Log-based Anomaly Detection using CNN model with Parameter Entity Labeling for Improving Log Preprocessing Approach

Thanaphit Sutthipanyo, Thanadon Lamsan, Woradon Thawornsusin, and Wittawin Susutti*
Department of Mathematics, Faculty of Science, King Mongkut's University of Technology Thonburi, Bangkok, Thailand
*Corresponding Author: wittawin.sus@kmutt.ac.th

Abstract— **To build a reliable system, anomaly detection is the principal task for ensuring the system's security. However, the complexity of systems and software has increased over time. As a result, the likelihood of system failures and vulnerabilities has also grown. For this reason, employing manual anomaly detection approaches is impractical. This work proposes the use of a Convolutional Neural Network for log-based anomaly detection and enhances a log parsing method through parameter entity labeling. We have chosen the ThunderBird and BlueGene/L datasets for our experiments, employing a down-sampling technique to address data imbalance issues and reduce model training time. The results show that when comparing the detection outcomes of models trained with the down-sampled training dataset and models trained with the full training dataset (without using down-sampling), the models trained with the full training dataset exhibit higher recall, while their precision and specificity remain comparable. Additionally, the results indicate that our approach demonstrates slightly better detection performance than the previous log parsing method. Precision, recall, and specificity reach 0.9999, 0.9933, and 0.9914, respectively, when experimenting with the ThunderBird dataset.**

Keywords — **Anomaly Detection, Log, CNN, Parameter Entity Labeling**

I. Introduction

Computer systems are essential for many critical technologies today. However, as systems and software have become more complex, the likelihood of system failures and vulnerabilities has increased. These problems can lead to system crashes or attacks by malicious actors. Anomaly detection is a technique that can help identify these vulnerabilities by looking for anomalies in system behavior. By identifying potential problems early, anomaly detection can help to reduce the likelihood of system failures and vulnerabilities. Due to the volume and velocity of data generated by complex systems, the manually anomaly detection approaches become nearly impossible.

System logs are a sequence of timestamped strings that are generated by computer systems. They are present in almost all computer systems and report information about events that occur during system operation. System logs store information about events and the system's status at a specific timestamp. This makes them a valuable source of information for anomaly monitoring. System logs can be used to identify unusual or unexpected activity, which can be a sign of a potential security threat.

One of challenges in this field of study is insufficiency of abnormal data. This leads to a data imbalance problem, where there are many more normal logs than abnormal logs. Some datasets do not even have labels [1, 2]. This creates barriers to supervised learning. Unsupervised and semi-supervised learning have thus received more attention in this field of study. However, not all deep learning models support unsupervised learning. As a result, many studies have focused on semi-supervised learning that learns from only labeled normal data and tests by normal and abnormal data [3]. Additionally, there are some studies that use sampling techniques for adjusting ratio of normal and abnormal data to make the data balance [4, 5].

Another challenge of log-based anomaly detection is the diversity of log messages. Log messages are constantly changing, and new logs are generated to represent unprecedented events. Additionally, the system's behavior can change any time depending on the constantly changing environment and situation. Furthermore, different threats make this issue more difficult to deal with. The major key that various studies employ to address this issue is to transform words in log content to numeric representation. The most common representation is semantic vector. Semantic vector is a common technique in the field of Natural Language Processing (NLP) for embedding the information of a word or message to numeric form as a vector. This vector represents the meaning and context of that word or message. For example, Word2Vec [6].

In this work, we proposed a log-based anomaly detection method using a Convolutional Neural Network (CNN) model. CNN models have the advantage of having a low number of hyperparameters and training time, as well as the ability to reduce the dimension of the input, making the model lightweight. We also proposed an approach to develop a log feature extraction method before ingesting the data to the model by improving the log parsing method, Drain [7]. Instead of displacing the parameters in the log as previous method, we handle parameters by replacing them with their semantic entity such as replacing "10.251.42.84" with "<IP>". This method is able to keep more context and semantic information of the logs than previous method. We trained and evaluated the proposed model using the ThunderBird and BlueGene/L dataset [8].

II. Related Works

A. Anomaly Detection

Anomaly detection is the task of finding rare or unexpected events in data stream. These events are often referred to as abnormal events. Anomaly detection can be used to discover new knowledge in data, and it is essential for a variety of tasks, especially those that require immediate response [9]. In recent years, anomaly detection has become increasingly important for building secure and reliable systems. This is because systems are becoming increasingly complex and the volume of data that they generate is growing rapidly. Rule-based anomaly detection methods are no longer sufficient for these complex systems. Machine learning models are now being used for anomaly detection. They can be divided into two main approaches: supervised learning and unsupervised learning.

Supervised Learning – The model is trained on labeled data. Liang et al. [10] investigated anomaly in high performance computer system using time series data by employing support vector machines (SVMs), and Peter et al. [11] applied logistic regression to estimate the likelihood of an event being anomalous.

Unsupervised Learning – The models do not require labeled data to train. This makes them suitable for real-world environments where labeled data may be unavailable. Xu et al. [12] proposed using principal component analysis (PCA) for anomaly detection in distributed computing systems. Lin et al. [13] designed an online anomaly detection system that uses clustering models.

B. *Deep Learning for Log-based Anomaly Detection*

In recent years, there has been a growing interest in using deep learning techniques for anomaly detection in system logs because of its success in text and image recognition. One of the first deep learning models for anomaly detection in log data was DeepLog [14], which was proposed by Du et al. in 2017. DeepLog uses a recurrent neural network (RNN) with long short-term memory (LSTM) units to learn the sequence of log messages. DeepLog achieved F1-score of 0.96 for HDFS dataset and 0.98 for OpenStack dataset.

Zhang et al. [15] proposed LogRobust, which is an RNN model with Bi-directional Long Short-Term Memory (Bi-LSTM) along with attention mechanism for dealing with mutability of logs and events sequence. It reached precision of 0.92 and recall of 0.97 for HDFS dataset. Meng et al. [16] proposed LogAnomaly, which is an LSTM model for detecting anomalies in the sequence of log keys. Moreover, they also proposed template2vec embedding method for transforming log keys to vectors. The evaluation result by testing with BlueGene/L (BGL) dataset reached precision and recall of 0.97 and 0.94 respectively. Studiawan et al. [17] employed an artificial neural network applied Gate Recurrent Unit (GRU) for sentiment analysis from logs of operating system which achieved a high F1-score of 0.99 for BGL dataset.

As mentioned above, RNNs have been shown to be effective for anomaly detection in log data but trending off accuracy against complexity that affect to expanding number of parameters and training time. Even RNNs are effective in text classification, structure of log is different to regular text and the nature of log messages is not longer enough to maximize RNNs performance [18]. While CNNs are more lightweight models. Generally, CNNs are employed for image classification. However, Yoon [19] proposed an application of CNNs for text classification. In addition, Rie et al. [20] suggested that CNNs are interesting option for dealing with short text.

The studies which employed CNNs for log-based anomaly detection also have outstanding performance as RNNs. Siyang et. al [18] applied CNNs for NLP with logkey2vec to detect abnormal log in big data systems. Their model achieved precision and recall of 0.97 and 0.99, respectively, for the HDFS dataset. Cheansunan et al. [21] proposed a similar approach but they also compared the training time of their model to RNNs. Their model was able to achieve the same accuracy as RNNs, but it took 2-3 times less time to train. Hashemi et al. [22] proposed OneLog, which is a CNN model for end-to-end anomaly detection. It compresses log parsing, embedding, and classification into one model. OneLog achieved precision and recall of 0.99 for the HDFS dataset. Gu et al. [23] proposed FLOGCNN, which is a CNN model that supports federated learning. This can be helpful for protecting the privacy of data.

C. *Log Preprocessing*

Logs are sequences of text that report the status of a system and the events that occur in it at a particular timestamp. As logs are unstructured data, they need to be preprocessed before they can be ingested into a model. There are two main approaches to log preprocessing.

Log Parsing - This approach identifies and displace the variable and parameter parts of the log message, which are mutable depending on situation and operation of system. The remaining constant parts are then kept. The outputs are the log key, which is a unique event templates, and a list of the extracted parameters from that log. Zhu et al. [24] implemented a tool that automates log parsing using various algorithms such as Drain [7] and Spell [25]. Many recent studies have use log parsing, including the popular works DeepLog by Du et al. [14], LogRobust by Zhang et al. [15], and LogAnomaly by Meng et al. [16].

Log Tokenization – This approach separates word or token in the log message into a list of tokens. The tokens are typically separated by whitespace. This method usually includes turning all characters to lowercase, remove special character and stop words. This method is more flexible than log parsing, as it can handle logs that do not follow a fixed template. However, it can also lose some of the meaning of the original log message. Logsy proposed by Nedelkoski et al. [26] used tokenization before embedding tokens to vectors to avoid the limitations of existing log parsing methods at that time.

Some studies have used a combination of log parsing and tokenization. Catillo et al. [27] proposed a method to estimate anomaly score of each token in log key before ingesting to autoencoder for classifying by estimating the reconstruction error, and FLOGCNN by Gu et al. [23] also using combination method.

After the logs have been preprocessed, they are typically converted into numeric representations using word embedding technique. Word2vec [6] is a popular word embedding model, which is a technique that uses a pre-trained model to transform words or phrases into semantic vectors.

III. Methodology

A. *Log Data Preparation*

The first step in the preprocessing pipeline is to split the dataset into K sets of training sets and testing sets using the K-Fold method. This helps to handle random factors and ensure that the results are consistent. The experimental results of all sets are then averaged.

The next step is to preprocess the log data. We use a combination method that combines log parsing by Drain [7] with tokenization. For conventional log parsing methods, the variable and parameter parts are replaced with placeholders. This means that the original information in these parts is lost. However, in this work, we propose a method that keeps certain information of those parts by replacing them with their semantic entities. For example, the IP address "172.16.96.116" would be replaced with "<IP>". The parameter entities that applied in our method are <IP>, <Number>, <Hexadecimal>, <RT>, <Path>, <Core>, <similar_prev_pattern>, <Identifier>, and <Node>. These entities represent different types of information that can be found in log data. By preserving more semantic information in the log data, our method is able to better understand the meaning of the log data.

The log keys are then constructed as log sequences by concatenating the l log lines preceding the observed line, sorted by timestamp. If the dataset contains N log lines, then the number of log sequences is $N - l$ sets. To deal with the imbalance of normal and abnormal data, we apply down sampling technique to reduce the ratio of normal class to abnormal class for balancing the data. We group the same log sequence and measure each group's size. If the observed line (last line) of that group is labeled as an anomaly and the size of that group is larger than a predefined threshold, then we randomly pick $\rho \cdot n(G_i)$ log sequences in that group, where $\rho \in (0,1)$ is the down sampling ratio and G_i is i-th log sequence.

After that, the log sequences are tokenized into token sequences using regex tokenization. The down sampled datasets are used only for training the model, but the full training datasets are used for evaluation. Then, each token in the token sequences is embedded to a d-dimensional semantic vector using Word2Vec. If there are n total tokens in the log sequence, then all n vectors are vertically concatenated to form a $n \times d$ dimensional matrix, which is the input for the model.

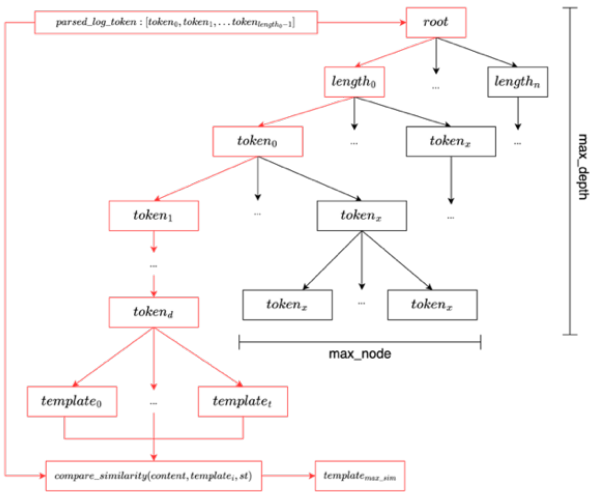

Figure 1. Tree structure of Drain for clustering log key.

B. Log parsing and Parameter Entity Labeling

In this work, we applied Drain [7], an open source[1] implemented by Logpai. Drain is a log parsing algorithm employing regular expression (regex). We customized some parts of Drain's source code for Parameter Entity Labeling (PEL). Drain clusters log keys by applied tree structure as shown in Figure 1. Where the keys of root's child nodes are number of tokens in log key and the keys of deeper level nodes are token appear in 1st ,2nd, 3rd, …, dth position of log key depending on that node's depth. If the token of log key is not matched with any node's keys at its position depth, the new node will be created using that token as its key. At the leave nodes of the tree, they contain set of log keys which are in the same cluster. After traversal to the leave node, the input log key is compared with all log keys in the leave node's cluster by estimating similarity using (1) formular to find most similar log key in that cluster:

$$S(T_1, T_2) = \frac{n(T_1 \cap T_2)}{n(T_1)} \tag{1}$$

Where $T_n = \{token_i \in \log key_n\}$,which T_1 and T_2 must have the same size.

C. Word Embedding

To transform log data into numerical representations, which are appropriate for ingesting to the model, we employed Word2Vec [6], a pre-trained model for embedding words into semantic vectors. We fine-tuned the Word2Vec model to enable it learn words in log massages that do not exist in the model's vocabulary. We did this by using Continuous Bag of Words (CBoW), an algorithm for training the model. The model learns by predicting the observed word using the words around that word.

D. Anomaly Detection Model

In this work, we applied a CNN model for text classification that was proposed by Yoon [19]. Figure 2 show the architecture of the CNN model.

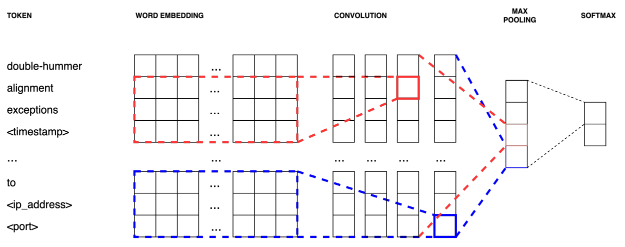

Figure 2. Architecture of CNN model applied for text classification.

Determined a row vector $\vec{x}_i \in \mathbb{R}^d$ is a d-dimensional semantic vector of i-th token in the token sequence of N lines of log. Where that token sequence contains total n tokens and defined matrix that is constructed by vertically concatenating all vectors of that token sequence.

$$X_{1:n} = \vec{x}_1 \oplus \vec{x}_2 \oplus \vec{x}_3 \oplus \dots \oplus \vec{x}_n \tag{2}$$

Where $\oplus$ is vertically concatenating operation and determined $X_{i:i+j} \in \mathbb{R}^{(j+1) \times d}$ refers to matrix constructed by concatenating $\vec{x}_i, \vec{x}_{i+1}, \vec{x}_{i+2}, \dots, \vec{x}_{i+j}$ vertically and $m_i \in \mathbb{R}$ is features extracted from convolution when applying kernel $W \in \mathbb{R}^{k \times d}$ with $X_{i:i+k-1}$ and represent each row of W as $\vec{w}_i \in \mathbb{R}^d$, so that:

$$m_i = f\left(\sum_{l=0}^{k-1} \vec{w}_{l+1} \cdot \vec{x}_{i+l} + \beta\right) \tag{3}$$

[1] https://github.com/logpai/logparser/tree/master/logparser/Drain

Defined $\beta \in \mathbb{R}$ is bias and f is activation function. m_i is determined by applied kernel on $X_{1:n}$ at each area in $\{X_{1:k}, \dots, X_{n-k+1:n}\}$ to build the feature map.

$$\vec{m} = [m_1, m_2, \dots, m_{n-k+1}] \tag{4}$$

Where $\vec{m} \in \mathbb{R}^{n-k+1}$ is feature map, then carry it to max-over time pooling layer to extract highest value feature of each feature map.

$$\hat{m} = max\{\vec{m}\} \tag{5}$$

Finally, these features are carried to Fully Connected (FC) layer with SoftMax as an activation function and the output is the probability of each class (in this case is probability to be normal and abnormal).

$$\hat{y}_i = \sigma\left(\sum_{j=1} c_{ij} \cdot \hat{m}_j + \beta_{FC}\right)_i \tag{6}$$

Where $\hat{m}_j$ is output from max pooling feature map of j-th kernel and β_{FC} is bias of FC layer and $\sigma(\mathrm{x})_i$ is SoftMax function of class i.

To train the model, we employed Adam as a model optimizer and cross-entropy as a loss function. We also determined the weight of each class to deal with the class imbalance problem.

E. Evaluation

To evaluate the performance of the predictive model, we used precision, recall, and specificity. F1-score was employed to show the overview of model predictive accuracy. We used two datasets for the experiments, Thunderbird and BlueGene/L (BGL) dataset, which contain logs of high-performance computer [8].

F. Experimental Scope and Setting

In this work, we investigated the impact of down sampling normal data and comparing between conventional log parsing method and log parsing with parameter entity labeling method on the predictive performance of the model. We down sampled normal data with the condition that the log sequence group is normative group, and its size is larger than the 90th percentile of all groups size. We employed Word2Vec pretrained by Google News 300, fine-tuned at 10 epochs, and trained the CNN model at 5 epochs with a learning rate of 0.001. The overview of the proposed method is shown in Figure 3.

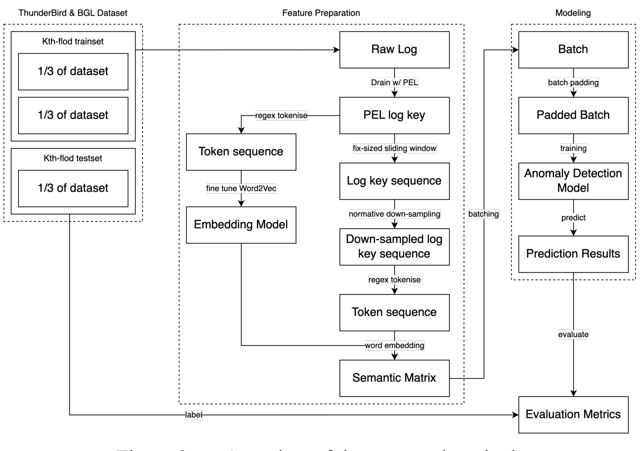

Figure 3. Overview of the proposed method.

IV. Experimental Results

The experimental results were divided into two sections. The first section presents the results of anomaly detection with down sampling normal data. The second section presents the results of anomaly detection compared between conventional log parsing and log parsing with parameter entity labeling. The ThunderBird and BGL datasets were split by 3-Fold method as shown in Table I and Table II.

TABLE I. Number and percentage of normal and abnormal log sequences of each fold regarding to ThunderBird

ThunderBird	**Number of Data**		
	Fold-1	***Fold-2***	***Fold-3***
Training dataset			
Normal data	6,313,763	6,500,297	6,428,604
Abnormal data	336,321	149,787	221,480
Total	**6,650,084**	**6,650,084**	**6,650,084**
Testing dataset			
Normal data	3,307,567	3,121,033	3,192,726
Abnormal data	17,473	204,007	132,314
Total	**3,325,040**	**3,325,040**	**3,325,040**

TABLE II. Number and percentage of normal and abnormal log sequences of each fold regarding to BGL

BlueGene/L	**Number of Data (Percentage)**		
	Fold-1	***Fold-2***	***Fold-3***
Training dataset			
Normal data	2,888,228	2,821,222	3,020,606
Abnormal data	254,097	321,103	121,718
Total	**3,142,325**	**3,142,325**	**3,142,324**
Testing dataset			
Normal data	1,476,798	1,543,804	1,344,420
Abnormal data	94,362	27,356	226,741
Total	**1,571,160**	**1,571,160**	**1,571,161**

A. Result of Anomaly Detection with Down Sampling Normal Data

We down sampled normal data with three ratios: 30%, 35%, and 40%. We evaluated the anomaly detection model by the four metrics, averaged from 3-Fold dataset.

The results in table III and table IV showed that down sampling normal data did not improve the predictive performance of the model. It aggravated model's accuracy to predict normal data, as evidenced by the decreasing recall when the normal data was decreased. Precision and specificity were only slightly affected.

TABLE III. Predictive performance results of anomaly detection with down sampling regarding to ThunderBird

ThunderBird's training dataset				
Ratio	**Precision**	**Recall**	**Specificity**	**F1-score**
30%	**1.0000 (±0.0000)**	0.9527 (±0.0267)	**0.9998 (±0.0001)**	0.9756 (±0.0140)
35%	0.9999 (±0.0001)	0.9321 (±0.0655)	0.9982 (±0.0017)	0.9636 (±0.0359)
40%	1.0000 (±0.0000)	0.9863 (±0.0017)	0.9992 (±0.0001)	0.9931 (±0.0009)
100%	0.9991 (±0.0012)	**0.9986 (±0.0012)**	0.9834 (±0.0220)	**0.9989 (±0.0003)**
ThunderBird's testing dataset				
30%	**1.0000 (±0.0000)**	0.9190 (±0.0572)	**0.9986 (±0.0017)**	0.9569 (±0.0308)
35%	1.0000 (±0.0000)	0.9107 (±0.0990)	0.9982 (±0.0022)	0.9503 (±0.0562)
40%	1.0000 (±0.0000)	0.9883 (±0.0084)	0.9985 (±0.0012)	0.9941 (±0.0043)
100%	0.9999 (±0.0000)	**0.9933 (±0.0094)**	0.9914 (±0.0112)	**0.9966 (±0.0047)**

TABLE IV. PREDICTIVE PERFORMANCE RESULTS OF ANOMALY DETECTION WITH DOWN SAMPLING REGARDING TO BGL

BlueGene/L's training dataset				
Ratio	**Precision**	**Recall**	**Specificity**	**F1-score**
30%	0.9922 (±0.0107)	0.5699 (±0.3203)	0.9220 (±0.1079)	0.6587 (±0.3113)
35%	**0.9998 (±0.0002)**	0.4920 (±0.3726)	**0.9993 (±0.0003)**	0.5737 (±0.3485)
40%	0.9997 (±0.0003)	0.4782 (±0.3517)	0.9990 (±0.0009)	0.5663 (±0.3437)
100%	0.9988 (±0.0010)	**0.8121 (±0.0956)**	0.9873 (±0.0099)	**0.8926 (±0.0598)**
BlueGene/L's testing dataset				
30%	**0.9994 (±0.0006)**	0.5308 (±0.3214)	0.9913 (±0.0069)	0.6375 (±0.2683)
35%	0.9982 (±0.0023)	0.4457 (±0.3802)	0.9741 (±0.0358)	0.5228 (±0.3570)
40%	0.9993 (±0.0006)	0.2771 (±0.1781)	**0.9961 (±0.0047)**	0.4006 (±0.2384)
100%	0.9989 (±0.0012)	**0.8704 (±0.0868)**	0.9801 (±0.0178)	**0.9279 (±0.0499)**

B. *Result of Anomaly Detection with Log Parsing and Parameter Entity Labeling*

The results of log parsing and parameter entity labeling with ThunderBird dataset were defective. For some parameters or some parts in the log content were not parameter part, they were labeled as "<similar_prev_pattern>". This issue was caused by underdetermining the max depth or max child node of the tree and incomplete customization of Drain to support parameter entity labeling. The sample of pestilent log keys caused by this issue are showed in table V.

However, this issue was solved for BGL dataset[2] by executing parameter entity labeling process again with only the parts that were labeled as "<similar_prev_pattern>".

The table VI and VII show the anomaly detection predictive performance of model trained by the ThunderBird and BGL datasets, respectively, comparing between using the conventional log parsing method by Drain and the proposed log parsing method along with parameter entity labeling by customized Drain.

C. *Comparing Proposed Approach with Other State-Of-the-Art Works*

In this section, we compare the predictive performance of our proposed approach with other state-of-the-art works that apply deep learning for anomaly detection. We selected works that used the ThunderBird and BGL datasets as we did, which are Farzad et al. [28], Sasho et al. [26], Guo et al. [29], and Van-Hoang et al. [30]. The results are shown in Table VIII and Table IX.

TABLE V. SAMPLES OF PESTILENT LOG KEYS OF THUNDERBIRD CAUSED BY DEFECTIVE LOG PARSING WITH PARAMETER ENTITY LABELING METHOD

Original log content	**Without PEL**	**With PEL**
data_thread() got not answer from any [Thunderbird_A6] datasource	data_thread() got not answer from any <*> datasource	data_thread() got not answer from any <similar_prev_pattern> datasource
synchronized to 10.100.30.250, stratum 3	synchronized to <*> stratum <*>	synchronized to <similar_prev_pattern> stratum <Numbers>

[2] The mentioned issue has not been solved for ThunderBird dataset because ThunderBird is enormous, it requires the long processing time thus we could not complete it on our schedule.

TABLE VI. PREDICTIVE PERFORMANCE RESULTS OF COMPARING BETWEEN CONVENTIONAL LOG PARSING AND LOG PARSING WITH PARAMETER ENTITY LABELING REGARDING TO THUNDERBIRD

ThunderBird dataset				
Conventional log parsing method				
Dataset	**Precision**	**Recall**	**Specificity**	**F1-score**
Train set	0.9983 (±0.0022)	0.9882 (±0.0165)	0.9679 (±0.0411)	0.9931 (±0.0078)
Test set	0.9998 (±0.0001)	0.9831 (±0.0233)	0.9785 (±0.0262)	0.9912 (±0.0120)
Log parsing with parameter entity labeling method				
Train set	**0.9991 (±0.0012)**	**0.9986 (±0.0012)**	**0.9834 (±0.0220)**	**0.9989 (±0.0003)**
Test set	**0.9999 (±0.0000)**	**0.9933 (±0.0094)**	**0.9914 (±0.0112)**	**0.9966 (±0.0047)**

TABLE VII. PREDICTIVE PERFORMANCE RESULTS OF COMPARING BETWEEN CONVENTIONAL LOG PARSING AND LOG PARSING WITH PARAMETER ENTITY LABELING REGARDING TO BGL

BlueGene/L dataset				
Conventional log parsing method				
Dataset	**Precision**	**Recall**	**Specificity**	**F1-score**
Train set	0.9818 (±0.0251)	0.5129 (±0.3320)	0.9284 (±0.0898)	0.6138 (±0.2810)
Test set	0.9505 (±0.0648)	0.6047 (±0.2955)	0.6593 (±0.4341)	0.6849 (±0.2091)
Log parsing with parameter entity labeling method				
Train set	**0.9988 (±0.0010)**	**0.8121 (±0.0956)**	**0.9873 (±0.0099)**	**0.8926 (±0.0598)**
Test set	**0.9989 (±0.0012)**	**0.8704 (±0.0868)**	**0.9801 (±0.0178)**	**0.9279 (±0.0499)**

TABLE VIII. COMPARING WITH OTHER STATE-OF-THE-ART WORKS REGARDING TO THUNDERBIRD

Work [ref.]	**Precision**	**Recall**	**Specificity**	**F1-score**
Auto-LSTM [28]	99.8%	98.2%	99.8%	99.0%
Auto-BLSTM [28]	99.8%	98.9%	**99.9%**	99.3%
Auto-GRU [28]	**99.9%**	98.5%	**99.9%**	99.3%
Logsy [26]	99.~%	**100.~%**	-	99.~%
LogBERT [29]	96.6%	96.5%	-	96.6%
NeuralLog [30]	93.~%	**100.~%**	-	96.~%
Proposed method	**99.9%**	99.3%	99.1%	**99.6%**

TABLE IX. COMPARING WITH OTHER STATE-OF-THE-ART WORKS REGARDING TO BGL

Work [ref.]	**Precision**	**Recall**	**Specificity**	**F1-score**
Auto-LSTM [28]	99.3%	99.8%	91.3%	99.5%
Auto-BLSTM [28]	99.4%	**99.9%**	92.1%	**99.6%**
Auto-GRU [28]	99.3%	**99.9%**	91.6%	**99.6%**
Logsy [26]	29.~%	98.~%	-	44.~%
LogBERT [29]	89.4%	92.3%	-	90.8%
NeuralLog [30]	98.~%	98.~%	-	98.~%
Proposed method	**99.9%**	87.0%	**98.0%**	92.8%

V. DISCUSSING AND CONCLUSION

We applied parameter entity labeling to improve the log parsing method. However, this method still has limitation, such as the use of regular expressions. According to the comparison of predictive results between the conventional log parsing method and the log parsing with parameter entity labeling method, the proposed method improves all metrics for both the ThunderBird and BGL dataset. In addition, the results of down sampling indicate that proposed model can detect anomalies even when the data is imbalanced. Observable from specificity, which is not dominated by the positive class, which is the majority of the data, reaching 0.99 for both the ThunderBird and BGL datasets. This reflects the model's ability to predict anomalies precisely and deal with imbalanced data. When comparing the evaluation metrics of each down sampling ratio, we found that precision and specificity are unchanged for the ThunderBird and BGL datasets. However, recall decreases when the ratio of normal class to abnormal class decreases, as is evident in the results for BGL.

ACKNOWLEDGMENT

We are grateful to the Mathematics Department, Faculty of Science, at King Mongkut's University of Technology Thonburi for their generous financial support.

REFERENCES

[1] S. He, J. Zhu, P. He and M. R. Lyu, "Loghub: a large collection of system log datasets towards automated log analytics," *arXiv:2008.06448,* 2020.

[2] A. Chuvakin, "Public security log sharing site," 8 November 2010. [Online]. Available: https://log-sharing.dreamhosters.com/. [Accessed 3 December 2022].

[3] M. Landauer, S. Onder, F. Skopik and M. Wurzenberger, "Deep Learning for Anomaly Detection in Log Data: A Survey," arXiv:2207.03820, 2022.

[4] A. Farzad and T. A. Gulliver, "Log Message Anomaly Detection with Oversampling," International Journal of Artificial Intelligence & Applications (IJAIA), vol. 11, no. 4, pp. 53-65, 2020.

[5] P. Sun, E. Yuepeng, T. Li, Y. Wu, J. Ge, J. You and B. Wu, "Context-Aware Learning for Anomaly Detection with Imbalanced Log Data," in 2020 IEEE 22nd International Conference on High Performance Computing and Communications, Yanuca Island, Cuvu, Fiji, 2020.

[6] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient estimation of word representations in vector space," arXiv:1301.3781, 2013.

[7] P. He, J. Zhu, Z. Zheng and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," IEEE International Conference on Web Services (ICWS), 2017.

[8] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in 37th annual IEEE/IFIP international conference on dependable systems and networks, 2007.

[9] Xhafa, P. Schneider and Fatos, "Chapter 3 - Anomaly detection: Concepts and methods," in Anomaly Detection and Complex Event Processing over IoT Data Streams With Application to eHealth and Patient Data Monitoring, Academic Press, 2022, pp. 49-66.

[10] Y. Liang, Y. Zhang, H. Xiong and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," 7th International Conference on Data Mining, 2007.

[11] P. Bodík, M. Goldszmidt, A. Fox and H. Andersen, "Fingerprinting the Datacenter: Automated Classification of Performance Crises," EuroSys 2010, 2009.

[12] L. H. A. F. D. P. a. M. I. J. Wei Xu, "Detecting large-scale system problems by mining console logs," SOSP'09, p. 117–132, 2009.

[13] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang and X. Chen, "Log clustering based problem identification for online service systems," ICSE-C'16 IEEE, p. 102–111, 2016.

[14] M. Du, F. Li, G. Zheng and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," CCS'17, vol. 1, p. 1285–1298, 2017.

[15] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen and D. Zhang, "Robust log-based anomaly detection on unstable log data," the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, p. 807–817, 2019.

[16] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," IJCAI, vol. 19, no. 7, p. 4739–4745, 2019.

[17] H. Studiawan, F. Sohel and C. Payne, "Anomaly Detection in Operating System Logs with Deep Learning-Based Sentiment Analysis," Anomaly Detection in Operating System Logs with Deep Learning-Based Sentiment Analysis, vol. 18, no. 5, pp. 2136-2148, 2021.

[18] S. Lu, X. Wei, Y. Li and L. Wang, " Detecting anomaly in big data system logs using convolutional neural network," 2018 IEEE 16th Intl Conf on Dependable, p. 151–158, 2018.

[19] Y. Kim, "Convolutional Neural Networks for Sentence Classification," arXiv:1408.5882, 2014.

[20] R. Johnson and T. Zhang, "Effective Use of Word Order for Text Categorization with Convolutional Neural Networks," arXiv:1412.1058, 2015.

[21] P. Cheansunan and P. Phunchongharn, "Detecting Anomalous Events on Distributed Systems Using Convolutional Neural Networks," 2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST), pp. 1-5, 2019.

[22] S. Hashemi and M. Mäntylä, "OneLog: Towards End-to-End Training in Software Log Anomaly Detection," arXiv:2104.07324, 2021.

[23] S. Gu, Y. Chu, W. Zhang, P. Liu, Q. Yin and Q. Li, "Research on system log anomaly detection combining two-way slice gru and ga-attention mechanism," Artificial Intelligence and Big Data (ICAIBD), p. 577– 583, 2021.

[24] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng and M. R. Lyu., "Tools and Benchmarks for Automated Log Parsing," International Conference on Software Engineering (ICSE), 2019.

[25] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," IEEE 16th International Conference on Data Mining (ICDM), 2016.

[26] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso and O. Kao, "Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs," 2020 IEEE International Conference on Data Mining (ICDM), p. 1196–1201, 2020.

[27] M. Catillo, A. Pecchia and U. Villano, "Autolog: Anomaly detection by deep autoencoding of system logs," Expert Systems with Applications, vol. 191, pp. 116263-116284, 2022.

[28] A. Farzad and T. A. Gulliver, "Log Message Anomaly Detection and Classification Using Auto-B/LSTM and Auto-GRU," arXiv:1911.08744, 2019.

[29] H. Guo, S. Yuan and X. Wu, "LogBERT: Log Anomaly Detection via BERT," in 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 2021.

[30] V.-H. Le and H. Zhang, "Log-based Anomaly Detection Without Log Parsing," arXiv:2108.01955 [cs.SE], 2021.