JOURNAL OF INFORMATION SCIENCE AND ENGINEERING XX, XXXX-XXXX (2024)
DOI: XX.XXX.XXXX.XX

Anomaly Detection on System Log with Language Modeling and Reconstruction Gate

SHUN-WEN HSIAO+ AND CHIH-CHUNG TSENG
Department of Management Information Systems
National Chengchi University
No. 64, Sec. 2, Zhinan Rd., Wenshan Dist., Taipei, 11605, Taiwan
E-mail: {hsiaom; 109356019}@nccu.edu.tw

System log is generally existing in software applications to help operators manage their services. Misbehavior and bugs in a system can cause vulnerabilities and put services in danger. Therefore, anomaly detection is adopted to aid operators to discover anomalous events in system log. With the development of deep learning models in Natural Language Processing (NLP), recent researches utilize language representation models to take semantics behind the log into consideration. The approach strengthens the adaptability of an anomaly detection model to log events with changing formats. We propose the Bert-based One-class classification with an explicit Reconstruction Gate (BORG) to recognize the benign session behavior of system log in different levels. Instead of a supervised classifier, our method integrates the anomaly detection objective with language representation, and comprise a composite malicious score in the detection phase to reflect the abnormality in trivial events. We evaluate our concept under two log data sets with contrasting statistic properties. The result shows the robustness of our method to challenging log data. The experiments and analysis are also presented to explain our outcomes.

Keywords: log data analysis, anomaly detection, natural language processing, deep learning, language model

1. Introduction

Anomaly detection has been studied for decades in data mining. The main objective is to learn a model only describing the normal samples. This assumes samples in a data set for anomaly detection are either normal or abnormal. Abnormal samples should have significant deviation from the normals. This property makes binary classification impractical in these data sets, since unseen abnormal samples can be completely different from the patterns previously learned by a classifier. For that reason, anomaly detection recognized samples only by the description generated from the normal model. Samples well-described are considered normal. Contrarily, samples poorly-described are regarded as anomalies. Due to the robustness of the concept, anomaly detection has been applied in many fields including cybersecurity, finance, medical science, etc. Analysis on system log is also one of its applications.

Received June xx, 2024; revised June xx, 2024; accepted September xx, 2024.
+Corresponding author

System log is a diary existing in most of the systems to record significant events. The existence of system log is inevitable because system developers require the information to troubleshoot problems whenever an error occurs. System log helps developers to understand the state of their programs and track down the root cause. Therefore, anomalous records in system log can indicate programming bugs and misbehavior of a system. Since these anomalies may be further exploited by adversaries and cause systems exposed to danger, discovering anomalies in the early stage is crucial for system operation.

To analyze the messages in system log, studies before tend to process the text by parsing in a heuristic manner. Some studies design log parsers to extract the common patterns, which we call them templates in our research. Each log message can be represented with the numerical identifier of the template it belongs. Hence, a group of consecutive log messages, which we call it a log session, can be described by a sequence of numbers. However, owing to the strengthened flexibility in modern software design, alterations to a system can be performed more frequently than before. By these means, operators nowadays not only have a great amount of logs, but these logs can be even changing their formats every service update. A log parser may wrongly identify the template of an event and causes false alarms in a detection model. In addition, an environment consisting of more than one system may require specific observation and domain-knowledge in each system to design and configure a parser. To mitigate these problems, we proposed a method based on a tokenizer in the field of NLP to generate contextualized language embedding and detect anomalous events in system log. This can eliminate the reliance on the correctness of current log parsers, and also improve the model robustness toward unstable log data.

Detecting anomalies can be generally separated into two phases, data description and anomaly identification. Many studies [1, 2, 3, 4, 5, 6, 7] adopt log parsers to represent log messages with a sequence of numerical identifiers. Statistical properties and sequential association can be then performed to describe the normal behavior in log sessions. In recent years, there are some studies emerging and taking semantics into their detection model. LogAnomaly [8] utilizes synonyms and antonyms to generate word vectors, and use the vectors in their detection. LogRobust [9] takes the advantage of the off-the-shelf word vectors in the similar way. HitAnomaly [10] initialized their network with the weights in BERT [11], a pre-trained language representation model, to obtain the semantic meaning. These methods all show better accuracy over methods with simple ID sequences, but some are still relying on different log parsers to acquire the text on log templates. In addition, some are performing supervised classification instead of anomaly detection that can easily memorize the ground truth labels impractically. Instead of utilizing log parser, other studies [12, 13, 14, 15] simplify the processing with tokenizers, and input tokens to their detection models directly. These studies decouple their methods from log parsers and achieve comparable results. We review the detail of these methods and recognize their limitation in Section 2.

Though these studies successfully address their problems under different observation, there is still no method fulfilling diversified practical issues in system log. We summarize these issues as the following challenges.

1. **Evolution of Log Formats.** Formats in log messages can be unstable with the development of services. Log parsers could not recognize log events with updates

to existing formats. Therefore, analyzing methods not relying on specific parsers are preferred. Additionally, representation including semantics for log events can indicate the direction of these changes, and further improve the robustness of the detection. To address this issue, our method employs tokenization techniques of BERT to directly convert log messages into sequences of tokens, without relying on specific log parsers. This approach eliminates the dependency on log parsers and improves the model's robustness to unstable log data. In addition, the syntax and context of this new 'log' dialect will be learned by the further training of language model.

2. **Exceptional Parameter Values.** Parameter values in system log are the variable part in program statements, which produce the log events. They can be a size of a file, an IP address, an instance identifier, etc. Exceptional parameter values usually indicate the occurrence of anomalies, e.g., unusual execution time spent and atypical size of a file transferred. Methods accepting inputs with parameter values can better reflect this kind of anomalies. Our proposed method addresses variable parameters by replacing these values with special tokens during a domain-specific preprocessing phase. To minimize human intervention, we only replace limited variables, such as IP. This approach prevents the tokenizer from erroneously parsing variables and provides sufficient contextual information for the language model to comprehend log messages. This approach enables the model to better capture such anomalies while simultaneously simplifying the model's input, making it easier to discern the underlying semantics of log messages.

3. **Sequential Relationship among Events.** Each log event stands for a certain operation executed by system programs. These operations are normally performed in a certain order. A session containing events executed disorderly can indicate the occurrence of an anomaly. This property requires the detection models to learn the sequential relationship in normal execution paths from these operations. To this end, by training the language model exclusively on benign logs, we ensure its proficiency in reconstructing normal log sequences. Conversely, when an anomalous log is processed, the reconstruction loss is markedly elevated, indicating the presence of an anomaly. Furthermore, the RNN-based model enables us to learn the characteristic patterns of benign log event sequences, facilitating the identification of anomalies based on both event order and contextual information.

To overcome the challenges, we define the anomalies with their source into ones in event level and the others in session level. In our method, Bert-based One-class classification with explicit Reconstruction Gate (BORG) integrates the pre-trained transformer [16] into anomaly detection with the help of recurrent based neural networks. In event level, we train a transformer encoder with the cloze [17] objective in the manner of the pre-training task in BERT [11] to recognize normal log events as a reconstruction model. In session level, we use the reconstruction model as an explicit gate to control the outputs of a pre-trained transformer encoder, and utilize a RNN-based network with the one-class classification objective proposed in Ruff's work [18] to shape the normal behavior in log sessions. Eventually, we can identify the anomalies in both levels by our definition. We evaluate the concept with two contrasting data sets in our experiments.

The contributions of our work are the following.

- We design an anomaly detection network based on language representation models that can construct a normal model comprehending semantics and sequential relationships for system log.
- Our method provides state-of-the-art performance with other anomaly detection model working on the challenging data set.
- We state insights in our experiment results by analyzing sequential properties in the data sets for the evaluation.

The rest of the paper is organized as follows. In Section 2, we review the background of NLP and works adopting anomaly detection on system log. Section 3 are the detailed design of our method and adopted models. Section 4 covers implementation details, evaluation in data sets, and findings in our experiments. In Section 5, we discuss the common issues we met in our experiments and insights by our observation. At last, Section 6 concludes this research and states possible future works.

2. Related Work

2.1 Background: Natural Language Processing

In Natural Language Processing, tokenization is the procedure first performed to separate a sentence into words and even subwords that are contained in a limited set. WordPiece [19] is optimized to select a number of subwords to minimize the number of tokens in segmented corpus. A sentence *"I like tacos."* can be separated into subwords *["I", "like", "ta", "##cos", "."]* by this tokenizer. Resulting tokens are then represented by identifiers in the vocabulary. And, identifiers can be mapped to vectors by an embedding layer in language representation models. We adopt WordPiece as our tokenizer for its adaptability to rare words in system log and the compatibility to broad pre-trained models.

For language representation models, they transform tokens to vectors containing semantic meaning. The way learning word embeddings before training on a task-specific objective is proposed [20] due to the diversity of tasks in NLP and limited labels in language data. The objectives in pre-training procedure are usually self-supervised. They make use of the properties existing in language itself to create ground truth labels. Cloze [17] is the one of the properties in language that was adopted in famous language representation model, BERT [11], as the objective of its masked language model. After pre-training, embeddings can be either independent or dependent on the context. Models such as Word2vec [21] and FastText [22] learn context-independent embedding with objectives similar to cloze in deep neural networks. Permanent embedding vectors are stored for each token after their training. People can obtain the same vectors for tokens, which are independent from their positions and other tokens in the samples which they belong to. With the development of neural networks for sequential data, such as LSTM [23], GRU [24], and Transformer [16], context-dependent word vectors can be derived by leveraging their structures. A token in different context can have different vectors. ELMo

Table 1. A comparison with other works based on stated challenges

	Change Log Format		**Parameter Value**	**Sequence Relation**	**Anomaly Detection**	
	Log Parser	Semantic				
DeepLog [5]	Spell	No	**Aware**	**Yes**	**Yes**	
Tiered LSTM [12, 13]	**N/A**	No	-	**Yes**	**Yes**	
LogAnomaly [8]	FT-Tree	**Yes**	-	**Yes**	**Yes**	
LogRobust [9]	Drain	**Yes**	-	**Yes**	No	
HitAnomaly [10]	Drain	**Yes**	**Aware**	**Yes**	No	
Logsy [14]	**N/A**	**Yes**	-	No	**Yes**	
LogBERT [6]	Drain	No	-	**Yes**	**Yes**	
OC4Seq [7]	Drain	No	-	**Yes**	**Yes**	
BORG	**N/A**	**Yes**	**Aware**	**Yes**	**Yes**	

[25] learns a linear combination from all internal states of a bidirectional LSTM network pre-trained on LM task. BERT [11] pre-trains an encoder-only transformer on masked language model and next sentence prediction. All of these designs can output representation vectors for input tokens, but they differ from the way taking the context into account. In our work, we utilize cloze task in BERT to be our reconstruction model for benign log events. And, a pre-trained transformer encoder is adopted to learn the semantic in normal messages.

2.2 Related Work: Log Analysis

In analyzing system log, a method can be generally divided into two phases: representation and detection. To represent log data, some works require the help of log parser such as Spell [26] and Drain [27]. Other works directly convert a log message to a sequence of tokens with tokenizer, such as NLTK [28] and WordPiece [19]. Still others use both of the two to further transform messages into more structured formats. For detection, most of the works perform anomaly detection with different algorithms, including principle component analysis (PCA), language modeling, and deep one-class classification [18]. Some others conduct classification on this task. Methods that perform classification usually have better prediction accuracy, but does not address the characteristics of the system log. We review following works from the perspective of these two phases.

Vaarrandi [29] proposes SLCT, a density-based clustering algorithm that considers the positions of words to each log message to identify frequent patterns. Samples out of patterns can be then considered anomalies. Xu et al. [2] perform PCA on count vectors, which they first obtain log templates by parsing system source code, and describe each session with the counts to each template. Similarly, Lou et al. [3] mine program invariants with linear relationships from the count vectors. New samples break the invariants are then identified as anomalies. To take in sequential association among logs, Zhang et al. [4] train a stacked LSTM as a binary classifier to detect malfunctions in web and mailer servers. DeepLog [5] constructs a bidirectional LSTM network to predict the next template identifier in the manner of language model. An anomaly can be detected if the corresponding template had a probability under a threshold. LogBert [6] utilizes the structure of BERT [11] with modified pre-training objectives, and treats events in a

session as tokens in a sentence. It detects anomalies by the number of masked events that are wrongly predicted. Still in the manner of log parsing, OC4Seq [7] utilizes RNN-based neural networks with the objective in Ruff's work [18] to learn the normal event sequence in multiple scales. For the purpose of handling unstable logs, LogRobust [9] leverages the word vectors provided by FastText [22] to represent semantic information in log messages instead of simple identifiers. An attention-based Bi-LSTM is then applied as a classifier to identify anomalies. In HitAnomaly [10], Huang et al. use a combination of transformer encoders with parameters initialized by BERT [11]. Both of the text in log templates and parameter values are represented by corresponding transformer encoders. Anomalies are recognized by their attention-based classifier in the last layer. Though both of LogRobust and HitAnomaly detect anomalies based on classification, they still present their independence from log parsers. Nedelkoski et al. [14] also proposes a classification-based method that utilizes a transformer to learn the normal representation of good log messages. Their anomalous samples can be derived from auxiliary log messages from other systems instead of ground truth labels in the same system. But, the method is only applied to one single log line at a time. It does not consider the sequential relationship between lines.

2.3 Comparison

We observe that every study successfully address some of the issues. We summarize the methods and the issues they have addressed in Table 1. These issues are based on the challenges we point out in Section 1. Many works [5, 8, 9, 10, 6] adopt a log parser as an important role in data representation. There are few studies [5, 10] introducing a structure for parameter values. Some works [14, 15] are only considering single log message instead of a session. To the best of our knowledge, we are the first study to address all these issues.

3. Bert-based One-class classifier with explicit Reconstruction Gate

3.1 Overview

To learn the normal behavior of system log in better granularity, we define the samples in system log as groups of consecutive log events that we call log sessions. We can categorize our anomalous samples into two levels. One includes anomalous sessions that can be identified by a single event, e.g., sessions containing alert events and sessions containing events with exceptional parameter values. The other considers the sequential relationship of the events in a session. They can be only determined in a log session by some properties such as an unexpected event order. We propose the Bert-based One-class classification model with explicit Reconstruction Gate (BORG). It includes the event reconstruction model (Section 3.3) and BERT-based one class classification model for sessions (Section 3.4). We present the overview of the proposed model in Fig. 1.

Principles of design: The Rec-Gate is designed to learn the semantic structure of normal log events and serve as a "gate" to control the output of Gated-BERT. Rec-Gate is trained using the same encoder architecture and masked language modeling objective as BERT, enabling it to understand the sentence structure underlying log events. Dur-

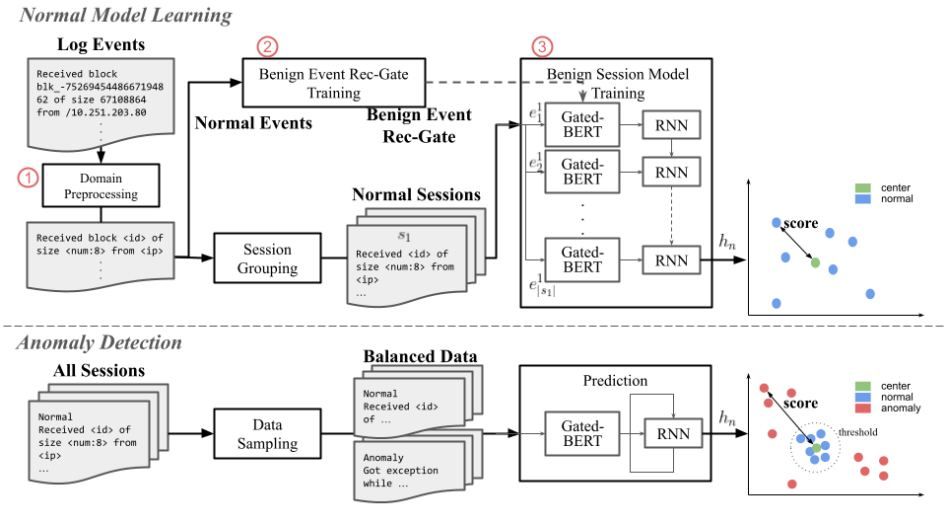

Fig. 1. An overview to the proposed model.

ing training, Rec-Gate attempts to predict masked tokens, learning the vocabulary and syntactic patterns of normal log events. In the testing phase, for normal log events, the prediction error (cross-entropy loss) of Rec-Gate will be lower; for anomalous or rare log events, the prediction error will be higher. This error value is used as a destructive scalar to adjust the output of Gated-BERT, enabling the subsequent RNN detection model to better identify anomalies. This design allows Gated-BERT to produce output vectors that incorporate event reconstruction information, assisting the RNN detection model in more comprehensively learning the normal behavior of log event sequences and identifying anomalies. In this case, anomaly in event can be captured by Rec-Gate, and the anomaly in event sequences can be captured by RNN and the one-class classifier. In practical applications, attackers may employ novel attack techniques, generating a single abnormal event and/or a sequence of events (which its pattern are considered as abnormal). With Gated-BERT, RNN, and one-class classifier, we can capture a more comprehensive set of log event patterns, improving the model's generalization ability to unknown attacks.

First, before our training, we process system log with an intuitive preprocessing technique to replace certain strings that are highly related to the domain of the system with special tokens. Secondly, with the normal events as our input samples, we train the event reconstruction model to learn the normal behavior in the level of individual. The architecture of this model is the same with the encoder in the transformer [16]. The objective is to learn the cloze task [17], which is adopted as a pre-training objective in BERT [11]. For our BERT-based one-class classification model, it stacks a recurrent neural network on a pre-trained transformer encoder. The trained event reconstruction model is then added into the structure as an explicit gate in the way explained in Section 3.4. The objective of our one-class classification model is to minimize the distance of the last hidden state to a center point. The concept is proposed by Ruff et al. [18] as an extension to Support

```
An example of single log event with <ID> = blk_-6867873931012347356:
 - Received block blk_-6867873931012347356 of size 67108864 from /10.251.39.64
After our preprocessing:
 - Received block <ID> of size <NUM:8> from <IP>
After BERT tokenizer:
 - received, block, <id>, of, size, <num:8>, from, <ip>

An example of log event sequence, which is labeled as anomaly:
 - Receiving block <ID> src: /<IP> dest: /<IP>
 - BLOCK* NameSystem.addStoredBlock: blockMap updated:
     <IP> is added to <ID> size <NUM:8>
 - PacketResponder <NUM:1> for block <ID> terminating
 - Received block <ID> of size <NUM:8> from /<IP>
 - PacketResponder <NUM:1> for block <ID> terminating
 - Received block <ID> of size <NUM:8> from /<IP>
 - PacketResponder <NUM:1> for block <ID> terminating
 - Received block <ID> of size <NUM:8> from /<IP>
 - BLOCK* NameSystem.addStoredBlock: blockMap updated:
     <IP> is added to <ID> size <NUM:8>
 - <IP> Served block <ID> to /<IP>
 - <IP>:Got exception while serving <ID> to /<IP>:
 - <IP> Served block <ID> to /<IP>
 - <IP>:Got exception while serving <ID> to /<IP>:
 - <IP> Served block <ID> to /<IP>
 - <IP>:Got exception while serving <ID> to /<IP>:
 - BLOCK* NameSystem.delete: <ID> is added to invalidSet of <IP>
 - Deleting block <ID> file /mnt/hadoop/dfs/data/current/subdir27/<ID>
 - Deleting block <ID> file /mnt/hadoop/dfs/data/current/subdir41/<ID>
 - Deleting block <ID> file /mnt/hadoop/dfs/data/current/subdir13/<ID>
 - Unexpected error trying to delete block <ID>. BlockInfo not found in volumeMap
```

Fig. 2. Examples of Events, Event Sequences, and Domain Pre-processing.

Vector Data Description (SVDD) [30] in deep learning. At last, we also composite the loss from event Rec-Gate with the distance to obtain the malicious scores. Log sessions with higher malicious scores after the transformation in BORG are considered anomalies.

3.2 Domain Preprocessing

Domain preprocessing is the way we adopt to teach the language model to understand domain words in system log corpus. Domain words include the physical machine address, which can be a long sequence of hexadecimal numbers, IP address, instance identifier, etc. Operators of a system can use this method to replace trivial information and to enhance the performance of the following detection. The number values can be also transformed into tokens representing ranges they belong to. For example, the IP address *10.251.39.64* in Fig. 2 is replaced with the token *<IP>*. Without abstracting the text, the WordPiece [19] tokenizer we adopted can convert each . as a token resulting a long token sequence for a simple IP address. However, with our prepressing method and tokenizer, IP is treated as a special token.This preprocessing is also intuitive for system operators reading the log messages. The specific information, such as hash values and IP address, is only meaningful to the developer who wrote the log statement. However, for operators, the information required to recognize anomalies is the existence of a certain concept and corresponding positions. Similarly, if we apply the method to parameter values, the numbers for an operator can be either in the range often seen or in an unexpected one. Figure 2 also shows an (anomaly) example of log event sequence, our language model

will learn the normal pattern of log sequence from the normal training data, and try to find out any deviation identified with the help of Rec-Gate,, RNN and one-class anomaly detection model.

3.3 Event Reconstruction Model

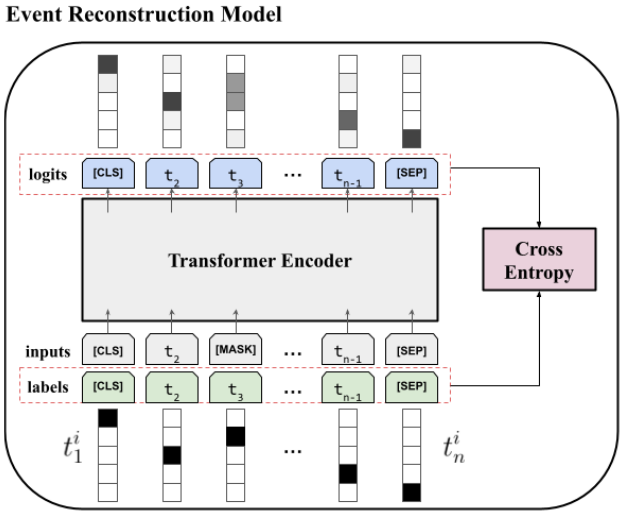

Fig. 3. An illustration to the event reconstruction model.

The event reconstruction model is a transformer encoder [16] learning cloze [17] task by using the training system log. As an anomaly detection system, if a token used in a sample in the testing dataset is unusual or rarely-seen, the event reconstruction model will output a larger value (cross-entropy loss) to point it out. The information will be consider for the later model. We illustrate the structure in Fig. 3. It acts as a Language Model (LM) to understand the sentence structure behind log events. Conventional language models can have different learning objectives. For example, a n-gram language model generally maximizes the probability to predict each word in a sentence by their previous n words, i.e.

$$LM(n-gram) = \prod_{i=1}^{n} p(w_i|w_{i-1}, w_{i-2}, ..., w_{i-n}),$$

In the transformer encoder, we follow the same way with BERT [11] to mask 15% of tokens with the special token, *[MASK]*. We have the model to predict the true sentence with the masked one. Instead of previous n tokens, the transformer encoder is given the information of all other tokens. In system log, we are given normal events is a system with the size $|E_{nl}|$, i.e., $E_{nl} = \{e_1, e_2, ..., e_{|E_{nl}|}\}$, the WordPiece [19] tokenizer first segments the text in each event into tokens. After masking, each token is then mapped to a vector by its index in the vocabulary. We get the token vectors, $x^i = \{x_1^i, x_2^i, ..., x_{|X^i|}^i\}$, where $i \in \{1, ..., |E_{nl}|\}$. Since the transformer encoder [16] fully utilizes the self-attention

mechanism,

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V,$$

we get the contextualized embedding for each token. That is, matrices Q,K,V are computed by multiplying the input matrix x_i by the weights W_Q,W_K,W_V. Self-attention can use dot product of the query feature matrix Q and key feature matrix K to calculate the similarity among token vectors, which is the attention score after a scalar $\frac{1}{\sqrt{d_k}}$ and the softmax function. With attention scores, the contextualized representations for each token can be calculated by the multiplication with value feature matrix V, and they can be recognized as the weighted average of each tokens that takes in all the other tokens. In a complete transformer, the multi-head attention is applied instead, which is a variation of the attention above. It uses more than one set of feature matrices, and concatenate the outputs. A detailed explanation is well presented by Lippe's work [1]. And, the structure of the transformer encoder enables itself to act as a language model considering all other tokens in a sentence.

In the end of the transformer encoder, a classification layer with weight $W_{cls} \in \mathbb{R}^{Vocab \times H}$ is appended to calculate the probability distribution for each token, where $Vocab$ and H are the vocabulary size and hidden dimension of the transformer. The logit values for the distribution can be

$$Logits_i = W_{cls} \cdot \phi_T(x_i; W^*) + bias, \tag{1}$$

where $Logits_i \in \mathbb{R}^{Vocab \times |e_i|}$, and $\phi_T(x_i; W^*)$ is the output embeddings from masked token vectors in i-th event. With $Logits_i$ and original token vectors, $t_i = \{t_{i,1}, t_{i,2}, ..., t_{i,|t_i|}\}$, we can compute the event loss ${l_e}^i$ by cross entropy loss, i.e.

$${l_e}^i = \frac{\sum_{j=1}^{|t^i|} {l_e}^i_j}{|t^i|}, \tag{2}$$

$${l_e}^i_j = -\sum_{c=1}^{Vocab} t^i_{j,c} \log \hat{y}_c, \tag{3}$$

$$\hat{y}_c = \frac{exp(Logits^i_{j,c})}{\sum_{k=1}^{Vocab} exp(Logits^i_{j,k})} \tag{4}$$

3.4 BERT-based One-class Classification Model

To learn the relationships between log events in a sequence and incorporate anomaly information into embedding vectors, the BERT-based One-class Classification Model includes two components: 1) a gated-BERT for outputting the embedding of event with the degree of anomaly in event reconstruction and 2) a RNN model to process the output

[1] https://pytorch-lightning.readthedocs.io/en/latest/notebooks/course_UvA-DL/05-transformers-and-MH-attention.html

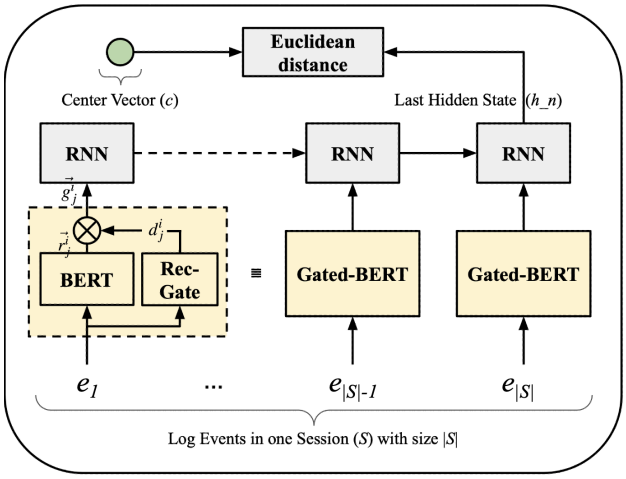

Fig. 4. An illustration to the BERT-based one-class classification model.

sequence of Gated-BERT and capture long-range dependencies and patterns in logs, and identifying anomalies based on the sequence and semantic information of events.

Figure 4 illustrates the two components. The gated-BERT adopts the structure and pre-trained weights of BERT-mini [31]. This helps embeddings for log events containing the semantic information behind the text. In addition, since we expect our normal model can also reflect the event-level abnormality, we utilize Rec-Gate (an event reconstruction model) that we have trained in the previous section to adjust the output embeddings from BERT for the RNN-based detection network. For the events that the Rec-Gate has learned before and recognized (i.e., with less loss value), the Gated-BERT outputs the original embeddings. In contrast, for unseen events, the Rec-Gate tends to output a larger value so that the corresponding a vector d can disrupt the embeddings from BERT. The disrupted embeddings would contain less information, and be considered the noise in the later RNN-based detection model. Figure. 4 presents the event reconstruction model as a **Rec-Gate**. ${l_e}^i_j$ is denoted as the loss of j-th log event in i-th session from the Rec-Gate. In our design, this value is set to the disruptive scalar d^i_j to indicating how good the event reconstruction model performs. If a rare-seen log event is observed (which is possibly an anomaly event), the loss (i.e., the cross-entropy of the masked token prediction) of the event reconstruction model would be larger for analysis. d^i_j value will be outputted by Rec-Gate and passed to RNN for further use.

The information of embedding $\vec{r^i_j}$ is then concatenated with the disruptive scalar d^i_j. We anticipate that the gates in RNN-like structure (such as LSTM and GRU) could work with d^i_j to tweak r^i_j for the one-class classfier.

$$\vec{g^i_j} = \vec{r^i_j} \oplus d^i_j, \tag{5}$$

and we can get the output $\vec{g^i_j}$ that includes an additional reconstruction information.

Here, we plot the distribution of d^i_j value using one of our experiment data set (BGL)

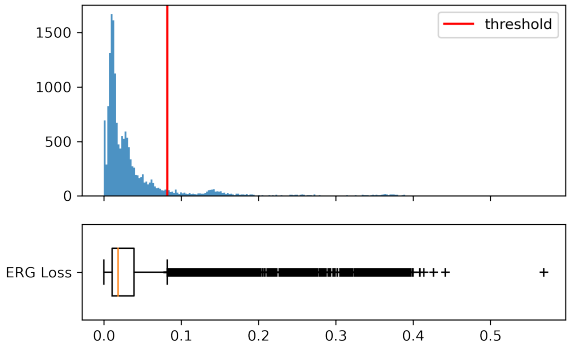

Fig. 5. An illustration of the outliers (detected by Rec-Gate with threshold τ) in BGL data set.

in Fig. 5. We can see that most of the loss value of Rec-Gate is larger than outlier threshold τ, which is $Q3 + 1.5IQR$, where

$$InterQuartileRange(IQR) = Q3 - Q1$$

For our detection network, it is a RNN-based model to cope with variable-length events in a log session, which is the nature of system log. We avoid to use strategies like truncation and padding in session level, since we observe that an anomaly can occur at the last few events in a session. Also, in anomaly detection, we should assume we do not know how long an anomalous log session can be. That makes it hard to determine the max length in such strategies.

At the top of Fig. 4, we adopt the One-Class Deep SVDD [18] as the objective to train our network. The one-class objective is to transform samples into a hypersphere with a center point c by updating a deep neural network, such that the distance between all samples and the center is minimized. Instead of deep neural network, we utilize the RNN-based network as our interval transformation that updates the same set of weights in a sequence of inputs, which is a similar idea with OC4Seq [7]. That is, given n sessions $\{s_1, s_2, ..., s_n\}$,

$$\min_W \frac{1}{n} \sum_{i=1}^{n} \left\| h_{|s_i|} - c \right\|^2, \tag{6}$$

we calculate the Euclidean distance between the last hidden state $h_{|s_i|}$ and initialized center c. And, $h_{|s_i|}$ is subject to approaching c after $\phi(;W)$'s transformation, i.e.,

$$h_{|s_i|} = \sum_{t=1}^{|s_i|} \phi(g^i_{t-1}; W). \tag{7}$$

$\phi(;W)$ can be any RNN-based network, e.g., GRU and LSTM, and the input timestamps are from the outputs of the gated-BERT g^i. Our initialized center c is calculated by the whole model only with initialized weights before training.

Table 2. The statistics in HDFS and BGL data set

	HDFS			BGL		
	Normal	Anomaly	Overall	Normal	Anomaly	Overall
event	-	-	11,175,629	4,399,503	348,460	4,747,963
session (sess)	558,223	16,838	575,061	166,095	15,140	181,235
unique event	-	-	239	17,742	2,306	20,048
tokens per event	-	-	15.1	14.8	14.1	14.8
events per sess	19.5	17.1	19.4	61.8	66.3	62.2

3.5 Composite Malicious Score

Generally, the distance of a new sample to the trained center c' is the score to evaluate its abnormality. We further compose the loss in event reconstruction model with the distance to make abnormality in event level more prominent. The malicious score we use to evaluate our method in the compositing way can be written as

$$\left\|h_{|s_i|} - c'\right\|^2 + \beta {l_s}^i, \tag{8}$$

where β is a hyper-parameter to determine how much the reconstruction loss (Rec-Loss) can contribute to the malicious score. And, the Rec-Loss l_s for i-th session is the average of the Rec-Loss l_e for each event in the session. The Rec-Loss of i-th session is defined as Eq. 9.

$${l_s}^i = max(\frac{1}{|s_i|}\sum_{j=1}^{|s_i|} {l_e}^i_j - \tau, 0), \tag{9}$$

where τ is determined by the data point of $Q3 + 1.5IQR$ like Fig. 5 in our work.

The design of Eq. 9 is based on an observation that anomaly events are usually observed continuously in a system. Therefore, if the average of the Rec-Loss is larger than τ, it strongly suggests that this session has above-average unseen or rare-seen logs.

4. Experiment

4.1 Data set

We use logs in Hadoop Distributed File System (HDFS) and BlueGene/L Supercomputer System (BGL) to evaluate our concepts. These data sets are accessible in Loghub [32] for research purposes. It provides a collection of system logs containing some data used in previous studies.

4.1.1 Hadoop Distributed File System

HDFS data set was first introduced by Xu et al. [2]. This data set contains 11,175,629 lines of logs collected from a Hadoop cluster with over 200 nodes. Every log event has corresponding block identifier, and there are 575,061 distinct identifiers. Xu et al. manually labeled the anomalous data blocks with the help of experts in Hadoop. We organize the contents and form log sessions by these identifiers. Anomalous sessions are directly determined by the labels to the block IDs.

4.1.2 BlueGene/L Supercomputer System

Blue Gene/L is a supercomputer with 131,072 processors and 32,768 GB memory designed by IBM. Oliner et al. [33] introduced the data set with the supercomputer provided by Lawrence Livermore National Labs. The data set consists of 4,747,963 messages including a total of 348,460 alerts caused by software and hardware failure at all levels. Anomalies in this data set are labeled on each event, and events do not contain specific identifiers. Therefore, we organize with a time sliding window to assemble log sessions. For any group of consecutive events containing at least one alert, we label the group as an abnormal log session. In our implementation, we also introduce a hyper-parameter M to decide the maximum number of events that a session can include. If a session included more than M events, we evenly separate the session s_i into $\lceil \frac{|s_i|}{M} \rceil$ sessions. In following experiments, we choose *10 seconds*, *4 seconds*, and *100* for the parameters of *window size*, *stride*, and M respectively.

4.2 Implementation

The domian pre-processing is implemented by regular expression library in Python 3.9. And, we construct our model with PyTorch 1.10.0 in the same environment. For our training samples in BERT-based one-class classification model, we randomly select 5% of normal log sessions in HDFS data set and 10% in BGL. To evaluate over unbalanced samples in these two data sets, we collect all anomalous sessions, and sample the same number of normal sessions from the sessions excluding training samples, to form a balanced testing samples for each data set for the better understanding of the performance of different models. The training and evaluating to the model are performed on a single NVIDIA GeForce GTX 1080 Ti graphics card.

4.3 Evaluation

4.3.1 Model Evaluation

Since we have a rich combinations in our architecture, Table 3 shows our evaluation results on HDFS and BGL data sets over different parameters and rnn-based model. In our embedding layer, we adopt BERT [31] with the mini size that has 4 encoder layers and 256 hidden dimensions. For the internal transformation in one-class objective, we experiment on each of the common RNN-based networks with 2 layers and 64 hidden units. At last, we use AUC, f1, precision and recall to evaluate the performance of the proposed system. We intentionally remove the Rec-Gate structure and its loss to perform ablation study (while only using Euclidean distance (L2 norm) to optimize the one-class classifier without using any information from the Rec-Gate). Our proposed system is marked as 'L2+RecLoss' which optimizes the Rec-Gate and the downstream classifier.

The composite malicious score shows significant performance improvement on BGL data set. This conforms with the finding in Fig. 6 that most of anomalous events shows higher loss values identified by Rec-Gate. And, the result can be more significant when we only count the unique log events. We observe that the performance improvement in HDFS data set is trivial. That also conforms with the distribution in Fig. 7. Events appearing in anomalous sessions are mostly similar to the ones in normal sessions. Though the two distribution can be still distinguished in unique events, the small portion which they

Table 3. The BORG performance in different combinations.

RNN Network		HDFS			
		AUC	f1 score	precision	recall
RNN(tanh)	L2 norm	0.900	0.892	0.983	0.816
RNN(relu)	L2 norm	0.864	0.840	0.987	0.738
GRU	L2 norm	0.807	0.807	0.966	0.712
LSTM	L2 norm	0.939	0.932	0.982	0.887
RNN(tanh)	L2+RecLoss	0.903	0.894	0.981	0.821
RNN(relu)	L2+RecLoss	0.866	0.840	**0.987**	0.739
GRU	L2+RecLoss	0.810	0.808	0.966	0.714
LSTM	L2+RecLoss	**0.945**	**0.936**	0.977	**0.898**
		BGL			
		AUC	f1 score	precision	recall
RNN(tanh)	L2 norm	0.638	0.719	0.722	0.729
RNN(relu)	L2 norm	0.537	0.750	0.644	**0.921**
GRU	L2 norm	0.755	0.784	0.741	0.835
LSTM	L2 norm	0.627	0.659	0.757	0.702
RNN(tanh)	L2+RecLoss	0.856	0.797	0.817	0.783
RNN(relu)	L2+RecLoss	0.848	0.811	0.787	0.844
GRU	L2+RecLoss	0.821	0.794	0.736	0.870
LSTM	L2+RecLoss	**0.903**	**0.823**	**0.914**	0.749

account for is not enough to make the difference significant when considering the entire data set.

4.3.2 Performance Comparison

We present the comparison of our method with other works in anomaly detection. The evaluation in HDFS and BGL are respectively displayed in Fig. 8 and Fig. 9. In HDFS data set, most of the baselines have f1 scores over 0.90. We show comparable results with the state-of-the-art method. However, these methods all rely on the log parsers that converts log events into numerical identifiers. We conduct a series of analyzing methods on the sequence properties in the next section to show our insights. In BGL data set, our method outperforms the existing methods in anomaly detection. BGL data set is a more challenging data set with its statistics properties showed in Table 2. After we replace domain words in log events with special tokens, the HDFS data set contains only 239 kinds of messages in 11,175,629 lines. In the other hand, the BGL data set contains over 20,000 unique messages remaining after our preprocessing. This property largely affects the performance of a log parser. Therefore, methods relying on log parsers usually show lower scores over this data set. In addition, BGL has a much more number of events in a session in average. That also makes the sequential relationship harder to learn in anomaly detection models.

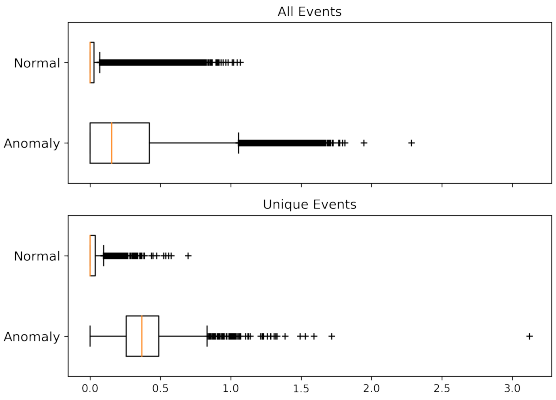

Fig. 6. The distribution of RecLoss in BGL data set.

4.3.3 Performance Analysis

To better interpret our experiment results, we analyze the sequence properties in log sessions. For the ease of analyzing sequences, we transform log events into template identifiers with the help of the log parser Drain [27]. And, we design a few criteria to determine if there is any identical sequences existing in both of data sets. The first method, *identical order I*, is directly performing unique function on ID sequences. The second approach, *identical order II*, is to perform unique function after removing redundant IDs. For example, a sequence *[1, 2, 2, 5]* can be converted to *[1, 2, 5]*. It is based on the fact that some events appear a few times consecutively in both of the data sets. And, it is also similar with the way human reading a sequence. We usually do not put much attention on the identical items in a row, but put attention on the changes in the sequence. Thirdly, we use count vectors to stand for a sequence in *identical counts* method, i.e., a sequence *[1, 2, 2, 5]* can be converted to *(1, 2, 0, 0, 1)*. Eventually, we check the number of unique sequences with these identical definitions under each data set. And, these unique sequences are later called *patterns*.

We present the statistics of HDFS and BGL data sets in Table 2 and Table 4. The following findings help us to prove the capability of the proposed system confronting the challenging data set.

1. There actually exists normal behavior and deviation in anomalies. We found the number of unique event sequence is much lower than the total amount of normal sessions. There are only 14,259 unique sequences in HDFS data set that contains 558,223 normal sessions. For the deviation, the abnormal samples have 4,126 unique sequences in 16,838 anomalous sessions. The ratios of remaining unique patterns across all identification methods in abnormal samples, are all much higher than the ratios in normal samples.

2. The normal behavior in BGL is more complex than the behavior in HDFS. For

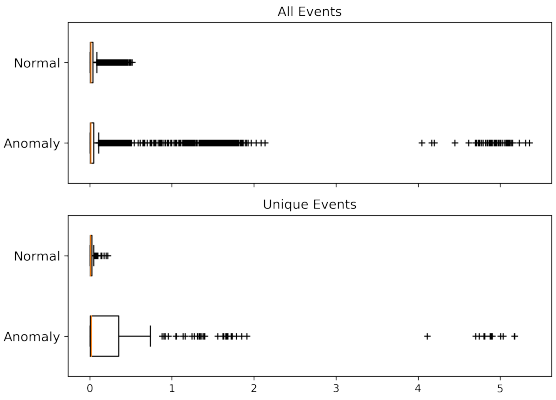

Fig. 7. The distribution of RecLoss in HDFS data set.

normal sessions in BGL data set, the ratios of unique patterns to total amount are all much larger than ratios in HDFS. Table 2 also shows the BGL data set has longer lengths for sequences in event and session level. In other words, we can say that BGL data set is harder to tell if a session is anomalous only by the order of log events.

3. There exists anomalies in event level. Since we are curious about if there are patterns both appearing in normal and abnormal sessions, we calculate the number by the intersection of column one and two. By our results, the answer is affirmative. We call these patterns *baffling patterns* because they can confuse models that only comprehend sequential relationship of log events. It shows such anomalies can only be recognized by models considering their messages.

4. Baffling patterns affect the accuracy more in BGL data set. In HDFS, the portion the baffling patterns takes in the testing data set is not significant enough to affect the performance of a model. That is, methods considering anomalies in event level such as exceptional parameter values would hardly have improvement over samples in HDFS. In BGL, the result is on the contrary. Whichever identification method we adopt, baffling patterns comprise over 20% of testing data set. Methods not considering event-level anomalies can have significant performance decrease in BGL samples.

5. The bottom of Fig. 2 shows an anomaly example of HDFS logs. This specific pattern of event logs is considered as anomaly by the BORG. We find that those events (for example, “Receiving block”, “BLOCK* NameSystem.addStoredBloc”, “PacketResponder”, “Served block”, etc) can exist in the system solely without reporting anomaly, but as a whole, our system identify them as an anomaly. We anticipate that our language model (with MLM) and the RNN can indeed learn the normal pattern.

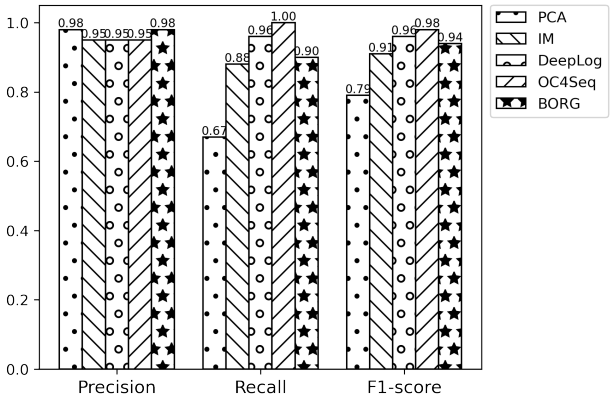

Fig. 8. Evaluation on HDFS data set.

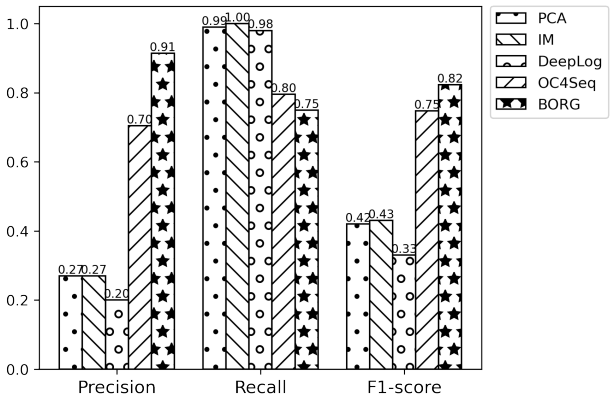

Fig. 9. Evaluation on BGL data set.

By the findings mentioned above, we verify our claiming that anomalies in system log can be generally categorized by event level and session level. The data sets that we evaluate our method indeed have contrasting properties by our statistics results. With the comparable results of these two data sets in Fig. 8 and Fig. 9, the way we utilize language model to reconstruct log events is confirmed to enhance the robustness of our anomaly detection model in system log.

5. Discussion

In this section, we state our limitation and possible improvement to our work. The findings in the procedure of our experiments and analyzing are also mentioned.

Our method integrates reconstruction based anomaly detection and one-class classification. However, the disruption we generate for the reconstruction is still based on a

HDFS					
Identification Methods	**# of patterns in**		**# of baffling patterns**	**# of baffling patterns in**	
	normal session	**abnormal session**		**testing[a] dataset**	**entire dataset**
Identification order I	14,259 (2.6%)	4,126 (24.5%)	10	0.073%	0.008%
Identification order II[b]	5,250 (0.9%)	3,097 (18.4%)	50	1.188%	0.103%
Identification counts	212 (0.04%)	394 (2.3%)	9	0.968%	0.096%
BGL					
Identification order I	31,508 (19%)	4,110 (27%)	129	22.7%	8.23%
Identification order II[b]	11,999 (7.2%)	1,642 (11%)	67	45.8%	34.7%
Identification counts	16,632 (10%)	3,174 (21%)	159	23.3%	8.39%

Table 4. The sequence statistics of HDFS and BGL data sets

random selection. Though the masking technique works fine in language representing, for anomaly detection on system log, the unstable interference would cause a poor learning in the one-class objective. Stabilizing the reconstruction process is the main task to accomplish in our future work. In addition, the method we adopt to initialize center point is passing training data through the entire model including the pre-trained transformer encoder. We observe that difference initialization strategies can affect the performance in one-class classification. Though we have experimented the influence of different activation function in RNN-based networks, we did not observe the hypershpere collapsing mentioned in original paper [18] that utilizes the deep neural network.

6. Conclusion

Our work designs the Bert-based One-class classification with an explicit Reconstruction Gate (BORG) to learn the normal model for system log in the event level and session level. We state the challenges in log analysis, including changing formats, parameter values, and sequential relationship for anomaly detection. To cope with these practical issues, we utilize the tokenizer and language representation models in NLP to eliminate the reliance on log parsers, and to learn the semantic information. Also, in the manner of anomaly detection, we integrate RNN-based networks with the one-class objective for sequential relationship instead of a supervised classifier. We design the Gated-BERT and Rec-Loss to reflect the event-level abnormality in our overall normal model for log sessions. With two common data sets possessing contrasting properties in our analysis, we demonstrate comparable results in our experiments and outperforms state-of-the-art methods in BGL data set. The sequential statistics also provide a deeper understanding to the anomalies of system log in different levels, and further verify our concepts.

ACKNOWLEDGMENT

This work was supported in part by the National Science and Technology Council, Taiwan under grant NSTC 113-2634-F-004-001-MBK and NSTC 113-2221-E-004-014-MY3.

REFERENCES

1. K. Yamanishi and Y. Maruyama, "Dynamic syslog mining for network failure monitoring," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 499–508.
2. W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.
3. J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.
4. K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, "Automated it system failure prediction: A deep learning approach," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 1291–1300.
5. M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.
6. H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.
7. Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen, and J. Tang, "Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 3726–3734.
8. W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, Vol. 19, no. 7, 2019, pp. 4739–4745.
9. X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.
10. S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, Vol. 17, no. 4, 2020, pp. 2064–2076.
11. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
12. A. R. Tuor, R. Baerwolf, N. Knowles, B. Hutchinson, N. Nichols, and R. Jasper, "Recurrent neural network language models for open vocabulary event-level cyber

anomaly detection," in *Workshops at the thirty-second AAAI conference on artificial intelligence*, 2018.
13. A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, 2018, pp. 1–8.
14. S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1196–1201.
15. Y. Lee, J. Kim, and P. Kang, "Lanobert: System log anomaly detection based on bert masked language model," *arXiv preprint arXiv:2111.09564*, 2021.
16. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, Vol. 30, 2017.
17. W. L. Taylor, ""cloze procedure": A new tool for measuring readability," *Journalism quarterly*, Vol. 30, no. 4, 1953, pp. 415–433.
18. L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*. PMLR, 2018, pp. 4393–4402.
19. Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.
20. R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.
21. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
22. A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.
23. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, Vol. 9, no. 8, 1997, pp. 1735–1780.
24. K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.
25. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237. [Online]. Available: https://aclanthology.org/N18-1202
26. M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.
27. P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

28. E. Loper and S. Bird, "Nltk: The natural language toolkit," *arXiv preprint cs/0205028*, 2002.
29. R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. IEEE, 2003, pp. 119–126.
30. D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, Vol. 54, no. 1, 2004, pp. 45–66.
31. I. Turc, M. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: The impact of student initialization on knowledge distillation," *CoRR*, Vol. abs/1908.08962, 2019. [Online]. Available: http://arxiv.org/abs/1908.08962
32. S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: a large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020.
33. 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 (DSN'07)*. IEEE, 2007, pp. 575–584.

Shun-Wen Hsiao received the B.S. and Ph.D. degree from the Department of Information Management, National Taiwan University, in 2004 and 2012, respectively. From 2006 to 2008, he participated in the iCAST Collaborate Research Project with the CyLab, Carnegie Mellon University. In 2012, he joined the Institute of Information Science, Academia Sinica, Taiwan, where he held a Post-Doctoral Research Fellowship. Since 2017, he has been with the Department of Management Information Systems, National Chengchi University, where he is currently an Associate Professor. His research interests are in the area of cybersecurity, malware behavior analysis, virtualization technology, and FinTech.

Chih-Chung Tseng Chih-Chung Tseng received the B.S. degree from the Department of Information Management, National Chengchi University, in 2022. His research interests include log analysis, natural language processing, and representation learning.