LogRep: Log-based Anomaly Detection by Representing both Semantic and Numeric Information in Raw Messages

Xiaoda Xie*†, Songlei Jian*†, Chenlin Huang§†, Fengyuan Yu†, Yujia Deng‡
†*The College of Computer, National University of Defense Technology*, Changsha, China
‡*China United Network Communications Co., Ltd. Hunan Branch*, Changsha, China
xiexiaoda@hotmail.com, {jiansonglei, clhuang, yufengyuan}@nudt.edu.cn, dengyj7@chinaunicom.cn

***Abstract*—Log-based anomaly detection plays an essential role in various system reliability-related fields including software reliability, network reliability, and so on. System log data is a kind of semi-structured heterogeneous data that contains both semantic parts and numeric variables which both reflect the abnormal behavior of the system. However, existing log-based anomaly detection methods fail to capture the numeric information in raw data which makes them degrade a lot when only limited labeled data is available. To comprehensively capture the semantic and numeric information to enhance anomaly detection, we propose LogRep, a novel representation-based log anomaly detection method that captures both semantic and numeric information in the learned representations. The newly proposed position-aware numeric representation learning module and the attention-based representation fusion module in LogRep solve the heterogeneity problem well in log data. Due to the high quality of learned log representation, LogRep can achieve a comparable anomaly detection performance with SOTA methods while the training data used in LogRep is two orders of magnitude less than that used in SOTA methods. When reducing the training data scale, the performance of SOTA methods drops a lot, while LogRep keeps a stable good performance on two public HDFS dataset, BGL dataset, and one self-collected dataset. Specifically, LogRep achieves the 10.6% and 5.8% improvements over the second-best method in terms of F1 score on the BGL and HDFS datasets when only 1% training data are available respectively.**

***Index Terms*—Log-based anomaly detection, Log representation learning, Limited training data, Log heterogeneity, Log data analysis**

I. Introduction

Log-based anomaly detection has been widely used in various fields as intrusion detection, malicious analysis, and traffic control for preventing software failures and improving system reliability [43, 4, 10, 30, 27]. Over the past decades, many methods have been designed to detect anomalies from log data, including the rule-based detection methods [6, 36, 38], deep learning-based detection methods [8, 47, 22, 11, 50]. Anomaly detection using deep learning model can provide high flexibility and dynamic approaches in different application situations [15, 11, 12, 44, 45], and log data is a suitable data source for extracting the features of the system.

As shown in Figure 1, a raw log message is a kind of semi-structured text, consisting of the **semantic part** and the **numeric part**, and both parts reflect the abnormal properties of logs. The semantic information of the log is the pre-defined formatted information in the log output, describing the event and presented like natural language. The numeric information consists of several variables, such as IP address, file size, packet sequence number, and so on. In Figure 1, the log term 2 and term 3 have the same semantic part while the numeric part is different, which leads to different anomaly results. However, existing log anomaly detection methods always overlook the numeric parts due to the difficulties of representing heterogeneous semi-structured data. As shown in Figure 1, the utilization of the log data can be divided into two major types, including the template-based methods [24, 16, 5, 8] and semantic-embedding based methods [22]. The template-based methods adopt specific parsing tools like Drain [14], Spell [7] and AEL [19] to retrieve the pattern contained inside the logs, and generate a set of templates. For each log item, a particular template is used to represent it. During this conversion process, different log messages are represented by the same semantic template, with all the numeric variable information dropped. The final data is a numeric

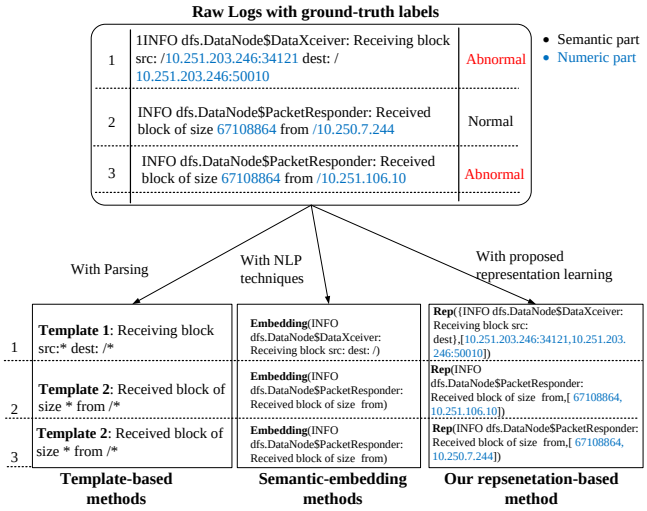


Fig. 1. The raw log demonstrations from HDFS dataset and the comparison between different log processing methods.

*Equal contribution.
§Corresponding author.

sequence consisting of template ID as the representation of the logs. In order to avoid the shortcomings of the template-based methods, the semantic embedding-based methods are designed to use NLP (i.e., natural language processing) tools to process all the semantic information in log messages by dropping the numeric parts. Generally, the existing approaches are not satisfying for the utilization of comprehensive information in log messages. Specifically, we summarize the following three main limitations or challenges of the existing log anomaly detection methods.

- *Limitation 1: Highly dependent on log parser.* The template-based methods adopt log parsing tools to separate semantic templates and numeric parts, but the effectiveness is highly dependent on the log parsing tools, and the current methods are hard to be extended to new datasets. When adopted on data with very different structures, the misunderstanding of the template parsing methods may ignore some valuable information, such as exception type, activity token, and message ID, which leads to poor anomaly detection results.
- *Limitation 2: Failing to utilize the numeric information.* Almost all log anomaly detection methods exclude numeric variables from original log messages in order to obtain templates or embedding conveniently. But as shown in Figure 1, there is abundant information in the numeric parts which are also important for anomaly detection. However, capturing the numeric information in the raw logs is a challenging task. For template-based methods, the numeric property is contradictory to the discrete countable templates. And for the semantic-embedding methods, the numeric parts, such as IP address and block size, are not natural languages in the usual sense so it is hard to unify the numeric part with semantic embedding.
- *Limitation 3: Requiring large-scale training data.* Most anomaly detection methods including both template-based and embedding-based, require a large scale of training data to improve detection precision and recall rate due to their insufficient feature learning abilities. But in real situations, it is not always possible to provide sufficient abnormal samples to support the anomaly feature learning, as anomaly behavior is relatively rare in most cases. Therefore, a better model is required, to reduce the training data scale, and mitigate the sample imbalance problem.

Considering the above limitations, it is vitally important for us to find a better representation method for log data to capture comprehensive information in both semantic and numeric parts without depending on any parser. Then these high-quality representations could be used as training data for the downstream model and therefore improve the model performance in accuracy and recall rate while reducing the requirements for data scale.

To address the heterogeneity in semi-structured log data, we propose a novel representation-based log anomaly detection method, i.e., *LogRep*, to represent both semantic and numeric information in the log representation so that the anomaly detection accuracy could be largely improved with quite limited training data. Specifically, we propose the BERT-based semantic representation learning module to learn the semantic information thoroughly and the position-aware numeric representation learning module to accurately capture the numeric information. Moreover, an attention-based representation fusion module is proposed to unify the two parts to benefit the detection process of the transformer-based anomaly detection module.

In summary, the paper provides the following contributions:

- We proposed a novel representation-based log anomaly detection method *LogRep*, which comprehensively utilizes both semantic and numeric information in raw logs, enabling high anomaly detection accuracy and low demand for training data.
- We proposed a position-aware numeric representation learning module and an attention-based representation fusion module to effectively unify the semantic and numeric information, solving the heterogeneity problem in semi-structured log data well.
- We evaluated LogRep on two public datasets and one self-collected dataset with different scales of training data. The results confirm the effectiveness of LogRep for representing log messages and detecting anomalies. The thorough ablation study also shows the contributions of each module.

II. Related Works

Various solutions have been released to detect anomalies based on logs [12, 15, 13, 34, 48, 21, 52]. The anomaly detection problem can be divided into two stages. The first stage is to generate appropriate feature data for the raw logs, and the second stage is to utilize the feature provided to detect anomalies from it. The feature generation stage largely involves template parsing and semantic embedding in previous research. The raw log messages are semi-structured texts containing both semantic information and numeric variables. The template parsing process generates an enumerated template list, containing the log messages formats appearing in the raw logs found by parsing tools, and then the log messages can be converted to the count vectors according to the matched template id. The methods for log parsing have been widely studied and adopted in anomaly detection [37, 9, 49, 53, 32, 51]. Deeplog and PLELog used Spell [8, 47, 7] to do the template parsing, and LogRobust used Drain [50, 14]. After the raw logs are transformed into an event count vector, there are various models used to detect anomalies from the vectorized data. Many traditional methods used in classification problems are utilized to solve log-based anomaly detection, such as Support Vector Machine (SVM) [24], Invariant Mining (IM) [26], Principal Component Analysis (PCA) [46, 16]. SVM is a widely adopted statistic method and has been tested as an anomaly detection model on the dataset of IBM BlueGene/L Event Logs to predict

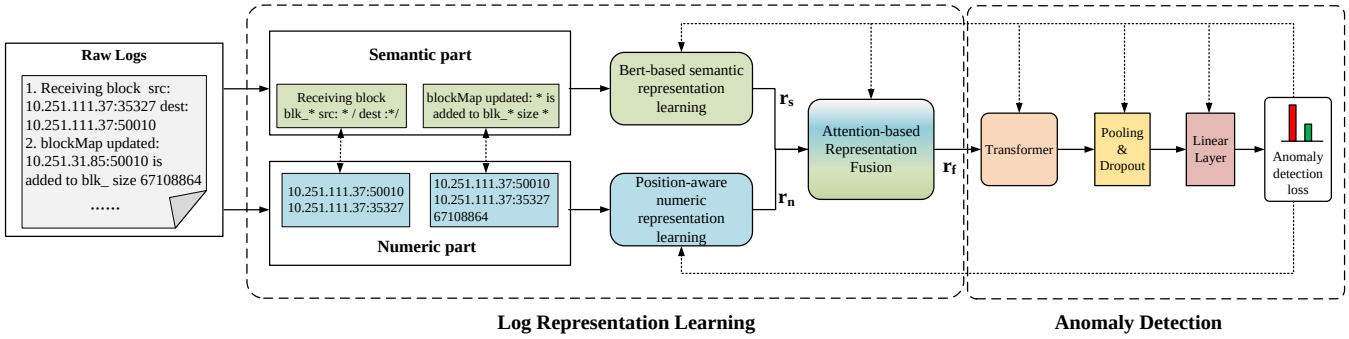


Fig. 2. The overview of LogRep.

failures [24]. PCA is mostly used for dimensional reduction of vectors by calculating the k components that catch the most variance among the high-dimension data. It is first utilized in log-based anomaly detection by finding the patterns between the dimensions of event count vectors introduced by Xu et al. [46, 16]. The IM method mined invariants from the counter vectors, utilizing the mining result to reveal the inherent linear characteristics of program flows and detect anomalies inside.

In order to generate the semantic embedding for logs, there are many methods adopting deep neural network models [11, 18, 42, 39] to process the template parsed from the raw logs. Those template embeddings can be utilized to improve the anomaly detection ability compared to representation methods using template count only. LogRobust [50] and PLELog [47] use pre-trained FastText [20] model to generate template embedding. They aim at learning features reflecting the anomaly from the representation combined with template count and their semantic embedding. After processing by the upstream network, the features are put into the last layer to output the result of classification. LogRobust is a supervised method that requires a suitable amount of labeled datasets. PLELog is a semi-supervised method. It adopts the HDBSCAN [28] algorithm to cluster all log sequences. Based on clustering results, it measures the probability that an unlabeled log sequence belongs to each label. The original deterministic labels are replaced by these probabilistic labels estimated in order to reduce the influence of noise incurred by unsuitable labeling [47]. Log2Vec [29] is a framework with specially designed mechanism to handle OOV (i.e., Out-of-vocabulary) words for covert logs to distributed information. It can also be applied to generate the representation for log messages in an anomaly detection task.

Except for the methods using embedding generated from templates, the semantic embedding-based method creating embedding for each log regardless of its template or format is also studied in work [22]. They remove all the numeric parts from the log messages and then apply the language model to vectorize the texts remaining to create a representation with full embedding for each log. In their approach, Neurallog utilized the pre-trained transformer-based language model BERT [40] to generate the embedding. For anomaly detection, Neurallog used a 1-layer encoder of transformer to classify the log sequence into anomaly or normal ones. However, in the state-of-art approaches, the utilization way of the non-semantic numbers in the logs is still unsolved.

III. The Proposed Method

A. Overview of LogRep

In this paper, we propose a novel method LogRep to overcome the limitations mentioned above and produce effective representations for anomaly detection. As shown in Figure 2, first we extract the semantic part and numeric part from the raw log dataset and establish the representation data for downstream model. Then the two parts are input to two modules, i.e., Bert-based semantic representation learning module and position-aware numeric representation learning module. The produced semantic representation $\mathbf{r}_s$ and numeric representation $\mathbf{r}_n$ are unified into final log representation $\mathbf{r}_f$ for each log term with the attention-based representation fusion module. Finally, the representations are used in transformer-based anomaly detection module. The whole pipeline is trained with the anomaly detection loss and each module is updated by the back-propagated gradient of the loss.

B. Log Representation Learning

In order to obtain the feature as input for the anomaly detection model, the raw logs must first be preprocessed and converted to vector data. Each log item is composed of the semantic part and the numeric part, namely the textual terms and the numeric terms. For example, a log "NameSystem.addStoredBlock: blockMap updated: 10.250.14.38:50010 is added to blk size 67108864" contains several numeric terms, they are the IP address, block id, and data size. The task of representation learning is to properly process those structures and generate the feature data that can effectively present the characteristic information.

1) Semantic Representation Learning: The semantic part is the major textual structure of the log, and can be treated as a kind of natural language. In the example above, the semantic part is "NameSystem.addStoredBlock: blockMap updated: * is added to * size *". In the template-based method, the numbers are considered redundant variables and will not be considered

when constructing the feature data, only the semantic part is recognized and used to match the format of certain template. Then each log is converted into a number indexing a certain template. Every different template can be regarded as a type of event, so the result of the template-based representation is the sequence of numbers that indicate the events identified from the log messages. In order to provide a more informative representation, the word embedding generated for each log template can be brought in to gather semantic information. In previous research, different methods have been adopted to generate template embedding. In LogRobust [50] and PLELog [47], pre-trained FastText model [20] is used to encode templates into sentence vectors.

But the performance of template-based methods are highly dependent on the template parsing process and its adaptiveness to certain log format, an effective solution is adopting the language model to conduct vector generation for the semantic part of all logs. In this type of method, all the numbers in the log message are also removed and only the main textual blocks are left for further processing. In our method, the two parts of the logs are separated, but those numbers will not be simply discarded. We leave them for further process and use a language model to generate the semantic vectors for the textual part. We choose Bidirectional Encoder Representations from Transformer, i.e., BERT, to process the texts, and generate word embedding for each of the logs.

BERT is an open-source machine learning framework for NLP, designed to generate embedding for sentences according to their context. The model was based on transformer, using the attention mechanism to process the relation between the log texts, and the output vector of the model is a numeric vector. The vector generated from log messages with close meaning will have a closer distance. The pre-trained BERT model has been used in generating log representation in previous research [11, 22]. There are several versions of the pre-trained model release, including BERT-Large, BERT-Base, and their variations. In our case, we choose the uncased BERT-Base model, which has 24 layers, 1024 hidden states, and 340 million parameters [40, 1].

After the numeric terms are separated from the log messages, only the semantic part is fed into the BERT uncased model to generate the representation $\mathbf{r_s}$, which is generated by the BERT model:

$$\mathbf{r_s} = \mathbf{BERT}(\mathbf{text}). \tag{1}$$

2) Numeric Representation learning: The embedding $\mathbf{r_s}$ represents the major structure of the log messages. The variables in log messages include the non-numeric ones and numeric ones. The non-numeric ones like the user names, can be dealt with using NLP techniques and integrated into $\mathbf{r_s}$, but the numeric variables are in different forms and require a special representation method. The numeric variables consist of different types of numbers from IP address to packet size, containing important information but hard to utilize. Those numbers like IP address are difficult to process in a traditional way because of their singular structure and the complexity in generating unified feature data to present such numbers. In order to properly make use of those numbers, we design a method based on attention mechanism to generate a position-aware numeric representation to enable the anomaly detection model to learn that unstructured information.

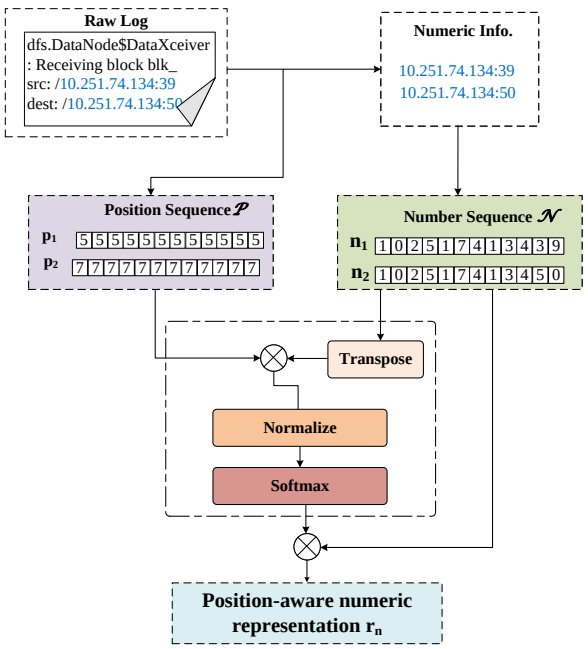


Fig. 3. Position-aware numeric representation learning process.

After we separate the numbers from the logs, first the numbers in each log are presented as a number vector $\mathcal{N} = \{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_l\}$, where $\mathbf{n}_i$ is the numeric terms and $\mathbf{l}$ is the count of the numbers. Then a position pattern $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_l\}$ is generated corresponding to the number vector $\mathcal{N}$, each value $\mathbf{p}_i$ indexing the position of each number pattern in the message respectively. The reason to especially generate such a position vector is due to the characteristic of those numbers. The numeric items in the log and their positions are relevant to each other and this connection is meaningful for anomaly detection. For example, the IP address in a log message may stand for the identity of a data packet sender or receiver respectively when appearing in different positions, and this difference could be representing different system statuses and events, therefore used to distinguish anomaly events. In order to extract such information represented by the numbers and their position in the sentences, we use the attention mechanism to quantify their relevance and generate vector data for training. Attention mechanism has been widely used in many deep learning applications, and proven to be useful in measuring the relationship between variables. We use an attention layer to process the number vector n and the position pattern vector p, instead of using raw numbers in representation, to enable the representation data to contain the complete information of the numbers with their position taken into consideration.

In this attention layer, we calculate the relation between the number vector $\mathbf{n}$ and the position pattern $\mathbf{p}$ using the attention mechanism as follows:

$$\alpha_{\mathbf{n}} = softmax(\frac{\mathbf{p}\mathbf{n}^{\mathbf{T}}}{\sqrt{d_n}}) \tag{2}$$

As the relation between the numbers and their position in the log message is important for representation, we use an attention layer to calculate the relation weight $\alpha_{\mathbf{n}}$. Then the value of $\alpha_{\mathbf{n}}$ is used to get a relation weighted vector $\mathbf{r_n}$:

$$\mathbf{r_n} = \alpha_{\mathbf{n}}\mathbf{n}. \tag{3}$$

As shown in Figure 3, after the attention calculation, both the initial numbers and their position information are brought into the final relation weighted vector r. This vector contains the full information of the numeric part in each log.

3) Representation Fusion: With the embedding $\mathbf{r_s}$ representing the information of the semantic part of the logs, and the vector $\mathbf{r_n}$ representing the numeric part, a comprehensive representation of the full log can be generated thereby. We design a fusion stage with an attention layer to combine the information retrieved from the numbers and the semantic embedding to learn the $\mathbf{r_m}$, which is calculated as follows:

$$\alpha_{\mathbf{s}} = softmax(\frac{\mathbf{r_n}\mathbf{r_s^T}}{\sqrt{d_{r_s}}}) \tag{4}$$

$$\mathbf{r_m} = \alpha_s \mathbf{r_s} \tag{5}$$

After a representation with a numeric part enhanced is generated, the word embedding $\mathbf{r_s}$ which contains the major semantic information of the logs is then fed into a positional encoding layer. In this stage, the positional information is added to the semantic vectors. We use an embedding generation function E utilizing *sin* and *cos* to do a positional encoding for the word embedding $\mathbf{r_s}$, this step is to enable the model to learn the position information of the log sequences [41] as follows:

$$\mathbf{r_e} = E(\mathbf{r_s}) + \mathbf{r_s}. \tag{6}$$

Then in the last step, we combine the positional embedding $\mathbf{r_e}$ which contains the semantic and positional information of the log messages, and the weighted vector $\mathbf{r_m}$:

$$\mathbf{r_f} = \mathbf{r_m} + \mathbf{r_e}. \tag{7}$$

The final representation $\mathbf{r_f}$ does not only contain the information in the semantic part of the logs but also brings in the numeric part, thus providing a more complete representation of the logs. This representation provides all the necessary information including the relationship between numbers and their positions, making the downstream model easier to learn the feature of the logs.

C. Anomaly Detection

The anomaly detection step is to use the feature data to distinguish anomaly items from ordinary ones. It is largely a two-category classification problem and there are various solutions. By utilizing the first k logs as a training dataset, a classification model can learn the anomaly pattern possessed by the log messages, thereby distinguishing the anomaly data from the normal ones. As the feature vectors are extracted from the log messages, an effective approach to process them and accomplish the classification is using the transformer model [41]. The transformer model is organized as stacked encoder layers and decoder layers, with a multi-head self-attention mechanism adopted to calculate similarity connections between data. Then we use the average pooling and dropout layer to process the output of the transformer model, the output is a vector $\mathbf{c}$ for each representation which is calculated as follows:

$$\mathbf{c} = F(\mathbf{r_f}) \tag{8}$$

$$\mathbf{s} = softmax(\mathbf{W} \cdot \mathbf{c}). \tag{9}$$

Then the fully connected layer maps $\mathbf{c}$ into a 2-tuple $\mathbf{s} = (\mathbf{s_0}, \mathbf{s_1})$, where $\mathbf{s_0}$,$\mathbf{s_1}$ are the scores for the two categories respectively, and network parameters $\mathbf{W}$ can be learned in training. The classification result is determined according to the two scores for each category. In training the model, we use the cross-entropy loss to evaluate the performance of the classification result. For classification scores tuple $(\mathbf{s_0}, \mathbf{s_1})$ processed by softmax function and ground truth $\mathbf{y}$, we have a cross-entropy:

$$loss = -\frac{1}{L}\sum_{i=1}^{L}[y_i log(s_{0i}) + (1 - y_i) log(s_{1i})] \tag{10}$$

where L is the number of samples in each training batch, and y_i is the ground truth of the i-th sample, equal to 1 for anomaly samples, and 0 for the others. Using cross entropy as the loss function, we can measure the distance from the model prediction to the real value.

Another limitation of the anomaly detection application is that the model training process requires a large amount of data to learn the anomaly features, but there are considerable difficulties in providing enough data for training. And the imbalance between the number of positive items and negative samples raises a challenge for the model to grab the exact information from the data, since most of the data items are negative ones, namely without anomaly, it will be difficult to learn the precise anomaly behavior patterns. In order to mitigate the sample imbalance issue, we resample the positive log items, with the fraction of the ratio of the positive number to the negative number.

IV. Experiments and Analysis

In this section, we evaluate our method LogRep by answering the following three questions.

RQ1: How does LogRep perform in anomaly detection compared with the state-of-art methods?

RQ2: How does the LogRep perform when the trained data scale is highly limited?

RQ3: To what degree do the newly introduced modules in LogRep contribute to log anomaly detection?

TABLE I
DATASETS USED IN OUR EXPERIMENT.

Dataset	Size	Normal Logs	Anomalies
BGL	743 M	4,747,963	348,460
HDFS	1.5 G	11,175,629	16,838
Self-collected HDFS	123 M	332,326	6433

A. Experimental Setup

1) Datasets: To evaluate the anomaly detection ability of the model, we conduct the experiment on two public datasets and one self-collected dataset as shown in Table I. The BlueGene/L (BGL) dataset [33] and the Hadoop Distributed File System (HDFS) dataset [17, 3] have been widely used in diverse researches of log-based anomaly detection before [47, 22, 24, 31]. The BGL dataset contains 4,399,502 normal logs and 348,460 anomaly logs. The HDFS dataset is collected from a private cloud environment and widely used as log data in anomaly detection. 16,838 logs in the dataset are anomalies, the other 11,175,629 are normal logs. We follow the previous processing method to sample HDFS dataset according to the data nodes ID, and apply a sliding window with the size and step of 20 to the BGL dataset. As the two datasets are both imbalanced between positive and negative item numbers, our resample strategy is adopted to duplicate the positive logs to balance the samples before the training.

Although the two datasets are widely used, the anomaly types are limited [46] and the data structure of the HDFS dataset is not suitable for testing model performance when the training data scale is limited. The anomaly labels in the HDFS dataset are related to certain entities, namely the data block with a certain id. Therefore log sequences are generated according to the block it belongs to, and the log sequences generated this way have different structures and can not be restricted in a unified way as the raw log messages are limited in a certain scale.

In addition to the public datasets, we collect the log messages on an HDFS server under continuous emulated malicious attacks, which cause the usual HDFS server errors including data access timeout, data block io failure, and datanode anomalies. In the new dataset we built, anomaly logs are generated along with the usual work logs, then we manually label the anomaly logs and the normal ones. We test the methods on our newly collected dataset including LogRep and several baselines, to compare the performance of those models on the dataset with artificially created anomaly logs and examine their flexibility on the particular dataset. The dataset we collect contains 332,326 normal logs and 6,433 anomaly logs. The logs are gathered during a period of about 7,000 hours. Different from the public HDFS dataset that contains lots of anomaly logs related to client exception and file operation failure [46], the anomaly logs in our dataset are generated from simulated malicious behavior, including different types of attacks. We test LogRep and other methods on this dataset to examine the ability of the models to accurately identify malicious actions from log message analysis.

2) Comparison Methods: We compare our method with the state-of-art approaches, including the anomaly detection models based on templates and the semantic information. For the methods based on deep learning and semantic embedding of templates, we compare LogRep with the following methods.

- LogRobust [50]: LogRobust is a supervised anomaly detection model. It incorporates a pre-trained FastText model for the generation of template embedding. The feature data generated is input into an attention-based Bi-LSTM model.
- Log2Vec [29]: Log2Vec is a method designed to extract the semantic information of logs, and can be used to generate a representation from raw logs, utilizing the down-stream model to accomplish the anomaly detection tasks.
- PLElog [47]: PLElog is a semi-supervised anomaly detection model which also used a pre-trained FastText model to generate the template embedding. It utilizes the HDBSCAN method to cluster log sequences into groups and uses a GRU-based classifier to detect anomalous log sequences.
- Neurallog [22]: Neurallog is an anomaly detection method without a template parsing process. In the Neurallog model, all numeric variables in the raw logs are discarded and the rest of the texts are processed by a pre-trained BERT language model.

We tested the above methods on the two public datasets HDFS and BGL, and only some of them on the self-collected dataset. We didn't conduct PLElog and Log2Vec on the self-collected dataset because these two methods require a special data format to be processed.

B. Data Preprocessing

An important factor of the model performance evaluation is the training dataset scale. It is obvious that if we train a model with a large scale of data, and only test it on a small fraction of data from the dataset, a high performance is not difficult to achieve. But a practical problem is that with the reduction of the training data length, it will be increasingly more difficult for the model to obtain the important features from the data, therefore weakening the ability of the model to detect anomalies. Using large training set to improve model performance would inevitably increase the training overhead, and can not be widely adopted in an application for the excessive requirements on the training data. In order to measure both the detection performance and the feasibility in practice of the methods, we conduct the experiment on datasets with limited training data scale. For the HDFS dataset, there are over 10 million raw log messages. As the usual way of pre-processing transform those raw logs into log sequences according to their related block id, the intermediate data still contains hundreds of thousands of data items. For BGL dataset, the number of the data items generated through sliding windows is also in that order of magnitude. But using a such scale of training data for anomaly detection is not always feasible. We want to evaluate the performance of the models in a practical situation.

The existing methods for log-based anomaly detection adopt large-scale data and only a small fraction of the dataset is left for testing. The training data fraction is usually 70% or 80% [47, 22, 8, 23], so the training data will include hundreds of thousand data items. There will be a long time and high hardware requirement training process, and the training data is far more than the test data, which weakens the credibility of the evaluation. Therefore we test the models with less training data to examine their practical anomaly detection ability. First, we conduct the experiment with 10% data for training. For HDFS dataset, we use training data from only 50, 000 data nodes, rather than several hundreds of thousands, and for BGL dataset the training data is sampled from 20, 000 sliding windows. Then we further reduce the training data scale to 1%, to evaluate the model performance under extreme circumstances.

C. Evaluation Metrics

We evaluate the performance of these models via the widely used metrics: *Precision*, *Recall*, and *F1-score*. *Precision* is $\frac{TP}{TP+FP}$ and *Recall* is $\frac{TP}{TP+FN}$. And *F1-score* is the measure of overall classification accuracy calculated by $\frac{2 \cdot Precision Recall}{Precision+Recall}$, where TP, FP, TN, FN refer to the number of true positives, false positives, true negatives, and false negatives respectively. Positive here means predicted by the model as anomaly and negative ones are predicted as normal. True or false are determined according to the labels in the datasets, which we consider as ground truths.

D. Implementation Detail

In the representation stage, we used the pre-trained language models including BERT and FastText (for baseline testing). In the anomaly detection stage, we used transformer model as the classifier. For the model parameter setting, we set the multi-head attention mechanism with its head number set to 12. The encoder layers are set to 1 layer, and the decoder layers are set to 0, and the size of the feed-forward network is 2048. The transformer model is trained using optimizer Adam [25], training 80 epochs with an initial learning rate of $2e-4$.

For the baselines compared in experiments, we used their default parameter settings except for the training data length. We used the open-source implementation of LogRobust [50, 2]. For the pre-trained models including FastText [20] and BERT [1, 40] are all fetched from public releases.

Our method is implemented with Python 3.6 and the PyTorch deep learning library [35]. All experiments are conducted on the Ubuntu 18.04.1 server with hardware including CPU Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz and NVIDIA GeForce RTX 2080 Ti $\times 8$.

E. RQ1: How does LogRep perform in anomaly detection compared with the state-of-art methods?

In order to compare LogRep with other methods, we first conduct the experiment with the training data of 10% of the dataset as a training set following the previous research. For each model tested in this way, we first generate the data for training from the raw log messages and then set the same training data length for each model.

TABLE II
PERFORMANCE WITH A TRAINING SET OF 10% DATA. THE BEST RESULTS ARE IN **BOLD**.

Method	Dataset					
	BGL			*HDFS*		
	P	***R***	***F1***	***P***	***R***	***F1***
PLELog	0.572	0.855	0.686	0.782	0.952	0.859
LogRobust	NaN	NaN	NaN*	0.976	0.986	0.981
LogRobust_ov	0.682	0.536	0.601	**0.985**	0.987	**0.986**
Log2Vec	0.234	0.526	0.324	0.806	0.982	0.885
NeuralLog	0.624	0.658	0.640	0.718	0.979	0.829
LogRep	**0.765**	**0.875**	**0.816**	0.881	**0.998**	0.936

*NaN denotes that the model outputs are all negative.

As shown in Table II, LogRep achieves the highest precision, recall and F1-score among all the methods compared, with the improvement ratio of 136%, 39%, and 91% respectively on BGL dataset. In some of the baselines compared, their initial implementation are configured with higher training data ratio, usually requiring much larger training data scale. Although the training ratio of 10%, with several hundreds of thousand logs as training data is enough for our approach, some representation methods cannot extract enough information from the training data and shows a lower accuracy. LogRobust cannot adapt to the imbalanced training data when 10% of the BGL dataset is used for training. Therefore we adjusted the data sampling of the open-source LogRobust implementation[2] to add oversample mechanism, marked as LogRobust_ov. LogRobust_ov achieves the best F1-score on the HDFS dataset due to that the HDFS logs are combined as block-related log groups which is suitable for LSTM model in LogRobust to learn features from template sequences. But on the BGL dataset, the log sequences are organized as time series windows, it raises a challenge to the LSTM model to accurately obtain the feature information especially when the training data scale is limited and anomaly samples are insufficient. The lack of anomaly information causes the vanilla LogRobust model to predict all test logs as normal, so the result only contains *TN* and *FN* items. With the help of oversampling, LogRobust_ov can perform better on BGL dataset, but LogRep still outperforms it.

In our experiment, the BGL dataset with non-overlapped fixing windows, so the feature data are generated without the usage of replicated log messages, and it is more difficult for the anomaly detection models to learn from training data in isolated time windows. Many methods have an obvious performance drop on the BGL dataset. But LogRep can still learn from the limited BGL training data and shows a high performance among other methods.

F. RQ2: How does the LogRep perform when trained data scale is highly limited?

In application situations, there are not always fully prepared large datasets for training. In the experiment above, we conduct the test with 1% data as training data. In BGL dataset, the

TABLE III
PERFORMANCE WITH TRAINING SET OF 1% DATA. THE BEST RESULTS ARE IN **BOLD**.

Method	Dataset					
	BGL			*HDFS*		
	P	***R***	***F1***	***P***	***R***	***F1***
PLELog	0.591	0.865	0.702	**1**	0.24	0.39
LogRobust	NaN	NaN	NaN	NaN	NaN	NaN
LogRobust_ov	NaN	NaN	NaN	0.871	0.916	0.892
Log2Vec	0.387	0.745	0.51	**1**	0.36	0.538
NeuralLog	0.71	**0.782**	0.744	0.941	0.835	0.885
LogRep	**0.994**	0.703	**0.823**	0.947	**0.926**	**0.936**

training data include 5,000 data items, and in HDFS dataset there are 6,000 items.

As shown in Table III, LogRep maintains the best performance on both datasets. On BGL dataset, LogRep achieves an improvement of 11% on *F1-score*. On HDFS dataset, LogRep shows an improvement of 6% on *F1-score* compared to other methods. Most of the models cannot learn the information with such inadequate training samples, leading to low accuracy. But with the advanced representation approach, LogRep remains a remarkable precision and recall even given only 1% data to learn. The performance decline is highlighted in the results on the BGL dataset, as most of the methods will downgrade to extremely low performance, our approach remains a high *F1-score*, which means most of the anomaly can be successfully detected even without sufficient data.

Compared to the results in Table II, more methods shows an extremely low performance in Table III. Although the methods like PLELog and LogRobust achieve good performance when trained with adequate data, which means they can successfully learn the features of the anomaly logs from the representation, they perform worse when the data is insufficient.

According to Table IV, LogRep achieves the best performance compared with the methods without using numeric information. Although the logs generated from the simulated malicious attacks contain less valuable numeric information than the public datasets, LogRep still retrieves some useful features from the raw logs and helps the anomaly detection model improve performance. But the anomaly logs there have very different structures compared to usual HDFS logs, the template-based method can not represent the real feature of the dataset properly and performs worse than the results on the public datasets.

As shown in Figure 4, LogRep helps the anomaly detection model maintain a stable performance compared to other methods, especially when the training data scale is highly limited. In a practical situation, using a training set generated from hundreds of thousands of logs is not always possible. Besides the high-cost process of collecting and data labeling, relying on a large scale of training data would also increase the difficulty of implementation, thereby affecting the feasibility of a method. When the training data scale is reduced, many of the methods can not learn enough information from the limited training data. LogRobust uses a representation that combined the log event count vector and semantic embedding of event

TABLE IV
PERFORMANCE ON THE SELF-COLLECTED DATASET. THE BEST RESULTS ARE IN **BOLD**.

Method	Precision	Recall	F1-score
LogRobust	NaN	NaN	NaN
LogRobust_ov	NaN	NaN	NaN
Log2Vec	0.752	**0.999**	0.859
NeuralLog	0.982	0.972	0.977
LogRep	**0.986**	0.986	**0.986**

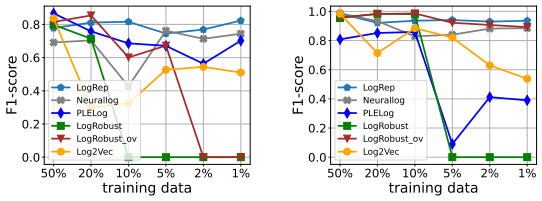
templates, this representation shows good performance when trained with sufficient data. But when the training data is reduced to a limit, the information loss in the template parsing stage is not negligible. Regarding the anomaly samples being fewer in each dataset, when the training scale is limited, it will be very difficult to learn about the features of the anomaly. Even with the help of the oversampling mechanism, the LogRobust_ov model is not available on the BGL dataset. The output tested of the anomaly detection model trained on a such scale of data samples are all negative, which means all samples are predicted as normal. PLELog is also designed for situations with sufficient training data, therefore cannot learn enough information when training data is not sufficient.

G. RQ3: To what degree do the newly introduced modules in LogRep contribute to log anomaly detection?

To evaluate the improvements by introducing the position-aware numeric representation learning module and attention-based representation fusion module for log anomaly detection, especially under situations lacking adequate data samples, we conduct several variants of LogRep to test the contribution of each module. In LogRep, the semantic embedding $\mathbf{s}$, numeric vector $\mathbf{n}$, and the position vector $\mathbf{p}$ are combined with the attention-based fusion method. In Table V, LogRep$_{S}$ indicates the representation using semantic embedding only. LogRep$_{SN}$ uses semantic embedding and numeric vectors to create a concatenated representation. LogRep$_{SP}$ similarly concatenate semantic embedding and positional pattern vectors, and LogRep$_{SPN}$ indicates the representation is generated with all three parts concatenated together.

Compared to the traditional approaches, we use numeric and positional information and designed the method to integrate those unstructured data. So we measure the effectiveness of this improvement from two aspects, including the vanilla form with the semantic sequence only, the combined data of event and number and position vectors via simple concatenation. We use the *F1-score* of the anomaly detection result as a metric to measure the contribution of each part of the representation. In order to present the effectiveness of each representation method, we use the transformer model with the same parameters setting to test the performance, except that some layer dimensions are different, only to match the input feature vector dimension.

As shown in Table V, the result of the ablation study shows that the numeric parts and positional information can be used to improve the performance of anomaly detection, but the



(a) The f1-score with different scale of training data on BGL dataset

(b) The f1-score with different scale of training data on HDFS dataset

Fig. 4. The F1 scores of LogRep, NeuralLog, PLElog, LogRobust and Log2Vec with with different scale of training data on HDFS and BGL dataset.

TABLE V
THE PERFORMANCE WITH DIFFERENT REPRESENTATION METHODS IN TERMS OF F1 SCORE. THE BEST RESULTS ARE IN **BOLD**.

Train data	BGL			HDFS		
	50%	10%	1%	50%	10%	1%
LogRep$_{S}$	0.872	0.640	0.744	**0.986**	0.829	0.637
LogRep$_{SN}$	0.593	0.542	0.318	0.942	0.795	0.158
LogRep$_{SP}$	0.713	0.607	0.252	0.925	0.673	0.574
LogRep$_{SPN}$	0.652	0.585	0.67	0.963	0.832	0.364
LogRep	**0.778**	**0.816**	**0.823**	0.985	**0.936**	**0.936**

simple concatenation of those vectors is not enough. Compared to the LogRep$_{S}$ with pure semantic embedding as feature data, when the positional vector or number vector is concatenated with it, the actual performance is not always better, sometimes LogRep$_{P}$ and LogRep$_{N}$ even performs worse than the embedding without numeric information. But LogRep applied position-aware numeric representation which adopted an attention mechanism to obtain the relation between two types of information. Compared with LogRep$_{SPN}$, LogRep achieves a better performance in each experiment, especially when the data scale is restricted.

V. DISCUSSION

In response to the three limitations proposed in Section I, we make the following discussion and analyze how LogRep addresses the limitations as well as the potential improvement of our work.

Without using any log parser, LogRep outperforms existing template-based methods. As mentioned above, the log parsing work needs a lot of handcrafted work and domain-specific knowledge which limits its use on new datasets. Moreover, the template-based log anomaly detection methods can only extract hundreds or even fewer templates from millions of raw logs. In this parsing process, a lot of valuable information is missing and then the anomaly detection module detects the abnormal behavior through several long template sequences. In this way, the length of the sequence becomes another sensitive hyper-parameter. However, LogRep adopts the SOTA NLP method, i.e., Bert, to represent the semantic information which captures more comprehensive information than template-based methods.

With a novel representation learning module and fusion module, LogRep succeeds to utilize the numeric information. We have demonstrated the usage of numeric information for log anomaly detection with a small case in Figure 1 and the results of the experiments also confirm the benefit of numeric parts. However, utilizing numeric information properly is not a trivial task since the numeric parts occur in various places and with varying lengths. Therefore, we propose the position-aware numeric representation learning module to consider the impact of the position and the different contributions from different numeric parts. Further, we design the attention-based representation fusion module to fuse the numeric representation and semantic representation so that the contribution of each part can vary with the different logs.

With the high-quality representation, LogRep achieves SOTA performance with much less training data. Like the classical classification problem, the training dataset is supposed to provide sufficient positive and negative labeled data, to enable the machine learning models to learn classification boundaries well. This is the same with anomaly detection methods. The specific challenge for anomaly detection is that in real situations it is not always possible to provide sufficient abnormal samples. Therefore, the quality of representation plays a decisive role in the anomaly detection process with limited training data. As shown in Table II and Table III, some methods, e.g., PLELog and LogRobust become invalid as the training data decreases, which reflects the complexity of the detection process. LogRep makes use of both semantic and numeric parts in each raw log and learns the information with neural networks automatically. With a more comprehensive log representation, a simple anomaly detector can detect abnormal behavior easily.

The LogRep is not only a log anomaly detection method but also a general log representation learning method that can be used for classification or sequence prediction tasks. The anomaly detection module can be also substituted by other detectors or classifiers. Moreover, LogRep can be extended to semi-supervised or unsupervised log representation methods.

VI. CONCLUSION

In this paper, we focus on the log-based anomaly detection models about the insufficient utilization of log information in log processing methods and the large data scale requirements in training data. To optimize the anomaly detection model, we propose a novel log representation method LogRep, bringing in the numeric blocks in the log messages, in combination with their position information, providing a more representative feature vector generation approach. To examine the improvement by using the novel feature vector representation, we conduct an experiment on the public datasets BGL, HDFS, and a self-collected dataset.

ACKNOWLEDGMENT

This work was partially supported by the National Natural Science Foundation of China (No. 62002371 and No.62172431), and the National University of Defense Technology Foundation (No. ZK21-17).

References

[1] 2020. Bert Model. https://github.com/google-research/bert.

[2] 2021. DeepLoglizer. https://github.com/logpai/deep-loglizer.

[3] 2021. Public Dataset Samples. https://github.com/logpai/loghub.

[4] Jeanderson Candido, Maurício Aniche, and Arie van Deursen. 2019. Contemporary software monitoring: A systematic literature review. *arXiv e-prints* (2019), arXiv–1912.

[5] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 36–43.

[6] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. 2012. Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering* 39, 6 (2012), 806–821.

[7] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1285–1298. https://doi.org/10.1145/3133956.3134015

[9] Qiang Fu, Jian-Guang Lou, Qingwei Lin, Rui Ding, Dongmei Zhang, and Tao Xie. 2013. Contextual analysis of program logs for understanding system behaviors. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 397–400.

[10] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. 2014. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 24–33.

[11] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. LogBERT: Log Anomaly Detection via BERT. In *2021 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN52387.2021.9534113

[12] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2016. An Evaluation Study on Log Parsing and Its Use in Log Mining. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 28 - July 1, 2016*. IEEE Computer Society, 654–661. https://doi.org/10.1109/DSN.2016.66

[13] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. 2017. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing* 15, 6 (2017), 931–944.

[14] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[15] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. *ACM Comput. Surv.* 54, 6 (2021), 130:1–130:37. https://doi.org/10.1145/3460345

[16] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 207–218.

[17] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448* (2020).

[18] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991* (2015).

[19] Zhen Ming Jiang, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting execution logs to execution events for enterprise applications (short paper). In *2008 The Eighth International Conference on Quality Software*. IEEE, 181–186.

[20] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[21] Subhendu Khatuya, Niloy Ganguly, Jayanta Basak, Madhumita Bharde, and Bivas Mitra. 2018. Adele: Anomaly detection from event log empiricism. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2114–2122.

[22] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 492–504.

[23] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults. In *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*, Marco Vieira, Henrique Madeira, Nuno Antunes, and Zheng Zheng (Eds.). IEEE, 92–103. https://doi.org/10.1109/ISSRE5003.2020.00018

[24] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 583–588.

[25] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[26] Jian-Guang Lou, Qiang Fu, Shenqi Yang, Ye Xu, and Jiang Li. 2010. Mining invariants from console logs for system problem detection. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*.

[27] Adetokunbo Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2010. Fast entropy based alert detection in super computer logs. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 52–58.

[28] Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.* 2, 11 (2017), 205.

[29] Weibin Meng, Ying Liu, Yuheng Huang, Shenglin Zhang, Federico Zaiter, Bingjin Chen, and Dan Pei. 2020. A semantic-aware representation framework for online log analysis. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–7.

[30] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, and Hua Cai. 2013. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems* 24, 6 (2013), 1245–1255.

[31] Karthik Nagaraj, Charles Killian, and Jennifer Neville. 2012. Structured comparative analysis of systems logs to diagnose performance problems. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 353–366.

[32] Adam Oliner, Archana Ganapathi, and Wei Xu. 2012. Advances and challenges in log analysis. *Commun. ACM* 55, 2 (2012), 55–61.

[33] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 575–584.

[34] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H Chin, and Sumayah Alrwais. 2015. Detection of early-stage enterprise infection by mining large-scale log data. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 45–56.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[36] Antonio Pecchia, Domenico Cotroneo, Zbigniew Kalbarczyk, and Ravishankar K Iyer. 2011. Improving log-based field failure data analysis of multi-node computing systems. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. IEEE, 97–108.

[37] Daan Schipper, Maurício Aniche, and Arie van Deursen. 2019. Tracing Back Log Data to its Log Statement: From Research to Practice. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 545–549. https://doi.org/10.1109/MSR.2019.00081

[38] Bianca Schroeder and Garth A Gibson. 2009. A large-scale study of failures in high-performance computing systems. *IEEE transactions on Dependable and Secure Computing* 7, 4 (2009), 337–350.

[39] Stefan Thaler, Vlado Menkonvski, and Milan Petkovic. 2017. Towards a neural language model for signature extraction from forensic logs. In *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, 1–6.

[40] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. *arXiv preprint arXiv:1908.08962v2* (2019).

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[42] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. 2017. Long short-term memory based operation log anomaly detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 236–242.

[43] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-cause metric location for microservice systems via log anomaly detection. In *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, 142–150.

[44] Wei Wang, Songlei Jian, Yusong Tan, Qingbo Wu, and Chenlin Huang. 2022. Representation learning-based network intrusion detection system by capturing explicit and implicit feature interactions. *Computers & Security* 112 (2022), 102537.

[45] Wei Wang, Songlei Jian, Yusong Tan, Qingbo Wu, and Chenlin Huang. 2023. Robust unsupervised network intrusion detection with self-supervised masked context reconstruction. *Computers & Security* 128 (2023), 103131.

[46] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 117–132.

[47] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. PLELog: Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. In *43rd IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 230–231. https://doi.org/10.1109/ICSE-Companion52605.2021.00106

[48] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. 2010. Sherlog:

error diagnosis by connecting clues from run-time logs. In *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*. 143–154.

[49] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1291–1300.

[50] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 807–817. https://doi.org/10.1145/3338906.3338931

[51] Yining Zhao and Haili Xiao. 2016. Extracting log patterns from system logs in large. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 1645–1652.

[52] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 683–694.

[53] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.