BERT-Log: Anomaly Detection for System Logs Based on Pre-trained Language Model

Song Chen\textsuperscript{a} and Hai Liao\textsuperscript{b}

\textsuperscript{a}School of Computer Engineering, Chengdu Technological University, Chengdu, China; \textsuperscript{b}School of Software, Sichuan Vocational College of Information Technology, Guangyuan, China

**ABSTRACT**

Logs are primary information resource for fault diagnosis and anomaly detection in large-scale computer systems, but it is hard to classify anomalies from system logs. Recent studies focus on extracting semantic information from unstructured log messages and converting it into word vectors. Therefore, LSTM approach is more suitable for time series data. Word2Vec is the up-to-date encoding method, but the order of words in sequences is not taken into account. In this article, we propose BERT-Log, which regards the log sequence as a natural language sequence, use pre-trained language model to learn the semantic representation of normal and anomalous logs, and a fully connected neural network is utilized to fine-tune the BERT model to detect abnormal. It can capture all the semantic information from log sequence including context and position. It has achieved the highest performance among all the methods on HDFS dataset, with an F1-score of 99.3%. We propose a new log feature extractor on BGL dataset to obtain log sequence by sliding window including node ID, window size and step size. BERT-Log approach detects anomalies on BGL dataset with an F1-score of 99.4%. It gives 19% performance improvement compared to LogRobust and 7% performance improvement compared to HitAnomaly.

**Introduction**

With the explosion of the number of business application on internet, building a trustworthy, stable and reliable system has become an important task. Currently any anomaly including network congestion, application breakdown, and resource allocation failure may impact millions of online users globally, so most of these systems are required to operate on a 24 × 7 basis (He et al. 2016; Hooshmand and Hosahalli 2022; Hu et al. 2022). Accurate and effective detection method can reduce system breakdown caused by the anomalies. System logs are widely used to record system states and significant events in network and service management (Lv, Luktarhan, and Chen 2021; Studiawan,
Sohel, and Payne 2021). We can debug performance issues and locate the root cause by these logs (Maeyens, Vorstermans, and Verbeke 2020; Mi et al. 2012).

Logs contain detailed information and runtime status during system operation (Liu et al. 2020; Lv, Luktarhan, and Chen 2021), and they are one of the most important data for anomaly detection. For example, an anomaly log of network traffic indicates that the traffic utilization exceeds the threshold and the system needs more network bandwidth to ensure user service. As the scale and complexity of the system increase, such as there are about 50GB logs per hour in a large-scale service system (Mi et al. 2013), it is hard to detect anomalies from system logs by traditional manual method. Recently, most research works aim at parsing critical information from logs, and then use vector encoding and deep learning techniques to classify anomalous logs automatically and accurately.

Retrieving system logs methods for anomaly detection can be usually classified into three categories: (1) Detecting anomalous logs by matching keywords or regular expression (Cherkasova et al. 2009; Yen et al. 2013). For example, operation engineer manually searches for keywords (e.g., “down,” “abort”) from logs to detect anomalies. These methods require that operation engineer must be familiar with the rule of anomalous messages. (2) Converting logs into count vectors and using deep learning algorithm to detect anomalies (He et al. 2018; Lou et al. 2010; Zhang and Sivasubramaniam 2008). These methods regard event as individuals and only counts the times of each event occurrence. It ignores the correlation between different events. (3) Extracting semantic information from log messages and converting it into word vectors (Du et al. 2017; Huang et al. 2020; Zhang et al. 2019a). These semantic vectors are trained to classify anomalous logs more effectively.

Raw log messages are unstructured, which contain many different format texts. It is hard to detect numerous anomalies based on unstructured logs. The purpose of log parsing (Du and Li 2016; He et al. 2017) is to structure logs to form group of event templates. HitAnomaly (Huang et al. 2020) is a semantic-based approach which utilizes a hierarchical transformer structure to model log templates and uses an attention mechanism as final classification model. BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model proposed by Devlin et al. (2018) of Google in 2018 which obtains new state-of-the-art results on eleven famous natural language processing (NLP) tasks. Compare with the previous hierarchical transformer, BERT contains pre-training and fine-tuning steps and has better performance on the large suite of sentence-level and token-level tasks. BERT has already been used in many fields (Do and Phan 2021; Peng, Xiao, and Yuan 2022). So it is better suited for handling semantic-based log sequences.

In this article, we propose BERT-Log which can detect log anomalies automatically based on BERT pre-trained and fine-tuned model. Firstly, according to timestamp and message content, we parse raw unstructured
logs into structured event templates by Drain (He et al. 2017) and get event templates. Then we use sliding window or session window to convert the ID of event templates into log sequence, and use BGL parsing algorithm based on node ID to process BGL dataset. Secondly, log sequence has been converted into embedding vector by pre-trained model. According to the position and segment information of token in log sequence, semantic vector is obtained after concatenation. Log encoders calculate the attention score in the sequence with multi-head attention mechanism, and it is used to describe semantic information in log sequence. Finally, we use full connection neural network to detect anomalies based on semantic log vectors. Anomaly is often influenced by the order of each word in the log sequence, so the anomalous log sequence is different from healthy sequence and we use supervised learning method to learn the semantic representation of normal logs and anomalous logs. Compared with other BERT based methods or transformer based methods, experiments results show that our proposed model can correctly represent semantic information of log sequence.

We evaluate BERT-Log method on two public log datasets including HDFS dataset (Xu et al. 2009) and BGL dataset (Oliner and Stearley 2007). For anomalous logs classification, BERT-Log approach has achieved the highest performance among all the methods on HDFS dataset, with an F1-score of 99.3%. And it detects anomalies on BGL dataset with an F1-score of 99.4%. The F1-score of 96.9% was obtained with only 1% of the dataset on HDFS dataset, and the F1-score of 98.9% was obtained with only 1% of the training ratio on BGL dataset. The result shows that BERT-Log based approach has got better accuracy and generalization ability than previous anomaly detection approaches.

The major contributions of this article are summarized as follows:

1. We propose BERT-Log, which regards the log sequence as a natural language sequence, use pre-trained language model to learn the semantic representation of normal and anomalous logs, and then a fully connected neural network is utilized to fine-tune the BERT model to detect abnormal.

2. A new log feature extractor on BGL dataset is proposed to obtain log sequence by sliding window including node ID, window size and step size. To the best of our knowledge, our work is the first to utilize node ID and time to form log sequence.

3. The method proposed in this paper achieves F1-score of 98.9% with 1% of the training ratio on BGL dataset. Compared with other related works, it has smaller parameters and stronger generalization ability.

The rest of this article is organized as follows. The related works are described in Section 2. We introduce the method of BERT-Log in Section 3.
Section 4 describes the experimental results. Finally, we conclude our work in Section 5.

**Related Works**

Logs record detailed information and runtime statues during system operation. It contains a timestamp and a log message indicating what has happened. Logs are important and primary information resource for fault diagnosis and anomaly detection in large-scale computer systems. However, since numerous raw log messages are unstructured, accurate anomaly detection and automatic log parsing are challenging. Many studies have focus on log collection, log templating, log vectorization, and classification for network and service management.

**Log Collection**

Log collection is one of the most important tasks for developers and operation engineers to monitor computer systems (Zhong, Guo, and Liu 2018; Zhu et al. 2019). There are many popular methods to receive logs from computer system or network device, such as log file (Tufek and Aktas 2021), syslog (Zhao et al. 2021), trap (Bretan 2017), snmp (Jukic, Hedi, and Sarabok 2019), program API (Ito et al. 2018).

Some open log files are generally used as raw data in the research work to detect anomalies. HDFS log file is a dataset collected from more than 200 Amazon’s EC2 nodes. BGL log file is a dataset collected from Blue-Gene/L supercomputer system at Lawrence Livermore National Labs. Openstack log file is a dataset generated on the cloud operation system. HPC log file is a dataset generated on a high performance cluster. In this article, we uses HDFS log file and BGL log file as the resources of log collection.

**Log Templating**

Logs are unstructured data, which consist of free-text information. The goal of log parsing is to convert these raw messages content into structured event templates. There are three categories of log parsers. The first category consists of clustering-based methods (e.g., IPLoM (Makanju, Zincir-Heywood, and Milios 2009), LogSig (Tang, Li, and Perng 2011)). The logs are classified into different clusters by distance. Event templates are generated from each cluster. Second category bases on heuristic-based methods (e.g., Drain (He et al. 2017), CLF (Zhang et al. 2019b)).These methods can directly extract log templates based on heuristic rules. For example, Drain uses a fixed depth parse tree to encode specially designed rules for parsing. The third category include NLP-based methods (e.g., HPM (Setia, Jyoti, and Duhan 2020), Logram (Dai et al.
Recent approaches to log analysis include LogBERT (Guo, Yuan, and Wu 2021), Random Forest (Aussel, Petetin, and Chabridon 2018)), such as N-GRAM dictionaries, random forest. It leverages NLP algorithm to achieve efficient log parsing. Compare with other methods, Drain achieves high accuracy and performance. In this article, we choose Drain as log parser.

**Log Vectorization and Classification**

Event templates are produced by log parser, and they are grouped into vectors by log vectorization. Log sequence consists of a set of event ID which represents event template. In HDFS dataset, we can form log sequence by block ID. In BGL dataset, the log sequence can be grouped by the sliding window.

Xu et al. (2009) use PCA algorithm to detect large-scale system problems by converting logs into count vectors. Lou et al. (2010) propose that IM approach can automatically detect anomalies in logs with the mined invariants. Support Vector Machine (SVM) is a supervised learning method for classification. He et al. (2018) show that an instance is regarded as an anomaly when it is located above the hyperplane by using SVM. Zhang and Sivasubramaniam (2008) apply Logistic Regression (LR) to classify anomalous logs.

PCA, IM, SVM and LR approaches detect anomalies based on count vectors, but these count vectors can’t describe the correlation among events. Recently, there have been many studies on semantic-based anomaly detection. Du et al. (2017) propose a DeepLog method which utilizes Long Short-Term Memory (LSTM) (Greff et al. 2017), to model a system log as a natural language sequence. LogRobust (Zhang et al. 2019a) represents log events as semantic vectors and use an attention-based Bi-LSTM (Chen et al. 2022) model to detect anomalies. Huang et al. (2020) utilize HitAnomaly method to model both log template sequences and parameter values and finally use an attention mechanism as classification model. LSTM, Bi-LSTM and Transformer approaches can extract semantic information from log sequence.

BERT model (Devlin et al. 2018) has obtained the best benchmark on eleven famous natural language processing tasks than other NLP models. There are some new state-of-the-art research works based on pre-trained language models. LogBERT (Guo, Yuan, and Wu 2021) learns the patterns of normal log sequences by two novel self-supervised training tasks, masked log message prediction and volume of hypersphere minimization, but it does not identify and train the semantic information of abnormal logs. NeuralLog (Le and Zhang 2021) proposes a novel log-based anomaly detection approach based on a Transformer-based classification model that does not require log parsing. LAnoBERT (Lee, Kim, and Kang 2021) and A2Log (Wittkopf et al. 2021) use unsupervised learning methods to detect anomalous logs based on BERT, but it does not include the interaction factors between normal logs and anomalous logs. UniLog (Zhu et al. 2021) proposes a pre-trained model based on Transformer for multitask anomalous logs detection, but it requires a lot of
computational capability. BERT-Log proposed in this article has smaller number of parameters and stronger generalization ability.

Challenges of Existing Methods

There are many log-based anomaly detection methods. Compare with the recent research works, the challenges of the existing methods are as follows.

(1) The first challenge is that raw logs should be converted into structured event templates automatically and accurately. Traditionally, log parsing depends on regular expressions which are marked by operation engineer manually. However, these manual approaches are inefficient for large number of logs. For example, thousands of new logs are produced in computer system every day, and we can’t input regular expressions for each new log immediately.

(2) The second challenge is that semantic information of log sequence must be effectively described. The studies (Cherkasova et al. 2009; Lou et al. 2010) apply LSTM and Bi-LSTM to convert log sequence to semantic vectors. But the LSTM and Bi-LSTM are more suitable for time series data. Word2Vec (Wang et al. 2021) is the up-to-date encoding method proposed in the HitAnomaly (He et al. 2018) to map each word in log template to a vector. Therefore, the order of words in the Word2Vec sequence is not taken into account. We should capture all the semantic information from log sequence including context and position.

(3) The third challenge is the definition of sliding window. There are lots of logs from different nodes in a long time, such as BGL dataset. Therefore, many anomalies may occur in different nodes, or different anomalies may occur in the same node for a long time. According to current approaches, it can’t locate each detailed anomaly on one node at a certain time.

(4) The fourth challenge is that model structure must be more suitable for real application scenarios. First, model does not depend on the Parser for some logs not in existing event templates. Second, model can record high detection performance without using abnormal data in the learning process.

Due to the challenges of current approaches, there is a need to propose a new novel detection method to classify anomalies. And then we could obtain the better performance and accuracy.

Methods

The purpose of our article is to detect log anomalies automatically based on pre-trained model. The structure of BERT-Log consists of event template
extractor, log semantic encoder and log anomaly classifier, as shown in Figure 1. The first step is parsing raw logs into structured event templates by Drain and forming log sequence, as described in Section 3.1. In Section 3.2, we product semantic log vectors by utilizing pre-trained language model. Finally we use linear classification to detect anomalies in Section 3.3.

**Event Template Extractor**

Raw logs consist of free-text information. The goal of log parsing is to convert raw messages content into structured event templates. Figure 2 shows that thirteen raw logs with the same block ID “blk_-5966704615899624963” are from HDFS dataset. The order 1, 2, 3 logs have the same event template “Receiving block <* src:/<* dest:/<*),” and the parameter values are not included. Each event template with a unique event ID can represent what has happened in certain block. Finally, we group the event ID of logs into log sequence.

The format of raw logs from HDFS or BGL dataset is different. Firstly, we will use a simple regular expression template to preprocess the logs according to domain knowledge. Then the preprocessed logs form a tree structure. Secondly, log groups (leaf nodes) are searched with the special encoding rules in nodes of the tree. If a corresponding log group is found, it means that log messages are matched with the event template stored in this log group. Otherwise, a new log group is created based on the log content. While paring a new log message, log parser will search the most appropriate log group or
create a new log group. Then we will obtain a structured event template for each log. Each event template has a unique event ID. Finally, log sequences identified by event ID are grouped according to sliding window or session window. HDFS logs with the same block ID record the allocation, writing, replication, deletion operation on the corresponding block. This unique block ID can be used as identifier for session window to group raw logs into log sequence. The parsed log sequences are shown as Table 1.

In this article, we propose an improved log parsing method based on BGL dataset. First, we use Drain to parse the BGL raw log to get a log sequence containing node ID, occurrence time, and message. The duration of BGL logs with the same node ID is longer than HDFS, so it maybe that many anomalies occur in different nodes for a long time.

The sliding window of traditional methods consists of window size and step size, thus a small number of logs with the same node ID are in the same sliding

![Figure 2. Grouping raw logs into log sequence.](image-url)

**Table 1**. Log sequences parsed from HDFS dataset.

| Block ID     | Log sequence                         | Label |
|--------------|--------------------------------------|-------|
| blk_8376667364205250596 | ['5', '5', '22', '5', '11', '9', '11', '9', '26', '26', '26', '22'] | 0     |
| blk_-3102267849859399193  | ['5', '5', '5', '5', '11', '9', '11', '9', '26', '26', '26', '22'] | 1     |
| blk_535106793607016574   | ['5', '22', '5', '5', '5', '11', '9', '11', '9', '26', '26', '26'] | 0     |
| blk_-48460618822572759   | ['22', '5', '5', '5', '5', '26', '26', '26', '11', '9', '11', '9', '11'] | 0     |
| blk_-6635163678784332104  | ['22', '5'] | 1     |
window, as shown in Figure 3a. Most of logs have an effect on one another only with the same node ID, so the sliding window cannot include sufficient interactional logs for training. In order to locate each anomaly in real occurrence time, we use sliding window to form log sequence. The sliding windows of BGL consist of node ID, window size and step size, as shown in Figure 3b. As long as it occurs in the same sliding window, logs are also grouped into the same log sequence, as described in Algorithm 1. The parsed BGL data sets are shown as Table 2.

**Figure 3.** Comparison of sliding window between traditional method and our proposed method.

**Algorithm 1:** Converting raw unstructured logs from BGL dataset into log sequence

**Input:** rawlogs = [log₁, log₂, . . . , logₙ] /*Inputting the raw logs*/

**Progress:**
1. events ← Drain(rawlogs) /*Converting raw logs into events by Drain method*/
2. times ← split_time(starttime, endtime, step) /*Producing time windows*/
3. for t = 1 : times.size do /*Traversing time windows*/
4.   for i = 1 : events.size do /*Traversing event list*/
5.     searched ← false
6.     if events[i].time not in times, do /*Matching time window for event*/
7.       continue;
8.     end if
9.     for j = 1 : seqs.size do /*Traversing sequences*/
10.    if events[i].nodeid == seqs[j].nodeid do /*Matching node id for event*/
11.      seqs[j].seq ← seqs[j].seq + events[i].eventid /*Adding new event*/
12.      searched ← true
13.    end if
14.   end for
15. end for
16. if searched == false do
17.   seqs[seq.size + 1].time ← times, /*Adding time into sequences*/
18.   seqs[seq.size + 1].nodeid ← events[i].nodeid /*Adding node id into sequences*/
19.   seqs[seq.size + 1].seq ← events[i].eventid /*Adding event id into sequences*/
20. end if
21. end if
22. Output: log sequence
Table 2. Log sequences parsed from BGL dataset.

| Node ID     | Log sequence                | Label |
|-------------|-----------------------------|-------|
| R62-M1-NC-I-J18-U01 | [590,590] | 1     |
| R55-M0-N9-C-J06-U11  | [1053,1113,1056,1052,1053,1052] | 0     |
| R65-M0-N4-C-J17-U11  | [291,220,218,57,58] | 1     |
| R63-M1-NB-C-J02-U11  | [291,220] | 0     |
| R62-M1-N7-C-J05-U11  | [291,220,218,57,291,220,218,57] | 1     |

**Log Semantic Encoder**

In this section, we first introduce how to embed log sequences by using log embedding module, and then log encoder module is proposed to implement semantic vector encoding on log sequences. Finally, we utilize pooling layer to obtain semantic vector. The main steps are described as following.

The log sequence vector $X$ is needed to do token embedding. Log sequence can be regard as a sentence token to be calculated in BERT-Log model. There are numerous log sequences in train dataset, we add special characters, such as [CLS] and [SEP], before and after each log sequence to facilitate BERT-Log to recognize. In order to improve the computational efficiency and eliminate the noises in the log sequence, we split the first 510 characters of the log sequence into the training model.

$$LogToken = [CLS]id_1id_2\ldots id_m[SEP], m \leq 510$$  \hspace{1cm} (1)

[CLS] is the beginning symbol of a log sequence, and [SEP] is the end symbol of a log sequence. Different log sequences can be identified by using [CLS] and [SEP]. Log token is the log sequence which has been added the mnemonic symbol. WordPiece model is a data-driven tokenization approach and used to split words in log sequence, each word in the log sequence must be mapped with dictionary. As shown in Equation 2, some words are masked in the log sequence to improve the accuracy of the training. Finally, in order to keep all the sentence lengths consist we add some pads to each sentence.

$$LogToken = [CLS]id_1[mask]id_2\ldots id_m[pad][pad][SEP], m \leq 510$$  \hspace{1cm} (2)

In order to capture effective semantic vector features, log embedding layer are designed in this article to map log sequences to a fixed dimension vector for representing logs. Log embedding layer consists of token embedding, segment embedding and position embedding. Firstly, token embedding will convert each log sequence token into a 768-dimensional vector representation: $T \in (batch\_size, length, 768)$. Segment embedding is implemented to get vector $S \in (batch\_size, length, 768)$ and then position embedding is implemented to get vector $P \in (batch\_size, length, 768)$. Finally we concatenate the three vectors to form the embedding vectors of log sequence. Log embedding layer uses the vectors as the embedding representation of log sequence. Log embedding is a combination of the vector $T$, $S$ and $P$, which is defined as follows:
The detail description for log embedding is shown as Algorithm 2.

**Algorithm 2: Log embedding**

**Input:** seq = id₁, id₂, ... , idₖ

seqs = [seq₁, seq₂, ... , seqₘ] /*Inputting log sequences*/

**Progress:**

1. token ← ∅ /*Initializing the token*/
2. for i = 1 : n do /*Traversing log sequences*/
   3. temp ← [CLS]seq[i] [SEP] /*Adding [CLS] [SEP] for each sequence*/
   4. token ← token + temp /*Forming the token*/
3. end for
4. token ← mask(token) /*Adding the mask for token*/
5. token ← pad(token) /*Padding the token*/
6. position ← position(position_ids) /*Position embedding*/
7. segment ← segment(token_type_ids) /*Segment embedding*/
8. X = token + position + segment /*Forming log embedding vectors*/
9. X = LayerNorm(X) /*Do the LayerNorm function*/
10. X = dropout(X) /*Do the dropout function*/

**Output:** log embedding vector X

Semantic vector will be encoded in the log encoding layer after log embedded. Log encoders are bidirectional encoding structure based on transformer, and it is mainly composed of 12 encoders. Each encoder consists of “multi-head” attention and feed forward (Vaswani et al. 2017). The log encoder is shown in Figure 4. A log sequence consists of many event IDs according to the order. Not every event in the sequence is important. Anomaly is usually decided by some events in the log sequence. Therefore, “multi-head” attention mechanism can capture the relations between the events well. The “multi-head” attention calculates attention score among log sequences. It consists of eight attention heads, and it calculates the attention score in turn.

X = [x₁, x₂, ..., xₙ] is the output vector of log embedding, and where n is the length of log sequence. In order to enhance the fit ability on log sequences, three lengths of log sequence, three matrices are used in “multi-head” attention. X is multiplied by the weight matrices W_q ∈ R^{d×d_q}, W_k ∈ R^{d×d_k} and W_v ∈ R^{d×d_v}, and it forms three matrices: query matrix Q, key matrix K, and value matrix V. For each header, self-attention function is performed by inputting X to get a new vector. A softmax function is utilized to obtain the weights on the values. The attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V.

\[
X = T + S + P
\]

(3)

\[
\text{multihead} = \text{concat}(\text{head}_1, \ldots, \text{head}_k)W^O
\]

(4)

\[
\text{head}_i = \text{softmax} \left( \frac{Q \cdot K^T}{\sqrt{d_k}} \right) V
\]

(5)
The vector at the [CLS] location of the hidden state, which is at the last layer, is used as the semantic representation of the log sequence [21]. It is found that this vector can better represent the semantics of the sentence.

The Log encoders is shown as Algorithm 3.

### Algorithm 3: Log encoders

**Input:** log embedding vector $X$

**Progress:**

1. Initialize $O \in \mathbb{R}^{d \times d}$ /*Initializing output vectors*/
2. for $j = 1 : encoder\_layers\_size$ /*Traversing encoder layers*/
3. Matrices $Q, K, V \in \mathbb{R}^{d \times d}$ /*Creating Q, K, V weight matrices*/
4. for $i = 1 : heads\_size$ /*Creating multi head attention matrices*/
5. Divide $Q, K$ and $V$ into multiple groups based on $X$
6. end for
7. attention $\leftarrow$ matmul($Q, K.$ permute($params)) / scale /*Computing attention*/
8. if mask != NULL do
9. attention $\leftarrow$ attention.m_fill(mask $\equiv 0, -1e10$)
10. end if
11. attention $\leftarrow$ softmax(attention, dim = -1) /* Normalizing attention*/
12. $\theta \leftarrow$ matmul(attention, $V$) /*Computing multi head attention*/
13. $\theta \leftarrow$ layerNorm($\theta$) /*Computing by layerNorm function*/
14. $\theta \leftarrow$ FFN($\theta$) /*Computing by FFN*/
15. $O \leftarrow O + \theta$ /*Computing output vectors*/
16. end for

**Output:** encoder\_layers\_vector: $O$

### Log Anomaly Classifier

In this section, we introduce how to build a log anomaly classifier to perform anomaly detection on the vector of the log semantic encode output, as shown in Figure 5.

Pre-trained language models are more stable than traditional models, but they cannot handle log detection data very well. Therefore, we use fine-tuning...
to train the pre-trained language model. It reduces the impact of language data distribution and log data changes by fine-tuning of BERT model on HDFS and BGL datasets that are closer to the target data distribution. We use full connection neural network and fine-turning to detect anomalous logs.

Obtain the semantic vector $\theta$ of the log sequence, and then build a log anomaly detection layer log-task on top of the last layer of the BERT model.

$$\theta_{\text{fine-tuning}} = \theta_{\text{pre-training}} + \theta_{\text{log-task}}$$ (6)

After neural network training, weight vector $w^{(l)}$ and bias item $b^{(l)}$ are obtained. $f$ is the activation function. For the input vector $X$ of the first layer, the output is calculated by the formula as following:

$$y = f \left( w^{(l)} x + b^{(l)} \right)$$ (7)

The number of input neurons is 768, the number of output neurons is 2, and $f$ is the network activation function.

After we have the prediction results of the log detection task layer, we use the cross-entropy loss function to calculate the log anomaly detection loss estimate.

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i = -[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$ (8)

$y_i$ represents the label of sample $i$, positive value is 1, negative value is 0. $p_i$ represents the probability that sample $i$ is predicted to be positive value.
Results

Experiment Setup

We use two public datasets to evaluate the performance of our algorithm: the HDFS dataset (Hadoop distributed file system), the BGL dataset (BlueGene/L supercomputer system). The detailed introduction of two datasets is shown in Table 3.

(1) The 11,175,629 log messages were collected from more than 200 Amazon EC2 nodes and formed the HDFS dataset. Each block operation such as allocation and deletion is recorded by a unique block ID. All of the logs are divided into 575,061 blocks according to the block ID and form the log sequences. The 16,838 blocks are marked as anomalies. (2) The 4,747,963 log messages were collected from the BlueGene/L supercomputer system at Lawrence Livermore National Labs. There are 348,460 log messages labeled as anomalies. BGL dataset has no unique ID for log sequence, so the sliding window is used to obtain the sequence. The sliding window in this study consists of node ID, window size and step size.

We implements our proposed model on a Windows server with Intel(R) Core(TM) i7-10700F CPU @ 2.90 GHz, 32 G memory and NVIDIA GeForce RTX 3060 GPU. The parameters of BERT-Log are described in Table 4.

Evaluation Metrics

In order to evaluate the effectiveness of the proposed model in anomaly detection, Accuracy, Precision, Recall and F1-Score are used as evaluation metrics. These metrics are defined as follows:

| Dataset | Span of time | File size | Logs | Anomalies |
|---------|--------------|-----------|------|-----------|
| HDFS    | 38.7 hours   | 743,185,031 bytes | 11,175,629 logs | 16,838 blocks |
| BGL     | 214.7 days   | 1,577,982,906 bytes | 4,747,963 logs | 348,469 logs |

Table 4. Parameters of BERT-Log.

| Parameters       | Value       |
|------------------|-------------|
| Learning rate    | 2e-5        |
| Batch size (HDFS)| 16          |
| Batch size (BGL) | 24          |
| Sequences size   | 510         |
| Vocab size       | 30522       |
| Hidden size      | 768         |
| Hidden layers    | 12          |
| Attention heads  | 12          |
| Optimizer        | AdamW       |
| Loss function    | CrossEntropyLoss |
(1) Accuracy: the percentage of log sequences that are correctly detected by the model among all the log sequences.

\[
Accuracy = \frac{TP + TN}{TP + FP + TN + FN}
\]  

(2) Precision: the percentage of anomalies that are correctly detected among all the detected anomalies by the model.

\[
Precision = \frac{TP}{TP + FP}
\]  

(3) Recall: the percentage of anomalies that are correctly detected by the model among all the anomalies.

\[
Recall = \frac{TP}{TP + FN}
\]  

(4) F1-Score: the harmonic mean of Precision and Recall. The maximum value of F1-Score is 1, and the minimum value of F1-Score is 0.

\[
F1\text{-Score} = \frac{2 \times Precision \times Recall}{Precision + Recall}
\]  

TP (true positive) is the number of anomalies that are correctly detected by the model. TN (true negative) is the number of normal log sequences that are detected by the model. FP (false positive) is the number of normal log sequences that are wrongly detected as anomalies by the model. FN (false negative) is the number of anomalies that are not detected by the model.

**Accuracy Evaluation of BERT-Log**

(1) Evaluation on HDFS dataset

Table 5 shows the accuracy of BERT-Log compared to twelve previous methods based on the HDFS dataset. Obviously, BERT-Log has achieved the highest performance among all the methods, with an F1-score of 99.3%. Count vectors based approaches classify anomalies with an F1-score from 70% to 95%, such as PCA, IM, LogCluster, SVM and LR. Semantic vectors based approaches detect anomalies with F1-score from 96% to 99%, such as DeepLog, LogRobust, HitAnomaly and BERT-Log. It indicates that LSTM, BiLSTM, Word2Vec and BERT are better suitable for capturing the semantic information of log sequence.

Recall presents the percentage of anomalies that are correctly detected by the model among all the anomalies. There is need to notify operation engineer immediately when anomalies occur. So recognition of anomalous logs is more important than recognition of normal logs. BERT-Log with a Recall of 99.6%, is better than any other semantic based method. It shows that semantic vectors
are effectively formed by the pre-training and fine-turning mode. Transformer based models such as LAnoBERT, NeuralLog, A2Log, Logbert, and UniLog, which utilize unsupervised methods or unlogged parsing templates, have got F1-scores ranging from 0.75 to 0.98. The BERT-Log benchmark is generally better than transformer based methods. As shown in Table 5, the results of PCA, IM, LogCluster, SVM, LR, DeepLog, LogRobust, HitAnomaly are released by HitAnomaly (Huang et al. 2020). The results of LAnoBERT are released by Lee, Kim, and Kang (2021). The results of NeuralLog are released by Le and Zhang (2021). The results of LogBERT are released by Guo, Yuan, and Wu (2021). The results of UniLog are released by Zhu et al. (2021).

(2) Evaluation on BGL dataset

After raw logs are parsed by Drain method, 1848 event templates are formed on BGL dataset. The number of BGL event templates is much more than HDFS with 48 templates. The duration of BGL dataset (214.7 days) is longer than HDFS dataset (38.7 hours). So the relationships of BGL logs are more complex than HDFS, it is very difficult to capture semantic information on BGL dataset. As shown in Table 5, the previous approaches get F1-score from 11% to 92% on BGL dataset, and most of F1-scores are lower than 50%. Transformer based method can get the better benchmark, and it’s F1-scores exceed 90%. But BERT-Log detect anomalies with an F1-score of 99.4%, it gives 19% performance improvement compared to LogRobust, 7% performance improvement compared to HitAnomaly, 8% performance improvement compared to LogBERT, 12% performance improvement compared to LAnoBERT.

BERT-Log model is better than other previous approaches on BGL dataset, which indicates that anomaly classification model benefits from sliding window, pre-training and fine-turning language model. The sliding window consists of node ID, window size and step size, so it can locate anomaly on each node and provide more accurate fault information to operation engineer. Pre-training and fine-turning language model can also provide more effective semantic information than other approaches. As shown in Table 6, the results

| Model      | Precision | Recall | F1-score |
|------------|-----------|--------|----------|
| PCA        | 1.00      | 0.67   | 0.80     |
| IM         | 0.81      | 0.62   | 0.70     |
| LogCluster | 1.00      | 0.83   | 0.91     |
| SVM        | 1.00      | 0.91   | 0.95     |
| LR         | 0.84      | 0.73   | 0.78     |
| DeepLog    | 0.96      | 0.96   | 0.96     |
| LogRobust  | 0.98      | 0.96   | 0.97     |
| HitAnomaly | 1.00      | 0.97   | 0.98     |
| LAnoBERT   | –         | –      | 0.96     |
| NeuralLog  | 0.96      | 1.00   | 0.98     |
| LogBERT    | 0.87      | 0.78   | 0.82     |
| UniLog     | 0.88      | 0.65   | 0.75     |
| BERT-Log   | 0.996≈1   | 0.996≈1| 0.993≈0.99|
of PCA, IM, LogCluster, SVM, LR, DeepLog, LogRobust, HitAnomaly are released by HitAnomaly (Huang et al. 2020). The results of LAnoBERT are released by Lee, Kim, and Kang (2021). The results of NeuralLog are released by Guo, Yuan, and Wu (2021). The results of UniLog are released by Zhu et al. (2021).

(3) Evaluation on HDFS dataset by Pre-trained Language Model

In order to obtain the effectiveness of the anomalous logs detection method based on pre-trained model proposed in this paper, we used another pre-trained models as log semantic encoders to compare the experiment results on the HDFS dataset, as shown in Table 7. The RoBERTa model and T5 model have obtained the results similar to BERT-Log, and their F1-scores are close to 1. However, the parameters of BERT-Log are the smallest among all the models, which is only 110 M. BERT-Log can obtain the best benchmark similar to the large model ERNIE with 750 M parameters. It indicates that BERT-Log is more suitable for log anomaly detection in real industrial applications. As shown in Table 7, the results of RoBERTa, T5, UniLM, ELECTRA, ERNIE, SpanBERT are from the experiments of this paper.

Experiments in Different Scale of Dataset

Table 8 shows the experiments on different scales of the HDFS dataset. In order to evaluate the performance of BERT-Log in different dataset sizes, we
take the front 1%, 10%, 20%, and 50% of the HDFS dataset as new datasets to classify anomalies. 80% of dataset are used as training set and 20% of dataset are used as testing set. BERT-Log achieves an F1-Score of 0.969 on the small sample set (1%), it gives 43% performance improvement compared to DeepLog and 53% performance improvement compared to LogCluster. It achieves an F1-Score of 0.993 on the dataset (10%), it gives 28% performance improvement compared to LR.

For example, the F1-Score of DeepLog is 0.535 on the 1% dataset and 0.357 on the 10% dataset. We can conclude that previous approaches are unstable in different dataset sizes. The performance of BERT-Log is more stable and it is better than other compared approaches. As shown in Table 8, the results of SVM, LogCluster, LR, DeepLog are from the experiments of this paper.

The BGL dataset is more difficult for anomaly detection compared to the HDFS dataset. So BGL dataset is better suitable to test the stability of model. The same method as HDFS, we take the training ratio 1%, 10%, 20%, and 50% of the BGL dataset to classify anomalies. Table 8 shows the F1-Score of BERT-Log on the three new datasets are all close to 1, and the F1-Score of SVM approach are only no more than 0.58, respectively. BERT-Log approach gives 75% performance improvement compared to SVM. Although the performance of LogRobust and HitAnomaly are stable, the F1-Score is not high enough. It indicates that BERT-Log approach has both better performance and stability on the BGL dataset. BERT-Log was trained with a small training set (1%) to predict 99% of new logs and it achieves an F1-score of 0.989. Compared with other methods, BERT-Log has better generalization ability. As shown in Table 9, the results of SVM, LogCluster, LR, DeepLog are from HitAnomaly (Huang et al. 2020). The results of A2log is from A2log (Wittkopp et al. 2021).

| Dataset Size | Model     | Precision | Recall | F1-Score |
|--------------|-----------|-----------|--------|----------|
| 1%           | SVM       | 1.000     | 0.163  | 0.280    |
|              | LogCluster| 1.000     | 0.279  | 0.436    |
|              | LR        | 1.000     | 0.163  | 0.280    |
|              | DeepLog   | 1.000     | 0.365  | 0.535    |
| 10%          | **BERT-Log** | **0.980** | **0.987** | **0.969** |
|              | SVM       | 0.996     | 0.489  | 0.656    |
|              | LogCluster| 1.000     | 0.240  | 0.386    |
|              | LR        | 0.989     | 0.521  | 0.682    |
|              | DeepLog   | 0.994     | 0.217  | 0.357    |
|              | **BERT-Log** | **0.997** | **0.999** | **0.996** |
| 20%          | SVM       | 0.968     | 0.539  | 0.692    |
|              | LogCluster| 1.000     | 0.234  | 0.379    |
|              | LR        | 0.973     | 0.748  | 0.846    |
|              | DeepLog   | 0.799     | 0.209  | 0.331    |
|              | **BERT-Log** | **0.994** | **0.999** | **0.993** |
| 50%          | SVM       | 0.975     | 0.569  | 0.719    |
|              | LogCluster| 1.000     | 0.194  | 0.325    |
|              | LR        | 0.974     | 0.567  | 0.716    |
|              | DeepLog   | 0.484     | 0.199  | 0.282    |
|              | **BERT-Log** | **1.000** | **0.995** | **0.998** |
The ROC is a curve drawn on a two-dimensional plane. False positive rate (FPR) is defined as the X-axis and true positive rate (TPR) is defined as the Y-axis. Area under the ROC Curve (AUC) refers to the area between the ROC curve and the X-axis. The bigger the AUC value is, the closer the curve is to the upper and left corner, which indicates that better the classification effect is.

In this paper, AUC values are used to evaluate the classification effect of the model. Figure 6 describes ROC curves of anomalous log detection models.
based on HDFS dataset. The AUC value of BERT-Log approach is 0.999 and it is very close to 1, which means that the TPR of positive samples are very high, and the FPR of negative samples are very low. It indicates that BERT-Log approach has a better classification effect than previous approaches, such as DeepLog, LR, and SVM.

**Conclusion**

Raw log messages are unstructured, which contain many different format texts. It is hard to detect numerous anomalies based on unstructured logs. This study proposes a BERT-Log method which can detect log anomalies automatically based on BERT pre-training language model. It can better capture semantic information from raw logs than previous LSTM, Bi-LSTM and Word2Vec methods. BERT-log consists of event template extractor, log semantic encoder, and log anomaly classifier. We evaluated our proposed method on two public log datasets: HDFS dataset and BGL dataset. The results show that BERT-Log-based method has got better performance than other anomaly detection methods.

In the future, we will reduce model training time to improve the real-time log processing capability of the model. Moreover, we plan to propose a new approach to directly classify anomalous logs base on the event templates.

**Disclosure statement**

No potential conflict of interest was reported by the author(s).

**Funding**

The work was supported by the CDTU PHD FUND [2020RC002].

**ORCID**

Hai Liao http://orcid.org/0000-0002-2862-7863

**References**

Aussel, N., Y. Petetin, and S. Chabridon. 2018. Improving Performances of Log Mining for Anomaly Prediction through NLP-based Log Parsing. In Proceedings of 26th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 237–43. Milwaukee.

Bretan, P. 2017. Trap analysis: An automated approach for deriving column height predictions in fault-bounded traps. Petroleum Geoscience 23 (1):56–69. doi:10.1144/10.44petgeo2016-022.
Chen, L. J., J. Ren, P. F. Chen, X. Mao, and Q. Zhao. 2022. Limited text speech synthesis with electroglottograph based on Bi-LSTM and modified Tacotron-2. Applied Intelligence. doi:10.1007/s10489-021-03075-x.

Cherkasova, L., K. Ozonat, N. F. Mi, J. Symons, and E. Smirni. 2009. Automated anomaly detection and performance modeling of enterprise applications. ACM Transactions on Computer Systems 27 (3):1–32. doi:10.1145/1629087.1629089.

Dai, H. T., H. Li, C. S. Chen, W. Y. Shang, and T. H. Chen. 2022. Logram: Efficient log parsing using n-Gram dictionaries. IEEE Transactions on Software Engineering 48 (3):879–92. doi:10.1109/TSE.2020.3007554.

Devlin, J., M. W. Chang, K. Lee, and K. Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, Oct 11.

Do, P., and T. H. V. Phan. 2021. Developing a BERT based triple classification model using knowledge graph embedding for question answering system. Applied Intelligence 52 (1):636–51. doi:10.1007/s10489-021-02460-w.

Du, M., and F. F. Li. 2016. Spell: Streaming parsing of system event logs. In Proceedings of the 16th IEEE International Conference on Data Mining, 859–64. Barcelona.

Du, M., F. F. Li, G. N. Zheng, and V. Srikumar. 2017. Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 24th ACM-SIGSAC Conference on Computer and Communications Security, 1285–98. Dallas.

Greff, K., R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber. 2017. LSTM: A search space Odyssey. IEEE Transactions on Neural Networks and Learning Systems 28 (10):2222–32. doi:10.1109/TNNLS.2016.2582924.

Guo, H. X., S. H. Yuan, and X. T. Wu. 2021. LogBERT: Log Anomaly Detection via BERT. In Proceedings of the IEEE International Joint Conference on Neural Networks, Shenzhen. Electr Network.

He, P. J., J. M. Zhu, S. L. He, J. Li, and M. R. Lyu. 2018. Towards automated log parsing for large-scale log data analysis. IEEE Transactions on Dependable and Secure Computing 15 (6):931–44. doi:10.1109/TDSC.2017.2762673.

He, S. L., J. M. Zhu, P. J. He, and M. R. Lyu. 2016. Experience report: System log analysis for anomaly detection. In Proceedings of the 27th International Symposium on Software Reliability Engineering, 207–18. Ottawa.

He, P. J., J. M. Zhu, Z. B. Zheng, and M. R. Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In Proceedings of the 24th IEEE International Conference on Web Services, 33–40. Honolulu.

Hooshmand, M. K., and D. Hosahalli. 2022. Network anomaly detection using deep learning techniques. CAAI Transactions on Intelligence Technology 7 (2):228–43. doi:10.1007/cit2.12078.

Huang, S. H., Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan. 2020. HitAnomaly: Hierarchical transformers for anomaly detection in system log. IEEE Transactions on Network and Service Management 17 (4):2064–76. doi:10.1109/TNSM.2020.3034647.

Hu, J., Y. J. Zhang, M. H. Zhao, and P. Li. 2022. Spatial-spectral extraction for hyperspectral anomaly detection. IEEE Geoscience and Remote Sensing Letters 19:19. doi:10.1109/LGRLS.2021.3130908.

Ito, K., H. Hasegawa, Y. Yamaguchi, and H. Shimada. 2018. Detecting privacy information abuse from Android apps by API call logs. Lecture Notes in Artificial Intelligence 11049:143–57. doi:10.1007/978-3-319-97916-8_10.

Jukic, O., I. Hedi, and A. Sarabok. 2019. Fault management API for SNMP agents. In Proceedings of the 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, 431–34. Opatija.

Lee, Y., J. Kim, and P. Kang. 2021. LAnoBERT: System log anomaly detection based on BERT masked language model. arXiv preprint arXiv:2111.09564, November 18.
Le, V. H., and H. Y. Zhang. 2021. Log-based anomaly detection without log parsing. In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering, Australia, 492–504. Electr Network.

Liu, C. B., L. L. Pan, Z. J. Gu, J. Wang, Y. Ren, and Z. Wang. 2020. Valid probabilistic anomaly detection models for system logs. Wireless Communications & Mobile Computing. doi:10.1007/s10489-021-02724-5.

Lou, J. G., Q. Fu, S. Q. Yang, Y. Xu, and J. Li. 2010. Mining invariants from console logs for system problem detection. In Proceedings of the 2010 USENIX Annual Technical Conference, Boston, 231–44.

Lv, D., N. Luktarhan, and Y. Y. Chen. 2021. ConAnomaly: Content-based anomaly detection for system logs. Sensors 21 (18):6125. doi:10.3390/s21186125.

Maeyens, J., A. Vorstermans, and M. Verbeke. 2020. Process mining on machine event logs for profiling abnormal behaviour and root cause analysis. Annals of Telecommunications 75 (9–10):563–72. doi:10.1007/s12243-020-00809-9.

Makanju, A., A. N. Zincir-Heywood, and E. E. Milios. 2009. Clustering Event Logs Using Iterative Partitioning. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1255–63. Paris.

Mi, H. B., H. M. Wang, Y. F. Zhou, M. R. Lyu, and H. Cai. 2012. Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs. Science China-Information Sciences 55 (12):2757–73. doi:10.1007/s11432-012-4747-8.

Mi, H. B., H. M. Wang, Y. F. Zhou, M. R. T. Lyu, and H. Cai. 2013. Toward fine-grained, unsupervised. Scalable Performance Diagnosis for Production Cloud Computing Systems 24 (6):1245–55. doi:10.1109/TPDS.2013.21.

Oliner, A., and J. Stearley. 2007. What supercomputers say: A study of five system logs. In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 575+. Edinburgh.

Peng, Y. Q., T. F. Xiao, and H. T. Yuan. 2022. Cooperative gating network based on a single BERT encoder for aspect term sentiment analysis. Applied Intelligence 52 (5):5867–79. doi:10.1007/s10489-021-02724-5.

Setia, S., V. Jyoti, and N. Duhan. 2020. HPM: A hybrid model for user’s behavior prediction based on N-Gram parsing and access logs. Scientific Programming. doi:10.1155/2020/8897244.

Studiawan, H., F. Sohel, and C. Payne. 2021. Anomaly detection in operating system logs with deep learning-based sentiment analysis. IEEE Transactions on Dependable and Secure Computing 18 (5):2136–48. doi:10.1109/TDSC.2020.3037903.

Tang, L., T. Li, and C. S. Perng. 2011. LogSig: Generating system events from raw textual logs. In Proceedings of the 2011 ACM International Conference on Information and Knowledge Management, Glasgow, 785–94.

Tufek, A., and M. S. Aktas. 2021. On the provenance extraction techniques from large scale log files. Concurrency and Computation-Practice & Experience. doi:10.1002/cpe.6559.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, et al. 2017. Attention Is All You Need. Proceedings of the 31st Annual Conference on Neural Information Processing Systems, Long Beach.

Wang, J., C. Q. Zhao, S. M. He, Y. Gu, O. Alfarraj, and A. Abugabah. 2021. LogUAD: Log unsupervised anomaly detection based on Word2Vec. Computer Systems Science and Engineering 41 (3):1207–22. doi:10.32604/csse.2022.022365.

Wittkopp, T., A. Acker, S. Nedelkoski, J. Bogatinovski, D. Scheinert, et al. 2021. A2Log: Attentive Augmented Log Anomaly Detection. arXiv preprint arXiv:2109.09537, Sep 20.
Xu, L., L. Huang, A. Fox, D. Patterson, and M. I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In Proceedings of the Twenty-second ACM SIGOPS Symposium on Operating Systems Principles, 117–32. Big Sky.
Yen, T. F., A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, et al. 2013. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. Proceedings of the 29th Annual Computer Security Applications Conference, 199–208, New Orleans.
Zhang, Y. Y., and A. Sivasubramaniam. 2008. Failure prediction in IBM BlueGene/L event logs. In Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, 2525±. Miami.

Zhang, L., X. S. Xie, K. P. Xie, Z. Wang, Y. Yu, et al. 2019b. An Efficient Log Parsing Algorithm Based on Heuristic Rules. Proceeding of the 13th International Symposium on Advanced Parallel Processing Technologies, 123–134, Tianjin.
Zhang, X., Y. Xu, Q. W. Lin, B. Qiao, H. Y. Zhang, et al. 2019a. Robust Log-Based Anomaly Detection on Unstable Log Data. Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference (ESEC) / Symposium on the Foundations of Software Engineering, 807–817, Tallinn.
Zhao, Z. F., W. N. Niu, X. S. Zhang, R. Zhang, Z. Yu, and C. Huang. 2021. Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network. Applied Intelligence 52 (8):8810–19. doi:10.1007/s10489-021-02863-9.
Zhong, Y., Y. B. Guo, and C. H. Liu. 2018. FLP: A feature-based method for log parsing. Electronics letters 54 (23):1334–35. doi:10.1049/el.2018.6079.
Zhu, J. M., S. L. He, J. Y. Liu, P. J. He, Q. Xie, et al. 2019. Tools and Benchmarks for Automated Log Parsing. Proceedings of the 41st International Conference on Software Engineering - Software Engineering in Practice, 121–130, Montreal.
Zhu, Y., W. B. Meng, Y. Liu, S. Zhang, T. Han, et al. 2021. UniLog: Deploy One Model and Specialize it for All Log Analysis Tasks. arXiv preprint arXiv:2112.03159, Dec 6.