mining (IM) [38]:* Like PCA, IM is an unsupervised method based on the counting matrix. It learns the invariants of the sequence. If the upcoming sequence does not satisfy the learned invariants' hypothesis, it is considered an anomaly.
- *DeepSVDD [26]:* It is a semi-supervised anomaly detection method that uses neural networks to learn features that map normal sequences to the hyper-sphere and abnormal sequences away from the hyper-sphere.
- *DeepLog [14]:* It is a customized semi-supervised method for log-based anomaly detection. This method uses an LSTM model to learn the patterns of normal sequences. In the detection stage, the next log events are predicted based on the recent sequences. If the actual events are predicted, then consider them normal.
- *LogAnomaly [12]:* It is a semi-supervised framework for log-based anomaly detection that detects log sequence anomalies from both sequence pattern and quantitative pattern perspectives.
- *LogRobust [7]:* It is a supervised method for log-based anomaly detection. Specifically, LogRobust represents the log events as semantic vectors and then uses an attention-based model to learn the contextual information in the log sequences. Note that LogRobust is effective for unstable logs.
- *LogBert [11]:* It is a semi-supervised method for log anomaly detection based on BERT. LogBert applies two training tasks, Masked Log Key Prediction (MLKP) and Volume of Hyper-sphere Minimization (VHM), to capture the patterns of normal log sequences.

C. Experimental Settings

Model Selection: For all methods, we report the results from the optimal parameters. The implementation of PCA and IM can be found at github.[2] For DeepLog, LogAnomaly, and LogRobust, the implementation can also be found at github.[3] The implementation of DeepSVDD was taken from the public source code.[4] For LogEncoder, we have selected different numbers of attention blocks and rnn blocks from 1 to 6. In addition, we select the objective weight α from {0.01, 0.1, 0.5, 1.0, 1.5, 2.0}. For the hyper-sphere of center $\mathbf{c}$, we use an initial set of model parameters to obtain all training data representation vectors. Then, we use the average value of all vectors as $\mathbf{c}$. Moreover, we set the fixed window size to 20 on HDFS, a 30-minute time window on BGL, and a 1-minute time window on Thunderbird. Note that the sliding window is overlapping. Finally, we have also selected latent size and mask ratio. For more details, which can be seen in Section IV-E.

Implementation Details: We implement LogEncoder based on Pytorch 1.7.1 on the Linux server with 8× NVIDIA GTX1080Ti GPUs. The model is optimized by Adam [39] with begin learning rate is 0.0001.

Evaluation Metrics: Anomaly detection can be seen as an imbalanced binary classification task. Therefore, we use F1 score, Precision, and Recall [14] to measure the performance of the log-based anomaly detection method. When an anomaly is detected, we label it as positive. The Precision, Recall, and F1 score are defined in (13), (14) and (15), which based on the confusion matrix, as shown in Table III.

[2]https://github.com/logpai/loglizer
[3]https://github.com/donglee-afar/logdeep
[4]https://github.com/GRSEB9S/deepSVDD/

TABLE IV
THE PERFORMANCE COMPARISON ON THREE DATASETS

Methods	HDFS			BGL			Thunderbird			Average Rank
	F1 score	Pricision	Recall	F1 score	Pricision	Recall	F1 score	Pricision	Recall	
PCA (Unsupervised)	0.646	0.662	0.632	0.401	0.313	0.899	0.581	0.474	0.955	6.333
IM (Unsupervised)	0.765	0.701	0.899	0.413	0.247	**0.956**	0.384	0.276	0.935	5.667
DeepSVDD (Semi-supervised)	0.732	0.711	0.728	0.534	0.356	0.673	0.491	0.377	0.837	6.556
DeepLog (Semi-supervised)	0.761	0.840	0.695	0.384	0.237	0.853	0.219	0.118	0.743	7.556
LogAnomaly (Semi-supervised)	0.835	0.833	0.849	0.462	0.289	0.851	0.614	0.506	0.872	5.111
LogBert (Semi-supervised)	0.758	0.803	0.718	0.895	0.867	0.925	0.802	0.877	0.794	4.556
LogEncoder (Offline)	0.821	0.817	0.836	0.723	0.687	0.824	**0.889**	**0.875**	**0.992**	3.889
LogEncoder (Online)	**0.944**	**0.945**	**0.943**	**0.896**	**0.935**	0.876	0.483	0.201	0.947	3.222
LogRobust (Supervised)	0.923	0.919	0.925	0.975	0.921	0.999	0.896	0.872	0.922	2.111

* Best result are highlighted. The larger is better. Average rank (the smaller is better) is record in the last column. Note that we report the top@1 result for the LogEncoder (Online).

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

$$F1 = \frac{2 * Precison * Recall}{Precison + Recall} \tag{15}$$

To evaluate the impact of different threshold selections on performance, we use AUC to measure the performance with different parameter selections. It takes values to range from 0 to 1. A value close to 1 indicates that the model performance is better, and lower than 0.5 indicates that the model results are not useful. Finally, we use **average rank** [40] to evaluate the overall performance of each method on different datasets.

D. Performance Comparison

In this subsection, we evaluate the performance of the offline-based and the online-based detection method on three public log data and compare them with five unsupervised or semi-supervised methods and one supervised method. The comparison results are shown in Table IV.

1) Comparison With Unsupervised and Semi-Supervised Methods: We can see from Table IV that the traditional data mining-based methods (PCA and IM) do not perform well on most datasets. We believe this is because the traditional methods learn normal patterns based on the count matrix. An interesting discovery is that IM is competitive compared to DeepLog and DeepSVDD on some datasets. For example, the f1 score of IM (0.765) outperforms DeepLog (0.761) and DeepSVDD (0.732) on HDFS. Deep learning based methods, such as DeepLog, DeepSVDD, and LogAnomaly, perform moderately well on HDFS. However, the performance of BGL and Thunderbird is much worse. We speculate that the reason is that the number of log events of BGL (377) and Thunderbird (1758) is larger than HDFS (47). Also, many log events are rare or even non-existent in the training data of both datasets, which makes them unstable. LogEncoder(Offline) obtained state-of-the-art results on Thunderbird. The latest state-of-the-art method, LogBert, has bad results on HDFS (f1 score is 0.758). Moreover, it has a good performance on BGL and Thunderbird. Especially, it has state-of-the-art results on BGL. In addition, the f1 score of LogEncoder (Offline) is only 0.014 lower than LogAnomaly on HDFS. This demonstrates that our method can map normal log sequences to a hyper-sphere and handle unstable logs.

In real-life scenarios, log-based anomaly detection methods are expected to detect abnormal events in real time. Therefore, we evaluate the proposed online based detection method because the features extracted by LogEncoder include the contextual information of the log event sequence. Our proposed online detection method outperforms all other methods on HDFS and BGL. However, the performance is much worse on Thunderbird. We can see that the recall of our method is high, but the precision is low, which indicates that a large number of normal events are predicted as abnormal events. We analyzed the differences between the three datasets, which mainly differ in the number of log events. HDFS, BGL, and Thunderbird contain the number of events of 47, 377, and 1758, respectively. We sorted them by instability, and the result is HDFS $<$ BGL $<$ Thunderbird.

The online one achieves higher recall and lower precision on Thunderbird, indicating it is hard to predict following events accurately. Because the testing set of Thunderbird contains more log events that do not appear in the training set, the instability is the highest. Furthermore, it also indicates that the performance of the online method decreases as the instability of the dataset ascends. The offline approach performs better on datasets with high instability. Otherwise, the online method should be preferred for better detection performance.

Compared to DeepLog and LogAnomaly, because the other methods are not suitable for online scenarios, LogEncoder (Online) is all better than DeepLog, with a lower f1 score than LogAnomaly on Thunderbird. Because the Thunderbird has the highest instability among the three datasets, more log events are unseen in the model. In order to validate that LogEncoder is suitable for datasets with high instability, we provide the top@k result in Fig. 7. We notice that the results of top@10 are quite better than top@1, which demonstrates that the online method can improve detection performance by increasing the number of candidate events on Thunderbird. Further demonstrates that the features extracted by LogEncoder include important contextual information.

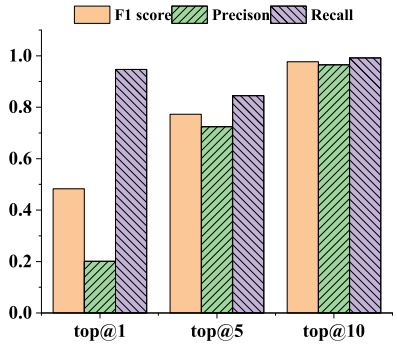

Fig. 7. LogEncoder online detection for top@k log event candidate on Thunderbird.

2) Comparison With the State-of-the-Art Supervised Methods: LogRobust achieves state-of-the-art performance on BGL and Thunderbird datasets because of its ability to handle unstable logs efficiently. LogEncoder achieves comparable results with supervised-based LogRobust on both unstable datasets. It means that our proposed approach could handle the unstable challenge. Moreover, the LogEncoder (Online) f1 score is 0.021 higher than LogRobust on the stable HDFS dataset. This demonstrates that LogEncoder is competitive compared to the supervised method. And our method reduces the performance gap with the supervised method LogRobust on BGL and Thunderbird.

In summary, the experimental results demonstrate that LogEncoder has a better performance on three log datasets compared to five unsupervised and semi-supervised methods. It is due to the LogEncoder learned features for anomaly detection by one-class objective and captures the contextual information of the log event sequence by diversity objective. Compared to the supervised method LogRobust (average rating of 2.111), LogEncoder shows competitive performance in both offline-based and online-based detection. The average ratings of them are 3.889 and 3.222, respectively.

E. Parameter Analysis and Ablation Studies

In this subsection, we analyze some key hyper-parameters and the components of LogEncoder. Fig. 8 shows the impact of AUC with different configurations on HDFS. Then, we demonstrate the necessity of attention block and rnn block design.

1) Impact of the Weight of the Contrastive Loss and Window Size: We first analyze the effect of the objective weight α on AUC, and the results are shown in Fig. 8a. We find that the best AUC is achieved when the α is 0.1, and the only slight change in AUC is observed when the α is further increased. LogEncoder will degrade to be similar to DeepSVDD when α is equal to 0.00. Fig. 8b shows the performance of different sliding window sizes. The offline method is stable, and the online method losses performance as the sequence length increases.

2) Impact of the Number of Attention Blocks: As shown in Fig. 8c, we evaluate the number of attention blocks from 1 to 6 and can see that the impact on AUC for both online-based and offline-based detection methods is low when the number is greater than 1. We measure this because the model complexity is insufficient to learn the features of the log event sequence when the number of attention blocks is 1. Moreover, attention blocks learn global and local information about log event sequences. Thus the impact on AUC is minimal for online-based detection.

3) Impact of the Number of rnn Blocks: Fig. 8d shows the trend of AUC for different numbers of rnn blocks. We can see that the offline-based detection method is stable with the different number of rnn blocks. This only affects the performance of the online-based detection method since the rnn block is used to learn the contextual information of the log event sequence. We also notice that the AUC does not change significantly when its number is greater than 2. Considering the efficiency of online-based detection, we set the number of rnn blocks to 2. We also investigate the impact of different latent sizes, and the results are shown in Fig. 8e. We can see that the offline-based detection method is stable for different latent sizes, which indicates that only a low latent space is required to separate normal and abnormal log event sequences. For the online-based method, the latent size to achieve the best AUC is 128, which indicates that the latent size has an impact on the contextual information representation of the log event sequences.

4) Impact of Log Event Level Mask Ratio: We analyze the effect of the mask ratio. As shown in Fig. 8f, the mask ratio only affects the performance of the online-based detection method, while the offline-based detection method is barely influenced. The online-based detection method achieves the optimal AUC when the mask ratio is 0.3, and after that, the AUC decreases. It is because when the mask ratio is too high, the input information of the model is limited, and it restricts the model from learning the contextual information of the log event sequence effectively. In summary, LogEncoder is fairly stable for different parameters and component configurations.

5) How the rnn Block and Attention Block Contribute to Anomaly Detection: To evaluate the benefit of rnn blocks and attention blocks, we analyze the combination of different numbers of rnn blocks (#RNN blocks) and attention blocks (#Attention blocks), and the results are shown in Fig. 9a and Fig. 9b. For the offline-based method, we can see that the F1 score gradually increases when the number of attention blocks is less than 3. Furthermore, when the number of attention blocks is fixed, the impact of different numbers of rnn blocks on F1 is slight. For example, the three curves corresponding to the number of rnn blocks equal to 3, 4, and 5 almost overlap. The reason is that the offline-based method is mainly based on the global and local information captured by the attention block. For the online-based method, we can see that different numbers of attention blocks slightly impact performance. Moreover, the F1 score is lower if the number of rnn blocks is 0 (black line), and a better F1 score is obtained if the number of rnn blocks is 5 (dark yellow line). This demonstrates that the online-based method relies on contextual information captured by the rnn block. In addition, we investigated how the results would be different if only to keep the MLP layer. Fig. 9c and Fig. 9d show the results of offline

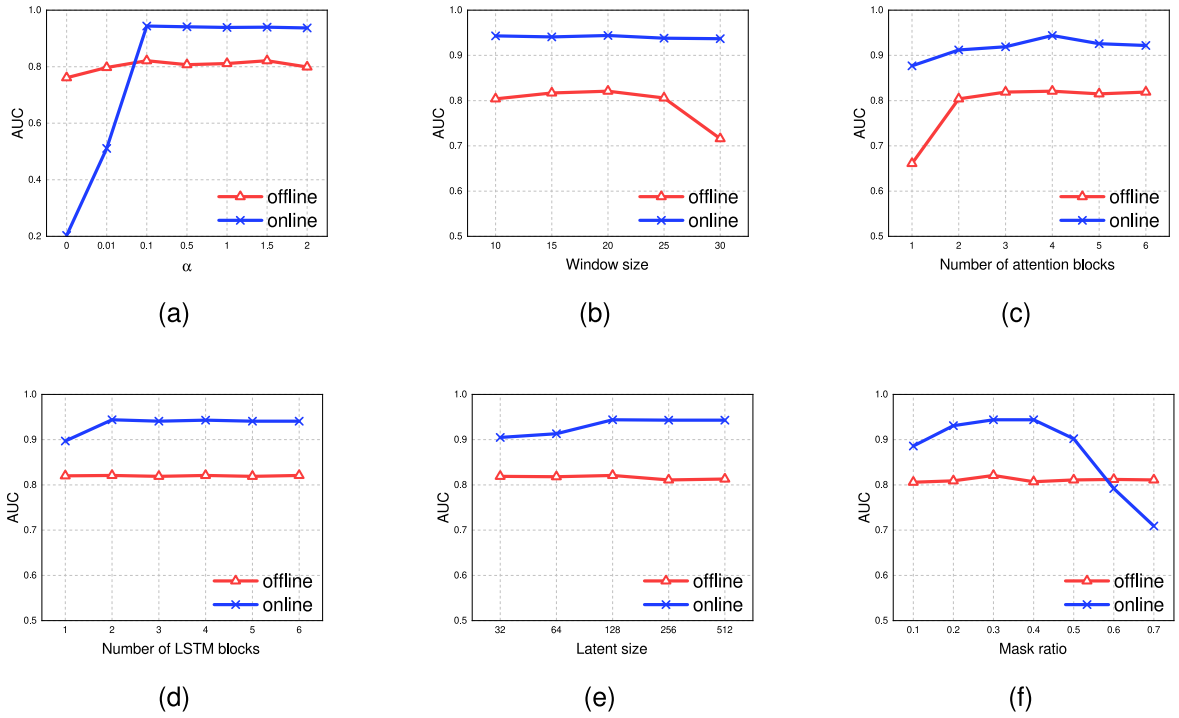

Fig. 8. Analysis of the impact of different component configurations and parameters of LogEncoder on HDFS data on AUC. Fig. 8(a) shown AUC with different α; Fig. 8(b) shown AUC with different window sizes; Fig. 8(c) shown AUC for different numbers of attention blocks; Fig. 8(d) shown AUC for different numbers of LSTM; Fig. 8(e) shown AUC with different latent sizes; Fig. 8(f) shown AUC with different mask ratios. The result demonstrates that LogEncoder is stable with different component configurations and parameters.

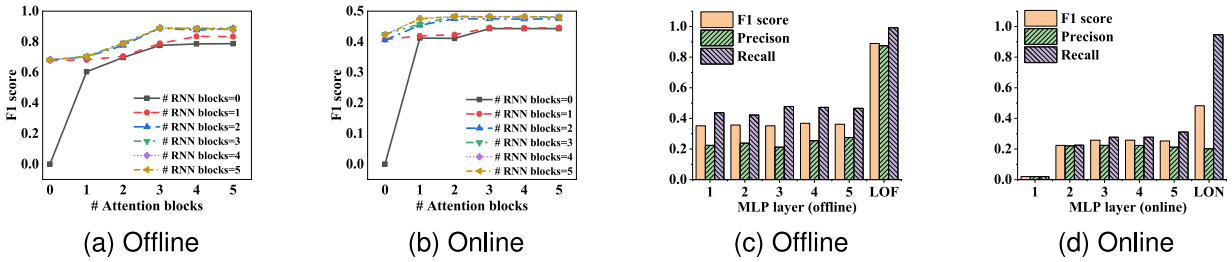

Fig. 9. Performance analysis of attention blocks, rnn blocks, and only MLP layer on the Thunderbird. In (c) and (d), "LOF" refers to LogEncoder(Offline), "LON" refers to LogEncoder(Online).

and online scenarios, respectively. For both offline and online scenarios, our method improves accuracy $+100\%$.

In summary, the ablation studies demonstrate the necessity of attention-based model design by comparing the learning model without the attention blocks. Furthermore, the performance of both the online-based and offline-based methods drops dramatically if the rnn block or attention block is removed.

F. Case Study

In this subsection, we will use a case study to demonstrate that *Log2Emb* preserves the semantic information of log events and mitigates the unstability of logs to some degree. Moreover, we show the distribution of anomaly scores computed by LogEncoder. Finally, we design an experiment to verify the robustness of the LogEncoder.

1) Semantic Information Captured by Log2Emb: To demonstrate that *Log2Emb* can preserve the semantic information of log events, we randomly select the semantic vector of an event in HDFS as a query log event and then retrieve the top-5 semantic vectors that are most similar to it using the cosine similarity function. The results are shown in Table V. We can see that the retrieval result of the top 5 has many keywords that are the same as the query events, which demonstrates that *Log2Emb* preserves the semantic information of log events.

2) Distribution of Anomaly Score: We show the distribution anomaly score of three datasets in Fig. 10. We can observe that the scores of normal log event sequences are mostly smaller than abnormal log event sequences, which indicates that LogEncoder maps the normal log event sequences into the hyper-sphere region of the latent space, while the abnormal log event sequences are randomly distributed over the whole latent space.

3) Robustness of Unstable and Noisy Logs: As described in Section I, log sequences are likely to be changed during the

TABLE V
LOG EVENT SEMANTIC VECTOR TOP-5 SIMILARITY RETRIEVAL RESULT ON HDFS

Query log event:
BLOCK NameSystem*addStoredBlock: blockMap updated:*is added to*size*
Top-5 log events:
BLOCK NameSystem*addStoredBlock: Redundant addStoredBlock request received for*on*size*
BLOCK NameSystem*delete:*is added to invalidSet of*
BLOCK NameSystem*allocateBlock:*
*Received block*src:*dest:*of size*
*Unexpected error trying to delete block*BlockInfo not found in volumeMap*

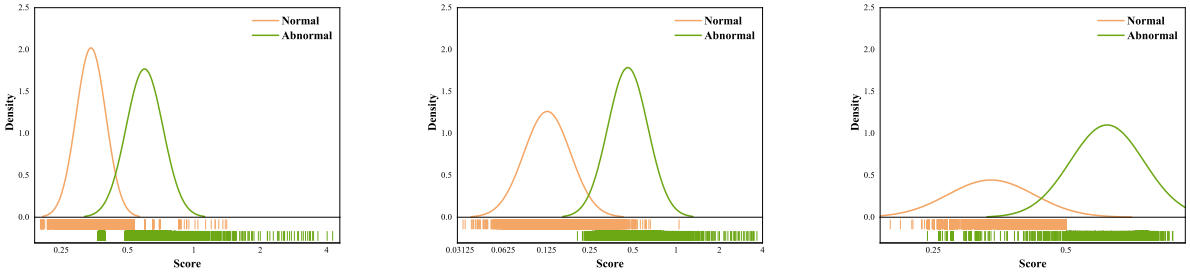

Fig. 10. Anomaly score distribution on three public datasets.

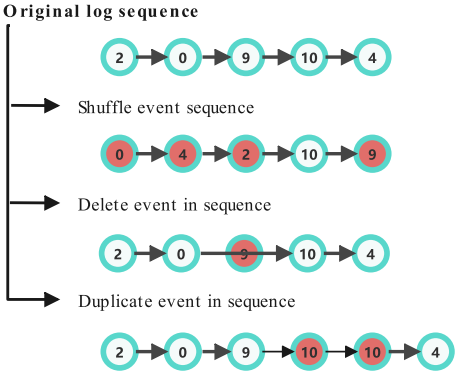

Fig. 11. Different event injection behaviors.

log evolution and collection stage. To show the robustness of our approach in dealing with the noise and unstable of log data, we randomly collected 10K normal logs from three datasets respectively as the training set. We randomly collected 10K normal and anomaly logs as the testing set. Then apply the following operations on testing log event sequence to simulate the unstable and noise in practice, such as data loss, duplication, and disorder when logs are uploaded to the data center. These operations are shown in Fig. 11.

- *Delete:* We randomly remove a few log events in the original log sequence.
- *Duplication:* We randomly select some log events and repeat them several times in the original sequence.
- *Shuffle:* We randomly shuffle the order of a few log events in the original log sequence.

We randomly injected the above operations with different ratios in the testing set. The experimental results on three log datasets are shown in Fig. 12 and Fig. 13. It can be seen that our approach, both offline-based and online-based perform stable with different injection ratios. For example, on the BGL data, when the injection rate is 20%, the F1 score of LogEncoder(Offline) declines just 0.7% compared to an injection rate of 0%(77.9%). The reason is that LogEncoder uses attention mechanisms and contrastive learning to capture the semantic information in log event sequences to eliminate noise and instability with similar semantic meanings.

V. RELATED WORK

In this section, we group existing anomaly detection methods into supervised, unsupervised, or semi-supervised. And review them respectively.

Supervised Anomaly Detection: Many research studies have used supervised learning for anomaly detection. For example, [41] uses a logistic regression model for log-based anomaly detection. Reference [42] proposed a hierarchical transformer structure to model both log template sequences and parameter values for anomaly detection. Reference [7] utilizes an attention-based Bi-LSTM model to capture the contextual information in the log sequences and automatically learns the importance of different log events. Reference [43] uses SVM to learn to correlate critical failures with log data that appeared before them. As mentioned in Section I, supervised methods rely on labeled information and require high-quality data, which leads to extensive overhead and suffers from the class imbalance challenge.

Unsupervised and Semi-supervised Anomaly Detection: Unsupervised and semi-supervised approaches are favored by researchers because they do not need labeled information

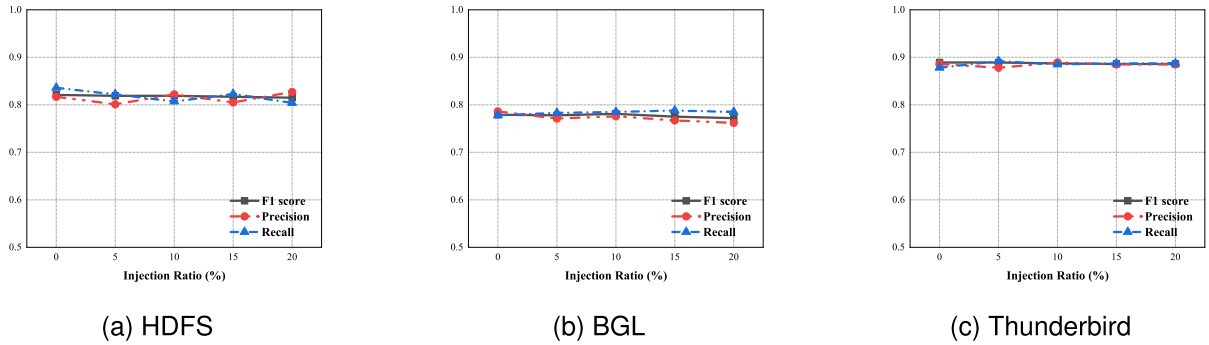

Fig. 12. The LogEncoder(Offline) performance with different injection ratios on three public datasets.

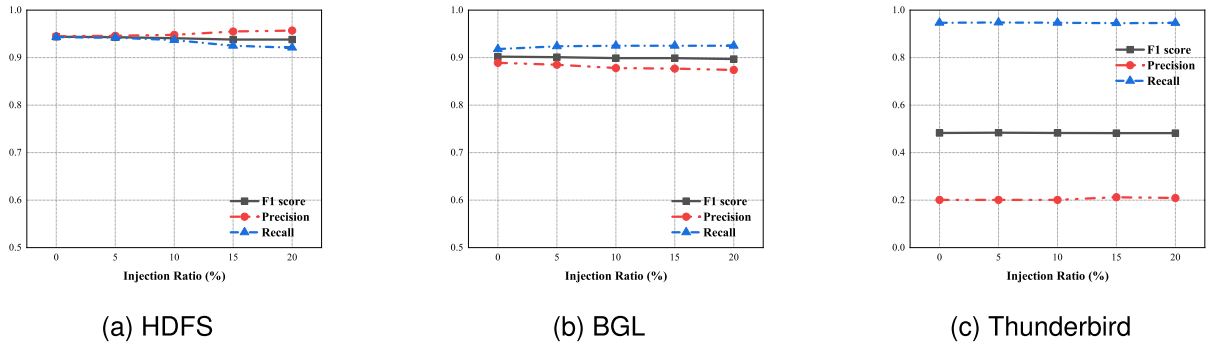

Fig. 13. The LogEncoder(Online) performance with different injection ratios on three public datasets.

and address the challenge of class imbalance. Reference [44] used spatio-temporal information to cluster the system logs into different clusters and separation of the clusters into two classes; an anomalous class and a normal class. Reference [14] proposed a deep neural network model which uses the LSTM to learn log patterns from normal execution to detect anomalies. Reference [12] proposed a framework for log-based anomaly detection, which models a log stream as a natural language sequence. Reference [15] proposed a CNN-based model to learn relationships between log events to detect the anomaly. Reference [3] uses a self-attention neural network to score each log message and then uses data augmentation to set decision boundaries. Reference [9] makes each log sequence into different clusters via cluster technical and then assigns labels to the unlabeled log sequences in each cluster by probabilistic label estimation. Reference [45] tackles the problem of log sequences containing redundant information, which apply a convolution layer and Transformer to capture local correlation and global dependency in long log sequences. Reference [8] designed a multi-scale RNN framework to capture local and global levels of sequential patterns simultaneously. Reference [46] proposed a dictionary-based log parsing approach without parameters. Then, BERT is used to extract features to detect anomalies. This method has robustness for log sequence order changes and time interval changes. Since log-based anomaly detection is affected by log parsing errors, [5] proposed a log-based anomaly detection method that does not require log parsing. The result shows that it can learn the semantic representation and achieve better performance. Reference [47] uses a unified graph representation to model the complex structure of trace logs. It trains gated graph neural networks (GGNNs) based one-class loss. In other scenarios, [4] used a deep learning model to extract log sequence features and train the model using a federated learning framework in a distributed IoT scenario, which prevents information leakage. Reference [4] proposed an anomaly detection model in a collaborative and privacy-preserving manner based on a new federated learning framework. Reference [48] proposed a novel lightweight log anomaly detection algorithm for handling large-scale logs on edge devices. However, unsupervised and semi-supervised anomaly detection methods are typically inferior to supervised anomaly detection due to the lack of prior information about the anomalies.

VI. Conclusion

In this paper, we proposed LogEncoder, a semi-supervised learning-based framework, to represent log event sequences for anomaly detection. LogEncoder not only discriminates between normal and abnormal but also preserves the contextual information of the sequence. Experimental results show that LogEncoder outperforms existing methods and adapts to unstable log data. We demonstrate through parameter analysis that LogEncoder can control the contextual information contained in sequence encoding by hyperparameter. In addition, the statistical distribution of the anomaly scores demonstrates that the abnormal and normal events are separable. Finally, we verified that the vectors obtained by *Log2Emb* contain semantic information and LogEncoder is robust to instability and noise in logs.

REFERENCES

[1] V. Susukailo, I. Opirsky, and O. Yaremko, "Methodology of ISMS establishment against modern cybersecurity threats," in *Future Intent-Based Networking*. Cham, Switzerland: Springer, 2022, pp. 257–271.

[2] A. Behera, C. R. Panigrahi, and B. Pati, "Unstructured log analysis for system anomaly detection—A study," in *Advances in Data Science and Management*. Singapore: Springer, 2022, pp. 497–509.

[3] T. Wittkopp et al., "A2Log: Attentive augmented log anomaly detection," 2021, *arXiv:2109.09537*.

[4] B. Li, S. Ma, R. Deng, C. K.-K. Raymond, and J. Yang, "Federated anomaly detection on system logs for the Internet of Things: A customizable and communication-efficient approach," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1705–1716, Jun. 2022.

[5] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2021, pp. 492–504.

[6] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2016, pp. 207–218.

[7] X. Zhang et al., "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 807–817.

[8] Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen, and J. Tang, "Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection," in *Proc. 27th ACM SIGKDD Conf. Knowl. Disc. Data Min.*, 2021, pp. 3726–3734.

[9] L. Yang et al., "PLELog: Semi-supervised log-based anomaly detection via probabilistic label estimation," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. Compan. (ICSE-Companion)*, 2021, pp. 230–231.

[10] S. Sivakumar, L. S. Videla, T. R. Kumar, J. Nagaraj, S. Itnal, and D. Haritha, "Review on Word2Vec word embedding neural net," in *Proc. Int. Conf. Smart Electron. Commun. (ICOSEC)*, 2020, pp. 282–290.

[11] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2021, pp. 1–8.

[12] W. Meng et al., "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. IJCAI*, vol. 19, 2019, pp. 4739–4745.

[13] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proc. 44th Int. Conf. Softw. Eng.*, 2022, pp. 1356–1367.

[14] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1285–1298.

[15] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *Proc. IEEE 16th Int. Conf Dependable, Auton. Secure Comput. 16th Int. Conf. Pervasive Intell. Comput., 4th Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 151–158.

[16] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey," 2021, *arXiv:2110.11334*.

[17] X. Liu et al., "Self-supervised learning: Generative or contrastive," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 857–876, Jan. 2023.

[18] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proc. IEEE 16th Int. Conf. Data Min. (ICDM)*, 2016, pp. 859–864.

[19] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2009, pp. 1255–1264.

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

[21] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *Proc. IEEE/ACM 26th Int. Conf. Program Comprehension (ICPC)*, 2018, pp. 167–177.

[22] S. Huang et al., "Paddy: An event log parsing approach using dynamic dictionary," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, 2020, pp. 1–8.

[23] J. Zhu et al., "Tools and benchmarks for automated log parsing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. Softw. Eng. Pract. (ICSE-SEIP)*, 2019, pp. 121–130.

[24] J. Pang, X. Pu, and C. Li, "A hybrid algorithm incorporating vector quantization and one-class support vector machine for industrial anomaly detection," *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 8786–8796, Dec. 2022.

[25] K. Lee, D.-W. Kim, K. H. Lee, and D. Lee, "Density-induced support vector data description," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 284–289, Jan. 2007.

[26] L. Ruff et al., "Deep one-class classification," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 4393–4402.

[27] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM Trans. Graph.*, vol. 36, no. 4, p. 72, 2017.

[28] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2018, *arXiv:1807.03748*.

[29] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 9726–9735.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[31] M. Joshi, P. Singh, and N. Zincir-Heywood, "Compromised tweet detection using siamese networks and fasttext representations," in *Proc. 15th Int. Conf. Netw. Service Manage. (CNSM)*, 2019, pp. 1–5.

[32] I. Yahav, O. Shehory, and D. Schwartz, "Comments mining with TF-IDF: The inherent bias and its removal," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 3, pp. 437–450, Mar. 2019.

[33] V. Singla, S. Singla, S. Feizi, and D. Jacobs, "Low curvature activations reduce overfitting in adversarial training," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 16403–16413.

[34] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3039–3071, 4th Quart., 2019.

[35] K. Sohn, C.-L. Li, J. Yoon, M. Jin, and T. Pfister, "Learning and evaluating representations for deep one-class classification," 2021, *arXiv:2011.02578*.

[36] J. Lu, F. Li, C. Liu, L. Li, X. Feng, and J. Xue, "CloudRaid: Detecting distributed concurrency bugs via log mining and enhancement," *IEEE Trans. Softw. Eng.*, vol. 48, no. 2, pp. 662–677, Feb. 2022.

[37] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, 2007, pp. 575–584.

[38] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2010, p. 24.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[40] X. Han, X. Chen, and L.-P. Liu, "GAN ensemble for anomaly detection," 2020, *arXiv:2012.07988*.

[41] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 111–124.

[42] S. Huang et al., "HitAnomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2064–2076, Dec. 2020.

[43] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, 2015, pp. 8–14.

[44] A. Makanju, A. N. Zincir-Heywood, E. E. Milios, and M. Latzel, "Spatio-temporal decomposition, clustering and identification for alert detection in system logs," in *Proc. 27th Annu. ACM Symp. Appl. Comput.*, 2012, pp. 621–628.

[45] C. Zhang, X. Wang, H. Zhang, H. Zhang, and P. Han, "Log sequence anomaly detection based on local information extraction and globally sparse transformer model," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4119–4133, Dec. 2021.

[46] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "SwissLog: Robust and unified deep learning based log anomaly detection for diverse faults," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2020, pp. 92–103.

[47] C. Zhang et al., "DeepTraLog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, Pittsburgh, PA, USA, 2022, pp. 623–634.

[48] Z. Wang, J. Tian, H. Fang, L. Chen, and J. Qin, "LightLog: A lightweight temporal convolutional network for log anomaly detection on the edge," *Comput. Netw.*, vol. 203, Feb. 2022, Art. no. 108616.

Jiaxing Qi received the B.S. degree in software engineering from the Industrial and Commercial College, Hebei University, Baoding, China, in 2017, and the M.S. degree in software engineering from Hebei University, in 2020. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Beihang University, Beijing, China. His main research interests include text analytic and natural language processing.

Zhongzhi Luan (Member, IEEE) is an Associate Professor with the School of Computer Science and Engineering, Beihang University. He has been engaged in teaching and research of distributed computing, high performance computing, and computer architecture for many years. His main research interests include resource management in distributed environment, supporting methods and technologies for application development on heterogeneous computing systems, performance analysis, and optimization of high performance computing applications.

Shaohan Huang received the B.S. and M.S. degrees from Beihang University, Beijing, China, where he is currently pursuing the Ph.D. degree. His current research interests include anomaly detection, text analytic, and natural language processing.

Carol Fung is an Associate Professor with Concordia University. Her research interests include collaborative intrusion detection networks, social networks, security issues in mobile networks and medical systems, security issues in next generation networking, and machine learning in intrusion detection. She serves as an Associate Editor in multiple journals, including IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and *Computer Networks* (Elsevier).

Hailong Yang (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Engineering, Beihang University, in 2014, where he is currently working as an Associate Professor. He has been involved in several scientific projects, such as performance analysis for big data systems and performance optimization for large scale applications. His research interests include parallel and distributed computing, HPC, performance optimization, and energy efficiency. He is a member of China Computer Federation.

Hanlu Li received the B.S. and M.S. degrees in computer science and technology from Beihang University, China, in 2008 and 2011, respectively, where she is currently an Engineer with the School of Computer Science and Engineering.

Danfeng Zhu received the Ph.D. degree in computer science from Beihang University, Beijing, China, where she is currently an Engineer with the Sino-German Joint Software Instuite, School of Computer Science and Engineering. She has published over 10 papers on top academic conferences and journals. Her research interests include computer architecture, memory optimization, and privacy preserving.

Depei Qian received the master's degree from the University of North Texas, Denton, TX, USA, in 1984. He is a Professor with the School of Computer Science and Engineering, Beihang University, China. He served as the Chief Scientist of China National High Technology Program on high performance computing for 20 years. His research interests include innovative technologies in distributed computing, high performance computing, and computer architecture. He is an academician of Chinese Academy of Science, and also a Fellow of the China Computer Federation.