# A New Distributed Log Anomaly Detection Method based on Message Middleware and ATT-GRU

**Wei Fang[124*], Xuelei Jia[13], Wen Zhang[13] and Victor S. Sheng[5]**
[1] School of Computer and Software, Engineering Research Center of Digital Forensics, Ministry of Education, Nanjing University of Information Science and Technology, Nanjing 210044, China
[e-mail: hsfangwei@sina.com]
[2] Jiangsu Collaborative Innovation Center of Atmospheric Environment and Equipment Technology, Nanjing University of Information Science and Technology, Nanjing 210044, China
[e-mail: hsfangwei@sina.com]
[3] Nanjing Xinda Institute of Meteorological Science and Technology Co., Ltd., Nanjing, Jiangsu 210044, China
[e-mail: xueleijia123@163.com]
[4] Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, Suzhou 215031, China
[e-mail: hsfangwei@sina.com]
[5]Department of Computer, Texas Tech University, Lubbock, TX 79409, USA
[e-mail: victor.sheng@ttu.edu]
[*]Corresponding author: Wei Fang

*Received May 30, 2022; accepted February 12, 2023; published February 28, 2023*

## *Abstract*

Logs play an important role in mastering the health of the system, experienced operation and maintenance engineer can judge which part of the system has a problem by checking the logs. In recent years, many system architectures have changed from single application to distributed application, which leads to a very huge number of logs in the system and manually check the logs to find system errors impractically. To solve the above problems, we propose a method based on Message Middleware and ATT-GRU (Attention Gate Recurrent Unit) to detect the logs anomaly of distributed systems. The works of this paper mainly include two aspects: (1) We design a high-performance distributed logs collection architecture to complete the logs collection of the distributed system. (2)We improve the existing GRU by introducing the attention mechanism to weight the key parts of the logs sequence, which can improve the training efficiency and recognition accuracy of the model to a certain extent. The results of experiments show that our method has better superiority and reliability.

***Keywords:*** Log Anomaly Detection, Message Middleware, Attention Mechanism, Gate Recurrent Unit (GRU)

This work was supported by the National Natural Science Foundation of China (Grant No.42075007), the Open Grants of the State Key Laboratory of Severe Weather（No.2021LASW-B19), and the Open Project of Provincial Key Laboratory for Computer Information Processing Technology under Grant KJS2275, Soochow University, the Postgraduate Research & Practice Innovation Program of Jiangsu Province(SJCX22_0341)

## 1. Introduction

The stability of system has always been the most concerned problem of software development engineers and operation and maintenance engineers. The consequences of software design vulnerabilities and security risks are very serious. There are many huge losses caused by software defects in history. In 1994, a Chinook helicopter crashed in Scotland, there was evidence that the system error of the helicopter was the culprit; Another flight accident caused by software vulnerability occurred in 1993, a JAS 39 Gripen fighter in Sweden crashed due to a bug in the flight control software [1]. Therefore, the security and stability of software are the most important issues in the software development industry. Anomaly detection of logs is mainly divided into traditional manual marking, machine learning methods and deep learning algorithms. With the development of artificial intelligence, deep learning provides new ideas for logs anomaly detection.

At present, mainstream service systems are developing in the direction of distributed microservices. The logs of a distributed system has two characteristics. First, the amount of logs is massive. A large scale aviation meteorological system produces logs of the order of 50Gb (about 120-200 million lines) per hour [2]; Since log data is massive and scattered, the analysis and anomaly detection of the entire distributed system log will be much more complicated than the stand-alone system, so an efficient log collection and analysis method is needed.

In view of the problems existing in the current research on log anomalies in distributed systems, we can summarize our key contributions as follows:

(1) We propose the first application of distributed message middleware to collect and transmit logs, which can well meet the needs of distributed systems and big data.

(2) In the log anomaly detection stage, our work proposes a log anomaly detection model based on ATT-GRU (Attention and GRU). The attention mechanism can be added to increase the degree of attention to the key log information. In addition, introducing the attention mechanism can improve the training efficiency of the neural network. Compared with the detection method based on LSTM [4]. GRU[5] has a simpler structure, fewer training samples, which is lightweight and easy to implement. Therefore, it is a good attempt to combine the attention mechanism and GRU to detect anomalies in the log sequence.

This paper first describes the algorithm of log collection and anomaly detection, and then verifies the feasibility of our method through experiments.

## 2. Related Work

In recent years, many scientific research teams have carried out related work on log anomaly detection and have achieved fruitful results. As early as 2004, Mike Chen[6] proposed the use of decision trees to detect errors in HDFS logs. Yinglung Liang[7] used SVM (Support Vector Machines, Support Vector Machines) to perform an exception handling on IBM BlueGene/L log data in order to develop effective fault tolerance strategies and predict system failure events. Since deep learning was proposed in 1957, it has been widely used in many fields. At present, the application of deep learning in the field of logs anomaly detection mainly includes three aspects. The first is the application research of RNN model. Among many models, RNN is unique in its powerful learning ability on time series features. Meng[8] add synonyms and antonyms to train the DLCE word vector, the word vector introduces semantic information for LSTM sequential detection, and obtains quantitative relationships such as opening and closing through word frequency statistics for quantitative detection; Yuan [9] used LSTM to propose an unsupervised online log anomaly detection framework, and designed a dynamic threshold algorithm that can dynamically adjust the input length of historical information, which can determine the input length according to recent detection events. Du[10] constructed a workflow model based on Bi-LSTM (Bi-directional Long Short-Term Memory) and implemented online detection of log anomalies with a 92% accuracy rate. The second is the application research of CNN (Convolutional Neural Network). CNN has attracted the attention of many scholars due to its superior effect in the field of image vision. Mei[11] used CNN-Text to detect software anomalies based on log data, reaching about 90% accuracy on different datasets. The third is the model based on the attention mechanism. The attention mechanism can not only calculate in parallel, improve the processing efficiency, but also solve the problem of gradient disappearance over the long distance of the log key. Huang[12] designed the log sequence encoder and parameter value encoder based on attention mechanism to capture the log semantic information, and the performance and robustness of the method are evaluated through experiments on unstable log datasets; Guo[13] built a sequence model based on multi-head attention to process the log stream as a template event sequence, and trained the proposed model for log anomaly detection through the prediction task of the next event. Nedelkoski [14]built a self-attention based encoder model and augmented the training data with auxiliary information such as system switching time, which enabled the model to more effectively capture the difference between normal and abnormal logs through intensive log data training, comparing this model with the PCA method on the public dataset, the accuracy rate reaches about 90%, and the F1 value is about 0.67. In the above studies, in the face of massive log data analysis, the deep learning method is generally better than the traditional method of manual feature extraction, but there are still problems such as taking the log as a whole as the analysis object, ignoring the log key and log parameter characteristics, and the poor universality of different models.

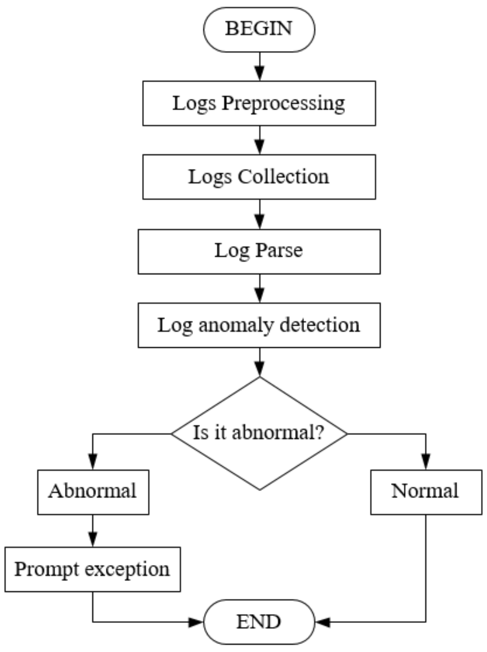


**Fig. 1.** Common log detection flowchart.

Common log anomaly detection methods can be summarized into three steps[14]: (1) log collection and preprocessing, (2) log analysis, (3) anomaly detection. Generally, the storage directory of the log file is specified when the system is developed, so it is necessary to collect and read the log file. After that, the log sequence needs to be parsed, and the way of parsing can be determined according to the selected anomaly detection method. Finally, anomaly detection is performed on the parsed log sequence. The flow chart of the entire log abnormality detection is shown in **Fig. 1**.

In summary, many existing studies have achieved certain results for system log anomaly detection. However, there are very few researches on log anomaly detection for distributed systems, and many studies analyze anomaly logs only for a single log record. This is limited. Sometimes system abnormalities are not recorded in a single log. As shown in the above, it is necessary to combine multiple previous logs for joint analysis to find the anomaly situation of the system.

## 3. Design of Distributed Log Anomaly Detection Method

### 3.1 Use Distributed Messaging Middleware to Collect Logs

Our work aims at the problem of log anomaly detection in distributed systems, so the efficient collection of log data is one of the key tasks and an indispensable step before log anomaly detection.

Each module of the distributed system is distributed on different physical machines, so it is essential to collect logs from the distributed cluster and collect the scattered logs into independent high-performance machines to facilitate the feature extraction of all logs and anomaly detection. Due to the huge amount of logs, the transmission overhead caused by network transmission will be very large. Therefore, it is necessary to perform a log preprocessing first to remove the parts of the log sequence that are not helpful for log anomaly detection. Preprocessing first reduces the amount of data that needs to be transmitted over the network to a certain extent, and improves the efficiency of the entire anomaly detection to a certain extent.

Filebeat is used in the log preprocessing and collection phase. Filebeat is a lightweight conveyor that can be used to collect logs from specified log files or locations. The collected logs can be filtered first by using the configuration parameter exclude_lines. The working mechanism of Filebeat is as follows: When Filebeat is started, it will start one or more inputs that are searched in the location specified by the log data. For each log found by Filebeat, Filebeat will start a harvester. Each harvester reads a single log of the new content and sends the new log data to libbeat, which aggregates events and sends the aggregated data to the output configured by Filebeat. The output can be Elasticsearch or Kafka, etc., In this paper ,we choose Kafka as the output of Filebeat.

The logs after preprocessing and collection can be transmitted, and the logs in the distributed cluster are collected into a physical machine with the best performance. Kafka is a high-throughput distributed publish-and-subscribe messaging system with high throughput, low latency, durability, and high concurrency. It is very suitable for the transmission of a large amount of data to real-time logs. The logs preprocessed and collected by Filebeat in different hosts are divided into Topic according to the host names of different machines to publish the log stream, the architecture diagram is shown in **Fig. 2**. There are three hosts in the distributed cluster used in this paper. The host names are Hadoop1, Hadoop2, and Hadoop3. The logs collected by Filebeat in the corresponding machines will be published to Topic named after the host name. The purpose of this design is to prevent logs from being mixed randomly and form "dirty data".

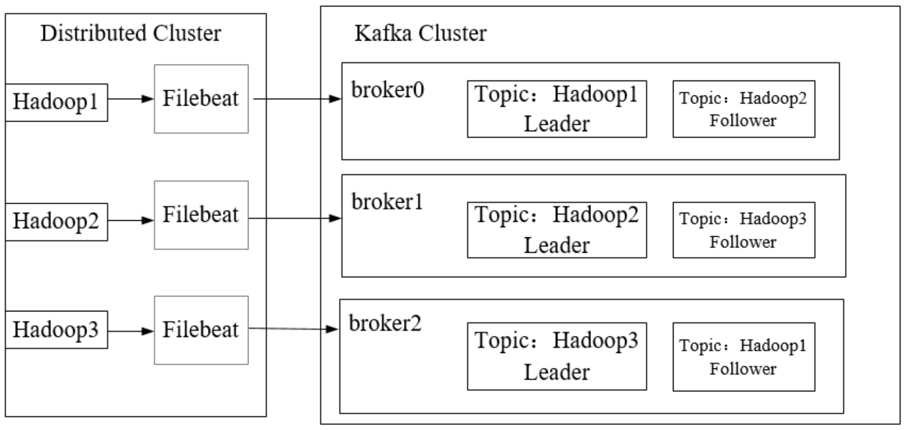


**Fig. 2.** The Architecture diagram of topic division log stream

It can be found from the architecture of **Fig. 2** that in order to achieve high availability in the Kafka cluster, each broker is configured with a backup follower node. If the leader node goes down, there can also be a follower node to maintain the normal operation of the Kafka cluster. Using Logstash to consume and read log data from the Kafka cluster at the stage of log data reception and storage. Logstash is an open source data collection engine[15] with real-time pipeline function. In Logstash, you can use Filter to configure log stream filtering, and output the log to a specified location for storage after normalization. The overall log filtering and collection algorithm flow chart is shown in **Fig. 3**.

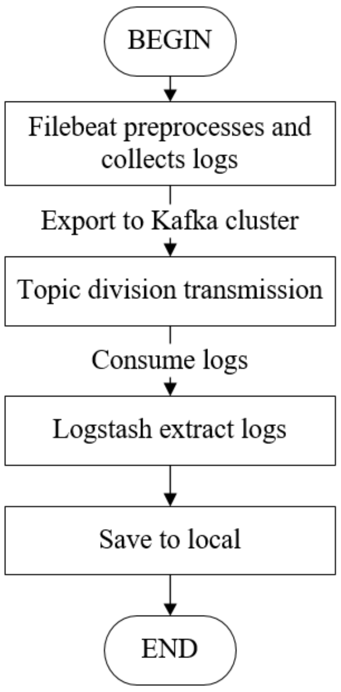


**Fig. 3.** Algorithm flow chart of log preprocessing and collection

From the above description of the log preprocessing and collection method based on message middleware, three message middleware are mainly used to preprocess, transmit, receive and save logs, and meet the requirements of large High throughput, low latency and high availability required for data transmission.

## 3.2 Attention Mechanism for Model Input

The attention mechanism[16]draws on human attention when working and thinking, and enhances the attention to the key information of the sample to improve the contribution of the key information to the result. The introduction of the attention mechanism into the neural network can improve the training efficiency of the neural network. The principle of the attention mechanism is as follows: If there are ***N*** input vectors$[x_1, x_2 \dots, x_N]$,, in order to select information related to a specific task, a query vector ***q*** and a scoring function s are introduced. The related formula is shown in (1).

$$\alpha_n = soft\max(s(x_n, q)) \tag{1}$$

In formula (1), $\alpha_n$ is the attention distribution, which represents the degree of correlation between the input vector and the query vector; Softmax is a multinomial logistic regression; the scoring function s usually uses an additive model as shown in formula (2). Among them, ***W, U, V*** are learnable parameters.

$$s(x,q) = V^T \tanh(Wx + Uq) \tag{2}$$

The structure of the attention mechanism is shown in **Fig. 4**.

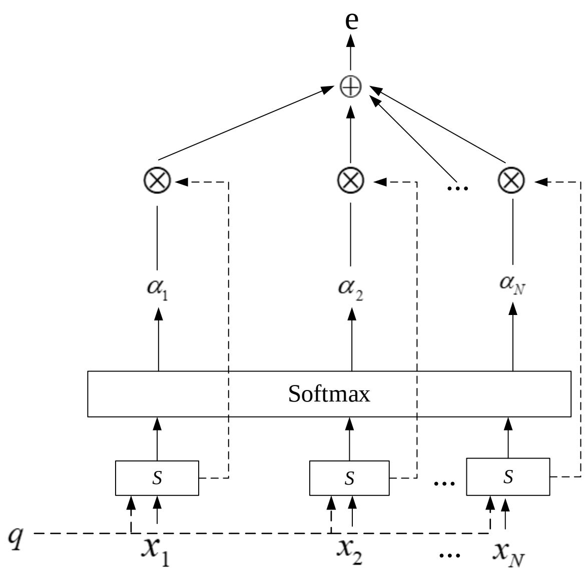


**Fig. 4.** The structure of the attention mechanism

In the **Fig. 4**, ***e*** is a summary of the input information that assigns the weight ratio according to the attention distribution.

The attention mechanism is mainly applied to the learning of visual semantics and visual representations, and usually acts on the middle layer features of neural networks. It is the most widely used in the Seq2Seq (sequence to sequence) model[17-18]. Seq2Seq is generally implemented through the Encoder-Decoder (encoding-decoding) framework[19]. The Encoder and Decoder parts can be any text, voice, image, and video data, etc. When solving some time series prediction problems, the attention mechanism is usually set between the

Encoder and the Decoder. When the Encoder outputs, it is necessary to focus on which parts of the input sequence, and then generate the next output according to the area of interest to make the sequence prediction more precise.

Since neural network-based machine learning is driven by data, we often need massive amounts of training data to offset the adverse effects of noise samples on training. The most direct manifestation is the reduction of sample efficiency. Therefore, we use the attention mechanism to give different levels of attention to input samples of different quality, thereby improving the efficiency of sample utilization.

Therefore, before the log samples are input into the GRU network for anomaly detection, attention weighting operations need to be performed on the correlation degree of the samples, so that the samples used for model training have better quality and improve the efficiency of the anomaly detection model.

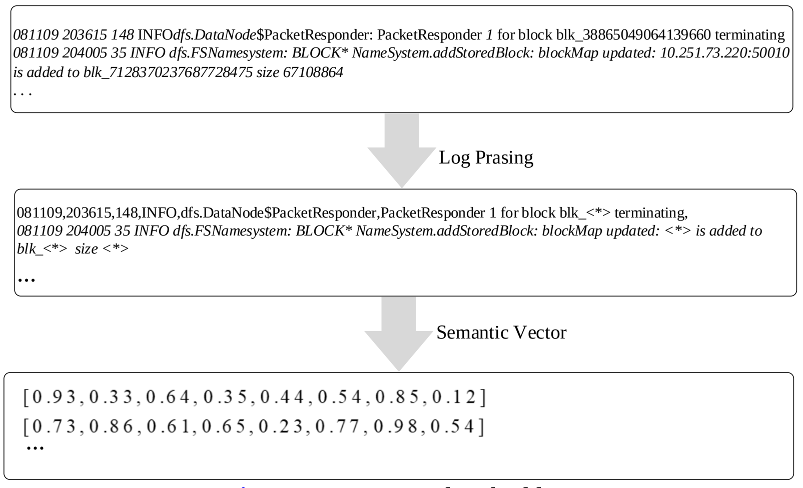
## 3.3 Log Anomaly Detection Model Based on GRU

The method used in the log anomaly detection part in this paper is a GRU-based deep learning detection model, and the method of converting log data to matrix vectorization usually includes TF-IDF[20], one-hot[21] and word2vec (word to vector)[22]. word2vec is a neural network-based word vector construction method proposed by Google in 2013. A weight matrix (3) can be trained through a shallow neural network structure to convert the high-dimensional sparse matrix obtained by one-hot encoding (4) into a dense vector matrix of low dimensionality (5). In addition, word2vec can reflect words with high similarity through the distance of corresponding word vectors in the vector space[23]. As you can see in **Fig. 5**.

$$W = (w_1, w_2, ..., w_d), W \in R^{d*n} \tag{3}$$

$$X = (X_1, X_2, X_3, ..., X_l) \tag{4}$$

$$Y = W * X, Y \in R^{d*l} \tag{5}$$



```
081109 203615 148 INFOdfs.DataNode$PacketResponder: PacketResponder 1 for block blk_38865049064139660 terminating
081109 204005 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.73.220:50010
is added to blk_7128370237687728475 size 67108864
. . .
```

```
081109,203615,148,INFO,dfs.DataNode$PacketResponder,PacketResponder 1 for block blk_<*> terminating,
081109 204005 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to
blk_<*>  size <*>

...
```

```
[0.93,0.33,0.64,0.35,0.44,0.54,0.85,0.12]
[0.73,0.86,0.61,0.65,0.23,0.77,0.98,0.54]
...
```

**Fig. 5.** Log parse and embedding

GRU was first proposed by Chung. The structure of a single GRU network is shown in **Fig. 6**. Compared with LSTM, GRU integrates the input gate and forget gate in LSTM into an update gate ***Zt***, replace the input gate of LSTM with ***Rt*** gate. Therefore, the number of gates in the GRU network is changed from 3 in LSTM to 2, which is effective to reduce the number of specific parameters and shortens the model training time. ***Zt*** is mainly used to summarize the past data of the new input information, and ***Rt*** is mainly to determine the probability of entering the state information in the previous unit into this unit. The network structure diagrams of a single LSTM and GRU are shown in **Fig. 6** and **Fig. 7**.

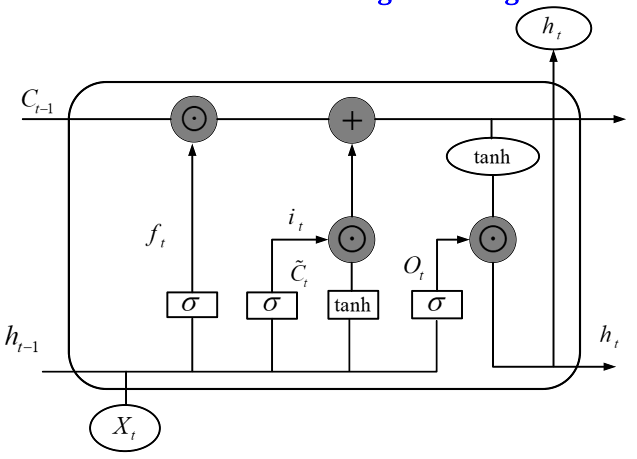


**Fig. 6.** The network structure diagrams of a single LSTM

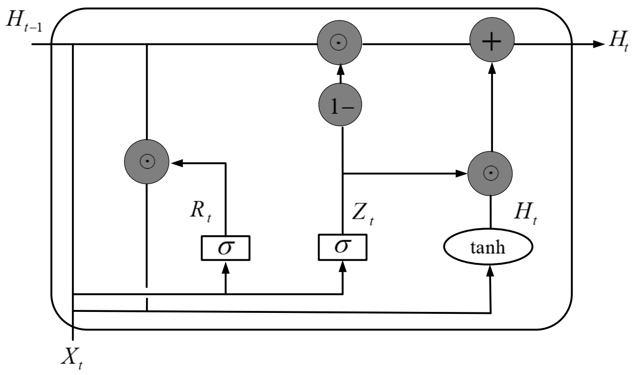


**Fig. 7.** The network structure diagrams of a single GRU

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \tag{6}$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz}) + b_z \tag{7}$$

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h) \tag{8}$$

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t \tag{9}$$

In **Fig. 7**, $\boldsymbol{X_t}$ is input sequence, $\boldsymbol{R_t}$ is reset gate, $\boldsymbol{Z_t}$ is update gate, $\boldsymbol{\tilde{H}_t}$ is candidate hidden state, and the activation function is sigmod. The reset gate $\boldsymbol{R_t}$ determines how the new input information is combined with the previous memory, and the update gate $\boldsymbol{Z_t}$ defines the amount of the previous memory saved to the current time step. If sets the reset gate to 1 and the update gate to 0, this model will change to the standard RNN model. According to formula (7), it can be seen that the output of the hidden state of the current state depends on the current input and the input of the previous hidden layer. Therefore, the final input depends on the current input vector and the previous input, which can handle the relationship between the log context and perform an anomaly detection training and detection on the logs sequence better.

### 3.4 Log Anomaly Detection Model based on ATT-GRU

The historical log records contain a large amount of system status information, which plays a vital role in predicting future abnormal conditions of the system. A gating unit is introduced into the GRU network, which automatically extracts features from historical log information, and at the same time improves the accuracy of predicting log abnormalities. In order to improve the contribution of important log records to abnormal prediction results, a pre-attention mechanism is adopted to assign different weights to log records, so that log records with high contribution to accurate prediction results are emphasized. The pseudo code of the entire log anomaly detection is shown in **Table 1**.

The ATT-GRU model is shown in **Fig. 8**. The entire model is composed of five parts, which are log input, attention layer, GRU network, fully connected layer, and output layer. The description of the model is as follows:

(1) Log input: Using the logs collected by the message middleware as the input of the entire model.

(2) Word2vec layer: The logs collected from the distributed cluster are vectorized using word2vec. Through training, the log content can be processed as a vector in a multi-dimensional vector space, and the similarity in the vector space can be used to represent the semantic similarity of the log. The processed log vector can be expressed as $X = [x_1, x_2, \dots, x_N]^T$ .

(3) Attention layer: The log matrix to be input into the GRU network needs to be processed by the attention layer first. Since the input log sequence contains a lot of information that is not helpful for prediction, it is necessary to assign a value to the training weight of the log sequence to improve the contribution of important log sequences, and also improve the efficiency of model training and the accuracy of the final prediction result. The calculation formula for the weight of the attention layer is shown in formula (10). The output of the m-th sequence is expressed as:

$$y_m = \alpha_m x_m \tag{10}$$

(4) GRU network: After the feature matrix of the log sequence is processed by the attention layer, the feature matrix of different log sequences is given different weights. These matrices with different weights are input into the GRU network for a log anomaly detection learning training. The input of the GRU layer is h：

$$h = GRU(h_{m-1}, y_m) \tag{11}$$

(5) Full Connection Layer: Using a fully connected layer to gradually improve learning level of the features of the log sequence, and classify the final prediction results. The activation function is ReLU[24].

**Table 1.** The Pseudo Code of the Entire Log Anomaly Detection

**Algorithm1** Log Anomaly Detection

```
Input: Collected Logs
Output: Prediction Result
1: Procedure ATT-GRU()
2:    input log ← collected logs
3:    processed logs vector $\boldsymbol{X}$ ← $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]^T$
4:    for i < N
5:       att-feature ← Attention($\boldsymbol{X}$)
6:    end for
7:    train_output_feature ← GRU(logs vector $\boldsymbol{X}$, att-feature)
```

```
8:    Prediction Result ← Softmax(train_output_feature)
9:  END Procedure
```

For the ATT-GRU-based detection model used in this paper, the main models and training parameters are shown in **Table 2** below. The main parameters used in the model and training are as follows.

**Table 2.** Model and Training Parameters

| | | |
|---|---|---|
| ATT-GRU Network | input_size | 1 |
| | hidden_size | 64 |
| | num_layers | 2 |
| | bidirectional | True |
| Train | batch_size | 2048 |
| | optimizer | adam |
| | learn_ratio | 0.001 |
| | epoch | 369 |
| | window_size | 10 |

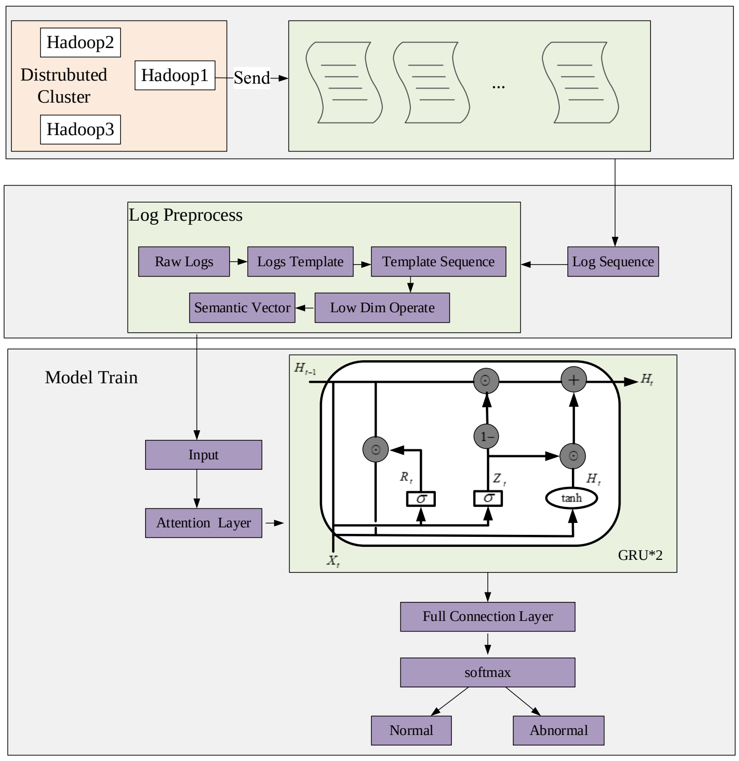


**Fig. 8.** The structure diagrams of ATT-GRU Model

During the model training process, CrossEntropyLoss[25]is used to optimize the network trainable parameters. Since our work is aimed at the prediction of log anomaly detection, it is a typical two-class classification problem. In the case of dichotomy, there are only two cases where the model needs to predict the result at the end. For each category, our predicted probabilities are $p$ and $1-p$. The expression is shown in formula (12).

$$L=\frac{1}{N}\sum_{i}L_i=\frac{1}{N}\sum_{i}-[y_i\bullet\log(p_i)+(1-y_i)\bullet\log(1-p_i)] \qquad (12)$$

$y_i$ represents the label of $i$, the positive class is 1, and the negative class is 0. $p_i$ represents the probability that the prediction of sample $i$ is positive.

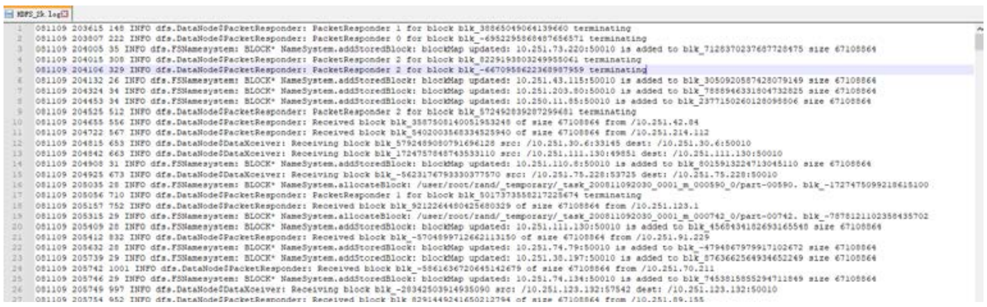
## 4. Experiments

### 4.1 Datasets

In our experiments, we used two most widely-used public datasets to evaluate our method, i.e., HDFS (Hadoop Distributed File System) dataset  and BGL (Blue Gene/L supercomputer) dataset , which have been widely used in the existing work on log-based anomaly detection. For ease of presentation, we call them HDFS and BGL directly, examples of the two logs are shown in the **Fig. 9**, **Fig. 10**. Each kind of logs contains time stamps, log sequence constants and parameter variables, which are standard time series data.

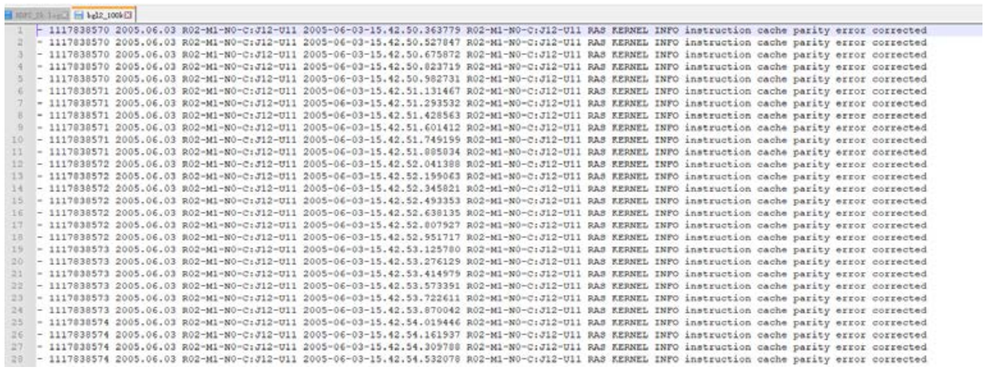
The training datasets for log anomaly detection in our experiments are HDFS(Hadoop Distributed File System) and BGL(Blue-Gene/L). The HDFS dataset is Amazon public 11,175,629 logs, which were generated on Amazon private cloud EC2 benchmark workload and classification labels have been flagged as abnormal logs by Amazon distributed systems experts. We selected 6000 normal log sequences and 6000 abnormal log sequences for training, and randomly collected 500,000 log sequences among the remaining log sequences, including 490,000 normal sequences and 10,000 abnormal sequences for testing.

The BGL data contains of 4,747,963 logs generated by the Blue-Gene/L supercomputer system, of which 48,460 logs were labelled as malfunctions. Unlike HDFS logs, BGL logs do not record the block ID generated by each log event. We select 3000 normal samples and 3000 abnormal samples as the training set, and the remaining 3000 abnormal sequences and 20,000 normal sequences as the test set to maintain the ratio of normal samples to abnormal samples in the real data.



```
081109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_38865049064139660 terminating
081109 203807 222 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block blk_-6952295868487656571 terminating
081109 204005 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.73.220:50010 is added to blk_7128370237687728475 size 67108864
081109 204015 308 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_8229193803249955061 terminating
081109 204106 329 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_-6670958622368987959 terminating
081109 204132 26 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.115:50010 is added to blk_3050920587428079149 size 67108864
081109 204324 34 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.203.80:50010 is added to blk_7888946331804732825 size 67108864
081109 204453 34 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.11.85:50010 is added to blk_2377150260128098806 size 67108864
081109 204525 512 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_572492839287299681 terminating
081109 204655 556 INFO dfs.DataNode$PacketResponder: Received block blk_3587508140051953248 of size 67108864 from /10.251.42.84
081109 204722 567 INFO dfs.DataNode$PacketResponder: Received block blk_5402003568334525940 of size 67108864 from /10.251.214.112
081109 204815 653 INFO dfs.DataNode$DataXceiver: Receiving block blk_5792489080791696128 src: /10.251.30.6:33145 dest: /10.251.30.6:50010
081109 204842 663 INFO dfs.DataNode$DataXceiver: Receiving block blk_1724757848743533110 src: /10.251.111.130:49851 dest: /10.251.111.130:50010
081109 204908 31 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.110.8:50010 is added to blk_8015913224713045110 size 67108864
081109 204925 673 INFO dfs.DataNode$DataXceiver: Receiving block blk_-5623176793330377570 src: /10.251.75.228:53725 dest: /10.251.75.228:50010
081109 205035 28 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock: /user/root/rand/_temporary/_task_200811092030_0001_m_000590_0/part-00590. blk_-1727475099218615100
081109 205056 710 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_5017373558217225674 terminating
081109 205157 752 INFO dfs.DataNode$PacketResponder: Received block blk_9212264480425680329 of size 67108864 from /10.251.123.1
081109 205315 29 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock: /user/root/rand/_temporary/_task_200811092030_0001_m_000742_0/part-00742. blk_-7878121102358435702
081109 205409 28 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.111.130:50010 is added to blk_4568434182693165548 size 67108864
081109 205412 832 INFO dfs.DataNode$PacketResponder: Received block blk_-5704899712662113150 of size 67108864 from /10.251.91.229
081109 205632 28 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.74.79:50010 is added to blk_-4794867979917102672 size 67108864
081109 205739 29 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.38.197:50010 is added to blk_8763662564934652249 size 67108864
081109 205742 1001 INFO dfs.DataNode$PacketResponder: Received block blk_-5861636720645142679 of size 67108864 from /10.251.70.211
081109 205746 29 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.74.134:50010 is added to blk_7453815855294711849 size 67108864
081109 205749 997 INFO dfs.DataNode$DataXceiver: Receiving block blk_-20342503914935090 src: /10.251.123.132:57542 dest: /10.251.123.132:50010
081109 205754 952 INFO dfs.DataNode$PacketResponder: Received block blk_8291449241650212794 of size 67108864 from /10.251.89.155
```

**Fig. 9.** HDFS Log



```
- 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.363779 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.527847 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.675872 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.823719 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.982731 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838571 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.51.131467 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838571 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.51.293532 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838571 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.51.428563 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838571 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.51.601412 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838571 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.51.749199 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838571 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.51.885034 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838572 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.52.041388 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838572 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.52.199063 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838572 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.52.345821 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838572 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.52.493353 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838572 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.52.638135 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838572 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.52.807927 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838572 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.52.951717 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838573 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.53.125780 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838573 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.53.276129 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838573 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.53.414979 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838573 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.53.573391 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838573 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.53.722611 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838573 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.53.870042 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838574 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.54.019446 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838574 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.54.161937 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838574 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.54.309788 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- 1117838574 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.54.532078 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
```

**Fig. 10.** BGL Log

## 4.2 Experiments Preparation

In the experiment, our work uses three Centos servers to form a Hadoop distributed cluster as the experimental basis. Each node will continuously generate HDFS (Hadoop Distributed File System) activity logs, and use Filebeat and Kafka to preprocess and collect HDFS logs in the cluster. The collected logs are sorted and filtered by Logstash to generate logs to be encoded. In order to make the training and testing of the log detection model based on ATT-GRU more efficient, a separate high-performance machine is used for testing at this stage. The machine configuration information for distributed cluster and model training, testing and verification is shown in **Table 3**.

**Table 3.** Hardware and software environment configuration table

| HostName | Memory | CPU | Kafka | GPU | Logstash | Hadoop | Filebeat |
|---|---|---|---|---|---|---|---|
| Hadoop1 (NameNode) | 4G | i5 -11300H | 0.11 | / | 7.13.3 | 3.1.3 | 7.14.0 |
| Hadoop2 (DataNode) | 4G | i5 -11300H | 0.11 | / | / | 3.1.3 | 7.14.0 |
| Hadoop3 (Secondary NameNode) | 2G | i5 -11300H | 0.11 | / | / | 3.1.3 | 7.14.0 |
| Win10 | 64GB | i7 9700K | / | 2*2080 Ti | / | / | / |

The experiments use HDFS and BGL logs to realize log anomaly detection in distributed system scenarios, and the anomaly detection mainly uses the log anomaly detection method based on ATT-GRU. The classification label of the dataset has been marked by Amazon distributed system experts. Implementation environment is as follows: CPU is Intel core i7 9700k, the memory size is 64GB, the GPU is two NVIDIA 2080Ti, and the deep learning framework is PyTorch 1.5.1.

The accuracy rate (Precision), recall rate (Recall) and comprehensive evaluation index (F1-Measure) are used to evaluate the detection effect of the model[26]. The formulas are as follows (13)-(15).

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

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

$$F1 - measure = \frac{2TP}{2TP + FP + FN} \tag{15}$$

TP: Abnormal sequence samples are correctly detected as abnormal sequence samples.
FN: Abnormal sequence samples are incorrectly detected as normal sequence samples.
TN: Normal sequence samples are correctly detected as normal sequence samples.
FP: Normal sequence samples are misclassified as abnormal sequence samples.

We compared ATT-GRU with some mainstream anomaly detection methods in recent years, such as LSTM[10], GRU and PCA[27]. Through comparison, it is found that the ATT-GRU detection model used in this paper has a significant advantage over the other four methods in terms of comprehensive evaluation indicators. The final experimental result comparison chart is shown in **Fig. 11** and **Fig. 12**, and the specific experimental result data is shown in **Table 4, 5**.

**Table 4.** Comparison of Experiments Result On HDFS

| Method Used | Precision ↑ | Recall ↑ | F1-measure ↑ |
|---|---|---|---|
| **ATT-GRU*** | **96.35%** | **97.79%** | **95.67%** |
| ATT-LSTM | 95.62% | 97.63% | 94.26% |
| LSTM | 95.73% | 93.34% | 94.52% |
| PCA | 97.50% | 63.50% | 76.90% |

**Table 5.** Comparison of Experiments Result On BGL

| Method Used | Precision ↑ | Recall ↑ | F1-measure ↑ |
|---|---|---|---|
| **ATT-GRU*** | **95.54%** | **98.63%** | **97.06%** |
| ATT-LSTM | 95.21% | 96.56% | 96.34% |
| LSTM | 95.34% | 92.55% | 95.75% |
| PCA | 92.45% | 65.54% | 79.12% |

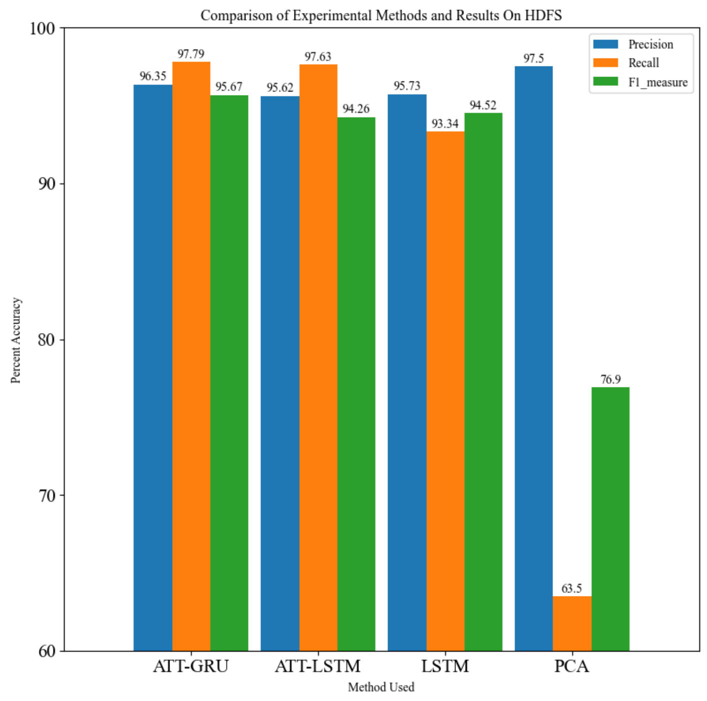


**Fig. 11.** Experiments Result Comparison On HDFS

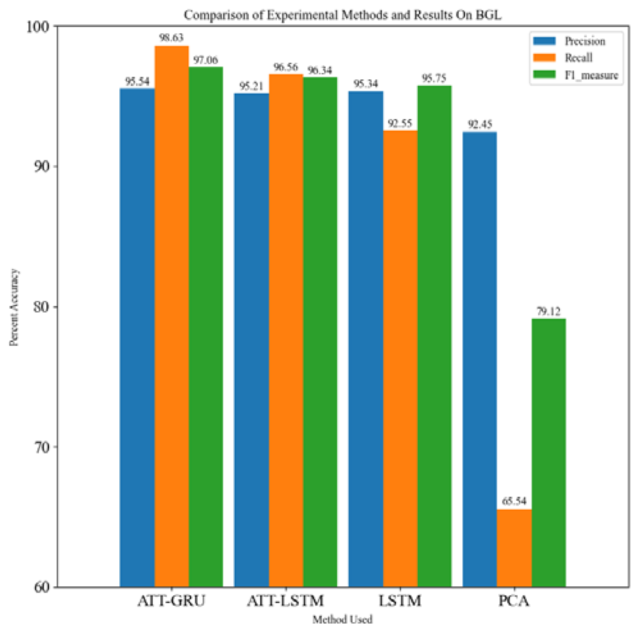


**Fig. 12.** Experiments Result Comparison On BGL

From the table and the result figure, it can be clearly seen that the ATT-GRU used in this paper is significantly better than the other three methods. It is noticeable that the three indicators of the PCA method are not very stable, which may be due to the presence of noise in the dimensionality reduction. Overall, ATT-GRU model used in this paper performs well under various indicators, which also proves that the model we proposed is reliable and excellent.

Our work is to add an attention mechanism before GRU network. In order to prove the effectiveness of the attention module added in this paper, we also compare the experimental results of ATT-GRU and GRU on two datasets. The experimental results are shown in **Table 6,7**.

**Table 6.** Comparison of Experiments Result On HDFS

| Method Used | Precision ↑ | Recall ↑ | F1-measure ↑ |
|---|---|---|---|
| **ATT-GRU** | **96.35%** | **97.79%** | **95.67%** |
| GRU | 95.73% | 93.34% | 94.97% |

**Table 7.** Comparison of Experiments Result On BGL

| Method Used | Precision ↑ | Recall ↑ | F1-measure ↑ |
|---|---|---|---|
| **ATT-GRU** | **95.54%** | **98.63%** | **97.06%** |
| GRU | 95.58% | 98.40% | 96.23% |

The effectiveness of the ATT-GRU method is proved by comparing the results of the ablation experiments, and each experimental effect is better than GRU. Each experiment is carried out in the same configuration environment, and the effectiveness of the method proposed in this paper is proved by comparison experiments and ablation experiments.

## 5. Conclusion

In this paper, we propose a novel distributed log anomaly detection method based on message middleware and ATT-GRU. First, we use the log collection architecture we designed to collect logs in a distributed environment in real time. After the log collection is completed, we encode the log sequence with semantic vectors, and then input it into the ATT-GRU log anomaly detection model for training. Based on the original GRU, we introduce attention mechanism to weight the log sequence. This operation can improve the utilization of important logs and efficiency of model training. Finally, the classification result is obtained by the softmax classifier. Comparative experiments show that our proposed ATT-GRU method is reliable and superior.

However, there are still problems that need to be improved in this paper, and more detailed research is needed in the future work. For example, considering the spatiotemporal information of logs. The spatiotemporal information here can represent the spatial relationship between different computer nodes, because there are many requests between different systems, which will inevitably lead to spatial correlation between logs. Whether incorporating spatially relevant information into the log anomaly detection model will have better results requires further research and experiments.

## Acknowledgement

This work was supported by the National Natural Science Foundation of China (Grant No.42075007), the Open Grants of the State Key Laboratory of Severe Weather(No.2021LASW-B19), and the Open Project of Provincial Key Laboratory for Computer Information Processing Technology under Grant KJS2275, Soochow University, the Postgraduate Research &Practice Innovation Program of Jiangsu Province(SJCX22_0341).

## References

[1] A. Hammouri, M. Hammad, M. Alnabhan and F. Alsarayrah, "Software bug prediction using machine learning approach," *International Journal of Advanced Computer Science and Applications*, vol.9, no.2, pp.78-83, 2018. Article (CrossRef Link)

[2] H. Mi, H. Wang, Y. Zhou, M. R. Lyu and H. Cai, "Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245-1255, 2013. Article (CrossRef Link)

[3] H. Yan and P. Chen, "Research on the evaluation method of power grid statistical data quality based on cloud model," *Computer Applications and Software*, vol.31, no.12, pp.100-103, 2014. Article (CrossRef Link)

[4] R. Kanthavel and R. Dhaya, "Prediction model using reinforcement deep learning technique for osteoarthritis disease diagnosis," *Computer Systems Science and Engineering*, vol. 42, no.1, pp. 257–269, 2022. Article (CrossRef Link)

[5] R. Dey and F. M. Salemt, "Gate-variants of gated recurrent unit (GRU) neural networks," in *Proc. of 2017 IEEE 60th International Midwest Symposium on Circuits and Systems*, Boston, USA, pp.1597-1600, 2017. Article (CrossRef Link)

[6] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan and E. Brewer, "Failure diagnosis using decision trees," in *Proc. of ICAC*, New York, NY, USA, pp. 36-43, 2004. Article (CrossRef Link)

[7] Y. Liang, Y. Zhang, X. Hui and R. Sahoo, "Failure Prediction in IBMBlueGene/L Event Logs," in *Proc. of Seventh IEEE International Conference on Data Mining*, Omaha, NE, USA, pp.583-588, 2007. Article (CrossRef Link)

[8] W.Meng,Y.Liu and Y.Zhu, "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," in *Proc. of IJCAI*, vol.19, no.7, pp. 4739-4745, 2019. Article (CrossRef Link)

[9] Y.Yuan, S.Adhatarao and M.Lin, "Ada: Adaptive deep log anomaly detector," in *Proc. of IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp.2449-2458, 2020. Article (CrossRef Link)

[10] M. Du, F. Li and G. Zheng, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp.1285-1298, 2017. Article (CrossRef Link)

[11] Y.D.MEI, X.CHEN and Y.Z.SUN, "A Method for Software System Anomaly Detection Based on Log Information and CNN-Text," *Chinese Journal of Computers*, vol.43, no.2, pp.366-380, 2020.

[12] S.Huang, Y.Liu and C.Fung, "HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log," *IEEE Transactions on Network and Service Management*, vol.17, no.4, pp.2064-2076, 2020. Article (CrossRef Link)

[13] Y.Guo, Y.Wen and C.Jiang, "Detecting Log Anomalies with Multi-Head Attention (LAMA)," *arXiv preprint arXiv*, vol. 2101, pp.2392, 2021. Article (CrossRef Link)

[14] S.Nedelkoski, J.Bogatinovski and A.Acker, "Self-attentive classification-based anomaly detection in unstructured logs," in *Proc. of 2020 IEEE International Conference on Data Mining (ICDM)*, pp.1196-1201, 2020. Article (CrossRef Link)

[15] B. H. Lee and D.M.Yang, "A Security Log Analysis System using Logstash based on Apache Elasticsearch," *Journal of the Korea Institute of Information and Communication Engineering*, vol.22, no.2, pp.382-389, 2018. Article (CrossRef Link)

[16] M.T. Luong, H. Pham and C.D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," *Computer Science*, pp.1-11, 2015. Article (CrossRef Link)

[17] W.Fang, F.H. Zhang, V.S.Sheng. Y.W.Ding, "SCENT: A new precipitation nowcasting method based on sparse correspondence and deep neural network," *Neurocomputing*, vol.448, pp.10-20, 2021. Article (CrossRef Link)

[18] L.Zhang, G.M. Zhu, P.Y .Shen, J. Song, S. A. Shah and M. Bennamoun, "Learning Spatiotemporal Features Using 3DCNN and Convolutional LSTM for Gesture Recognition," in *Proc. of 2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, Venice, Italy, pp. 3120-3128, 2017. Article (CrossRef Link)

[19] M. Pirhooshyaran and L.V.Snyder, “Forecasting, hindcasting and feature selection of ocean waves via recurrent and sequence-to-sequence networks,” *Ocean Engineering*, vol.207.no.C4, pp.107424, 2020. Article (CrossRef Link)

[20] D. Hiemstra, “A probabilistic justification for using tf×idf term weighting in information retrieval, ” *International Journal on Digital Libraries*, vol.3, no.2, pp.131-139, 2000. Article (CrossRef Link)

[21] S. Hanzawa, T. Sakata, K. Kajigaya, R. Takemura and T. Kawahara, “A large-scale and low-power CAM architecture featuring a one-hot-spot block code for IP-address lookup in a network router,” *IEEE Journal of Solid-State Circuits*, vol.40, no.4, pp.853-861, 2005. Article (CrossRef Link)

[22] T. Mikolov, K. Chen, G. Corrado and J. Dean, “Efficient estimation of word representations in vector space,” ar*Xiv preprint arXiv*, vol.1301, pp.3781, 2013. Article (CrossRef Link)

[23] Q. Q. Niu, “Design and Implementation of Web Anomaly Detection Algorithm Based on Machine Learning,” M.S. dissertation, Beijing University of Posts and Telecommunications, BeiJing, China, 2020.

[24] I.Diakonikolas, S.Goel, S.Karmalkar, A.R.Klivans and M.Soltanolkotabi, “Approximation schemes for relu regression,” in *Proc. of Conference on Learning Theory*, PMLR, pp.1452-1485, 2020. Article (CrossRef Link)

[25] X.Zeng, Y.Zhang and X.Wang, “Fine-grained image retrieval via piecewise cross entropy loss,” *Image and Vision Computing*, vol. 93, no.103820, 2020. Article (CrossRef Link)

[26] R.Yacouby and D.Axman, “Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models,” in *Proc. of the First Workshop on Evaluation and Comparison of NLP Systems*, pp.79-91, 2020. Article (CrossRef Link)

[27] W.Xu, L.Huang and A.Fox, “Largescale system problem detection by mining console logs,” in *Proc. of SOSP’09*, pp. 117-132, 2009. Article (CrossRef Link)

**Wei Fang** is currently a Professor in the Jiangsu Engineering Center of Network Monitoring at the NanJing University of Information Science & Technology in China, also with State Key Laboratory for Novel Software Technology, Nanjing University. He received his Ph.D and M.Sc degrees from Soochow University all in Computer Science. His research interests are in the areas of Cloud Computing, Big Data, Deep Learning. He is a PC member for a number of international conferences and a reviewer for several international journals.

**Xuelei Jia** received the B.S. degree in software engineering from the Jiangsu University of Technology, China, in 2020. He is now pursuing a M.S. degree in Computer Science and Technology at Nanjing University of Information Science & Technology, Nanjing, JiangSu, China. His research interests include the deep learning and microservice.

**Wen Zhang** received the B.S. degree in software engineering from the Yancheng Teachers University, China, in 2020. She is now pursuing a M.S. degree in Computer Science and Technology at Nanjing University of Information Science & Technology, Nanjing, JiangSu, China. His research interests include the deep learning and Air route planning.

**Victor S. Sheng** (Senior Member, IEEE) received the master degree in computer science from the University of New Brunswick, Canada, in 2003, and the Ph.D. degree in computer science from Western University, London, ON, Canada, in 2007. He was an Associate Research Scientist and a NSERC Postdoctoral Fellow of information systems with the Stern Business School, New York University, after he received his Ph.D. He was an Associate Professor of computer science with the University of Central Arkansas, and the Founding Director of the Data Analytics Lab (DAL). Since 2018, he has been a Tenured Associate Professor with the Department of Computer Science, Texas Tech University, Lubbock, TX, USA. His research interests include data mining, machine learning, and related applications.