symmetry

Article

LogLS: Research on System Log Anomaly Detection Method Based on Dual LSTM

Yiyong Chen, Nurbol Luktarhan * and Dan Lv

College of Information Science and Engineering, Xinjiang University, Urumqi 830046, China; chenyiyong578@stu.xju.edu.cn (Y.C.); lvdan@stu.xju.edu.cn (D.L.)
* Correspondence: nurbol@xju.edu.cn

Abstract: System logs record the status and important events of the system at different time periods. They are important resources for administrators to understand and manage the system. Detecting anomalies in logs is critical to identifying system faults in time. However, with the increasing size and complexity of today's software systems, the number of logs has exploded. In many cases, the traditional manual log-checking method becomes impractical and time-consuming. On the other hand, existing automatic log anomaly detection methods are error-prone and often use indices or log templates. In this work, we propose LogLS, a system log anomaly detection method based on dual long short-term memory (LSTM) with symmetric structure, which regarded the system log as a natural-language sequence and modeled the log according to the preorder relationship and postorder relationship. LogLS is optimized based on the DeepLog method to solve the problem of poor prediction performance of LSTM on long sequences. By providing a feedback mechanism, it implements the prediction of logs that do not appear. To evaluate LogLS, we conducted experiments on two real datasets, and the experimental results demonstrate the effectiveness of our proposed method in log anomaly detection.

Keywords: system logs; anomaly detection; LSTM; time series forecasting

check for updates

Citation: Chen, Y.; Luktarhan, N.; Lv, D. LogLS: Research on System Log Anomaly Detection Method Based on Dual LSTM. *Symmetry* **2022**, *14*, 454. https://doi.org/10.3390/sym14030454

Academic Editor: Juan Alberto Rodríguez Velázquez

Received: 25 January 2022
Accepted: 16 February 2022
Published: 24 February 2022

1. Introduction

Many log files are often produced during the operation of modern systems. They reflect the running state of the system and record the activity information of specific events in the system. They are valuable resources to understand the state of the system. Therefore, system logs are an important data source for performance monitoring and anomaly detection and have become a research hotspot in the field of anomaly detection [1].

At present, log anomaly detection can be roughly divided into three categories: rule-based anomaly detection, unsupervised anomaly detection, and supervised anomaly detection.

Rule-based exception detection [2] generally requires a manual analysis of logs and rule creation in advance, and the degree of automation is also low. For example, ref. [3] created rule sets based on analyzing log time series information, which effectively reduced the false-positive rate of the system but with low automation and high labor cost.

With the unsupervised anomaly detection method [4], the manual consumption is reduced to a large extent, and there is no need for premarked training data. Anomaly detection can be performed by judging the difference between the log sequence to be detected and the normal log sequence. Wei Xu et al. [5] used abstract syntax trees (AST) and principal component analysis (PCA) to process the parsed log feature set, reduce the complexity of the feature set to be analyzed, and obtain effective exceptions test results. Jian-Guang Lou et al. [6,7] proposed an unstructured log analysis technology for anomaly detection, a new algorithm for automatically discovering program invariants in logs, and mining program invariants in log message groups, which revealed the inherent linearity of the program workflow. Qingwei Lin et al. [8] proposed a method of log clustering for log problem identification. In log clustering, each log sequence was represented by a vector, the similarity value between the two log sequences was calculated, and the clustering hierarchy was applied. The hierarchical clustering grouped similar log sequences into clusters.

Based on the supervised anomaly detection method [9], it is necessary to use the premarked data to perform log anomaly detection through a pretrained model. For example, ref. [10] used a log event counting vector to train a logistic regression model and calculated the abnormal probability of a log sequence. In reference [11], the log event counting vector was input into a support vector machine (SVM) to train the hyperplane, and the position relationship between the detection log sequence and the hyperplane was judged to determine the abnormal log sequence.

With the development of deep learning [12], scholars have promoted the association between neural networks and log detection, and some abnormal log detection methods based on neural networks have gradually emerged [13], which have achieved good results in practical applications. Bin Xia et al. [14] proposed a generative adversary network based on long short-term memory (LSTM) that used a custom log parser to extract structured information and convert each log into an event. Min Du et al. [15] proposed a deep neural network model that used LSTM to model system logs as natural-language sequences. DeepLog obtained the log model from normal execution and performed anomaly detection on log data through this model. Compared with machine learning methods, DeepLog overcomes the deficiency of the recurrent neural network (RNN) and has high accuracy. However, the method for anomaly detection only considers the impact of the log's pre-event on the current event while ignoring the postsequence when performing anomaly detection. The impact of events on current events is prone to deviations over time in a long sequence of events, which subsequently affects the accuracy of subsequent event predictions, lowering the confidence in the results. In terms of efficiency under detection, this method still has room for improvement.

Zhang X et al. [16] used log event semantic information to extract log sequence context information through a bi-LSTM model to effectively improve the accuracy of log anomaly detection. To solve the problem of low detection efficiency of long sequence abnormal logs, Yang Ruipeng et al. [11] introduced a time convolutional neural network into abnormal log detection and replaced the fully connected layer with adaptive global average pooling, which effectively solved the overfitting problem and improved detection efficiency.

To solve the problem that unstructured logs [17] are difficult to detect directly, this paper adds a filtering function based on the Spell [18,19] method to parse unstructured logs more accurately, which lays a good foundation for anomaly detection of log sequences. In this paper, a log path anomaly detection method based on a dual LSTM network is proposed, which fully considers the interaction and latent symmetry information between the events before and after and makes up for the problem that the log sequence anomaly detection method based on deep learning ignores the log time correlation. Compared with similar previous methods, this method has improved accuracy and F1-measure on HDFS datasets.

The contributions of this paper are as follows:

1. By adding a filter step on Spell, the problem that the current log template is not accurate enough is solved, which is more conducive to the performance of the later log detection work.
2. Two LSTM models are constructed according to the preorder and postorder relations of the log sequence. Combined with the two models, a complete log anomaly detection model is constructed, which fully considers the interaction of before and after log events and solves the problem of insufficient feature mining of log sequences.
3. This paper provides an update mechanism. It solves the problem of inaccurate detection of unseen log sequences.

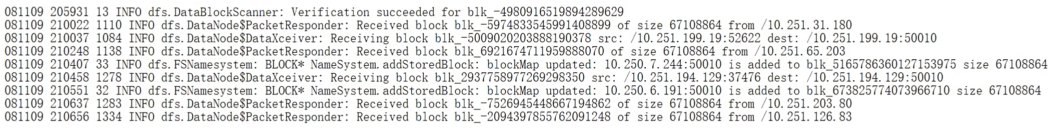
The rest of this article is organized as follows. In Section 2, we introduce the related work. In Section 3, we summarize the construction of the model from three aspects:

log parsing, feature extraction, and anomaly detection. Finally, we evaluate the model's performance in Section 4 and summarize this article in Section 5.

2. Preliminaries

2.1. Log Parser

The first step of abnormal log detection is to collect the log, obtain the original data and preprocess it. After log collection, we preprocess the data. According to specific requirements, invalid information in the log data is further removed, including repeated information and useless information. Finally, we obtain the log data, as shown in Figure 1. Then, we parse the data, analyze the log structure and information, and obtain the log template for each event as the basis for subsequent feature extraction.



```
081109 205931 13 INFO dfs.DataBlockScanner: Verification succeeded for blk_-4980916519894289629
081109 210022 1110 INFO dfs.DataNode$PacketResponder: Received block blk_-5974833545991408899 of size 67108864 from /10.251.31.180
081109 210037 1084 INFO dfs.DataNode$DataXceiver: Receiving block blk_-5009020203888190378 src: /10.251.199.19:52622 dest: /10.251.199.19:50010
081109 210248 1138 INFO dfs.DataNode$PacketResponder: Received block blk_6921674711959888070 of size 67108864 from /10.251.65.203
081109 210407 33 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.7.244:50010 is added to blk_5165786360127153975 size 67108864
081109 210458 1278 INFO dfs.DataNode$DataXceiver: Receiving block blk_2937758977269298350 src: /10.251.194.129:37476 dest: /10.251.194.129:50010
081109 210551 32 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.6.191:50010 is added to blk_6738257740739667l0 size 67108864
081109 210637 1283 INFO dfs.DataNode$PacketResponder: Received block blk_-7526945448667194862 of size 67108864 from /10.251.203.80
081109 210656 1334 INFO dfs.DataNode$PacketResponder: Received block blk_-2094397855762091248 of size 67108864 from /10.251.126.83
```

Figure 1. HDFS log information sample.

The purpose of the log parser is to normalize the structured log data. By parsing the log file to extract the log key, the log sequence is constructed, and the log data of different structures are converted into log key sequences with the same format (sometimes called a log event sequence) [20]. For example, the log "081109 205931 13 INFO dfs.DataBlockScanner: Verification succeeded for blk_-4980916519894289629" in Figure 1 indicates that the content fragment is "Verification succeeded for blk_-4980916519894289629", the constant part is "Verification succeeded for ", and the variable is "blk_-4980916519894289629"; variable parameters can be abstracted as *. The common constant in all similar log entries is called the log key, which can be used to indicate the log type. By replacing it with a log key k_1 (e.g., Verification succeeded for (*)), we obtain a sequence $\{k_1, k_2, k_3, \ldots, k_i\}$. The parameter value is a very valuable type of information in the log and reflects the operating status and performance of the system. Some parameter values can also be used as the identification of the execution sequence, such as block_id in the HDFS log, as a unique identifier, which can extract the log sequence of a specific module.

At present, there are many automatic log analysis methods in the industry. CFG [21] mainly uses code analysis with AST and CFG algorithms. This method adopts the offline mode; with general accuracy, the construction time complexity is O(n^3), and the search efficiency is high. LKE [22] first converts the free-format text of the log into log keys, and then clusters the obtained keys. By mapping each log message to a series of log keys, the log message can be converted into a log key sequence. The algorithm used in this method is clustering, which adopts offline mode and has average accuracy. The construction time complexity is O(n^2), and the search efficiency is very low. Logarm [23] is an effective automatic log parsing method using an n-gram dictionary in natural-language processing. The method adopts the online mode, the construction time complexity is O(n), and the accuracy and efficiency are relatively high.

Drain [24] uses a fixed-depth tree structure to represent log messages and effectively extract common templates. The method adopts the online mode, the construction time complexity is O(n), and the accuracy and search efficiency are relatively high. Spell is a more advanced log parsing tool. It uses the longest common subalgorithm (LCS) to parse logs in a stream. It is implemented by MIT's logPAI team [25,26] as an open-source system and can analyze logs online. This article uses Spell to parse the log template from the log data, and manually filters, deduplicates, and merges the obtained templates to obtain the final log template that is used as the basis for log anomaly detection.

2.2. Architecture and Overview

The architecture of log anomaly detection is shown in Figure 2. It includes three main modules: log analysis, log key anomaly detection, and a workflow model for anomaly detection.

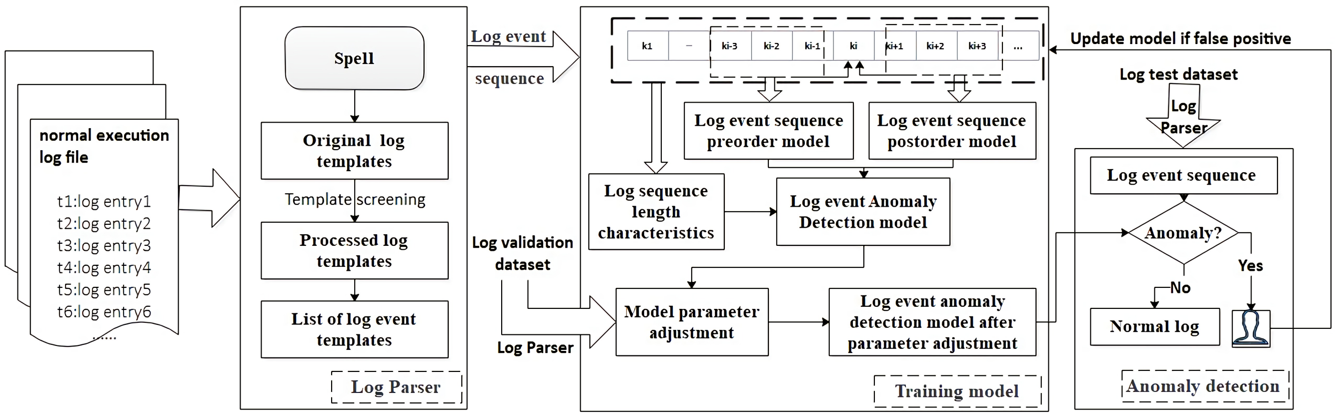


Figure 2. LogLS architecture.

In the log parsing stage, the parsing results obtained by simply using the Spell method are not completely correct. Some log templates are duplicated or redundant, which leads to poor anomaly detection effects. In response to this situation, in this article, to improve the Spell method we add manual filtering to obtain the final log template list. Then, the unstructured log data are parsed into log events according to the log template list, and finally, the log event sequence file is obtained for model training.

In the training phase, the training set is the log entries from the normal system execution path, i.e., the log events obtained after log parsing. For different system logs, there are different ways to construct a log event sequence. If the log has a unique identifier, such as an HDFS log, the log event sequence is constructed according to the unique identifier. If there is no unique identifier, such as the BGL log, we use a sliding window to construct the log event sequence. Using the log event sequence parsed in the log file for model training, we obtain prelog and postlog event sequence LSTM models . These two LSTM models are combined to form a complete anomaly detection model. Of course, how to combine these two models requires the use of another feature (length) of the log event sequence. During model training, we count the distribution of the length of the log event sequence for preliminary filtering of the log event sequence. Finally, according to the log verification dataset and many tuning experiments, we obtain the most suitable model parameters.

In the detection phase, we use the log test dataset to perform model detection. First, we generate a log event sequence from the original log through the parsed log template and then determine the use method of the preorder and postorder detection models according to the sequence length characteristics. According to the selection of the model, a certain log event in the incoming log event sequence is detected, and whether the log event deviates from the prediction result of the normal log sequence is determined. If there is too much deviation, it is judged as anomaly. If the result of the judgment is normal, the next log event detection work is performed on the sequence until the entire log event sequence detection is completed or an anomaly log event that deviates from the normal log sequence is found. If a certain log event in the log sequence is predicted to be abnormal, the entire log sequence is marked as abnormal. This method provides a feedback mechanism, and logs marked as anomalies will be provided to the user for further operation. If the user finds that the detected abnormal marking result is falsely reported, the falsely reported log event sequence can be added to the training set. When there are many false positives, users can retrain the model according to the new training set to gradually update the model.

3. Methodology

This paper proposes a system log anomaly detection method based on dual LSTM. Through the cooperation of two LSTM models, the gradient problem of a single LSTM model in long sequence prediction [27] is solved, and the performance of anomaly detection is improved. Figure 2 shows the three main modules of this method: log parsing, log key anomaly detection model, and anomaly detection workflow model. In the rest of this section, we will cover the various parts of this method in detail.

3.1. Log Parser (Spell)

In the log parsing module, we convert the original log into a structured log through the log parser. Log parsing is considered a common preprocessing of unstructured logs and is an important part of most log analysis tasks. There are many parsing methods at present, among which Spell is currently the better one. It is a log parsing method based on the longest common subsequence (LCS). The time complexity of this method for processing each log entry e is close to linear (linearly related to the size of e) and unstructured log messages are parsed into structured log templates. Although Spell is a better log analysis method at present, the log template parsed by this method is not completely correct. If it is used directly to generate log events and compose a log event sequence for anomaly detection, it is difficult to achieve optimal results.

The main dataset of this article is HDFS logs, which can well reflect the effectiveness of the LogLS method. Table 1 shows the HDFS log template parsed by the Spell method. There are 37 types of HDFS log templates obtained. We can see that some log templates are duplicated. For example, the three log event templates in {E7, E8, E9} are similar, and E8 can contain the other two. If we directly use these templates to generate log sequences, it will seriously affect the effect of log anomaly detection, so this method has room for improvement. This article adds manual deduplication steps to further optimize the effect of log parsing and achieve a higher level of detection.

Table 1. The event template obtained from parsing the HDFS log by the spell method.

EventId	EventTemplate
E1	Adding an already existing block (*)
E2	Verification succeeded for (*)
...	...
E7	writeBlock (*) received exception java.io.IOException Connection reset by peer
E8	writeBlock (*) received exception (*)
E9	writeBlock (*) received exception java.io.IOException Could not read from stream
...	...
E14	PacketResponder (*)(*) Exception java.io.IOException Broken pipe
E15	PacketResponder (*) 2 Exception (*)
E16	PacketResponder (*) 1 Exception (*)
...	...
E36	Deleting block (*) file (*)
E37	BLOCK* NameSystem.allocateBlock (*) (*)

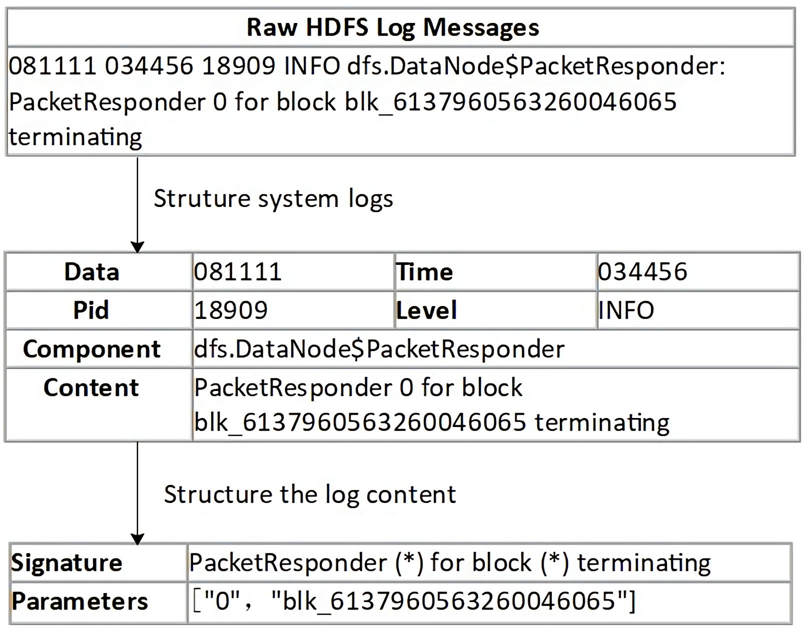
To improve accuracy, we need to eliminate duplicate data in the log event template obtained by Spell. We do so by selecting a few representative log entries that conform to each event template and check whether they are repeated. If they are repeated, according to the frequency of the log event template, they are unified into the event template with the most frequent occurrences. The log events in Table 1 are reprocessed, and the results are shown in Table 2.

Table 2. HDFS log event template after processing.

EventId	EventTemplate
E1	Adding an already existing block (*)
E2	(*)Verification succeeded for (*)
...	...
E7	writeBlock (*) received exception (*)
E8	PacketResponder (*) for block (*) Interrupted.
E9	Received block (*) of size (*) from (*)
...	...
E14	Exception in receiveBlock for block (*) (*)
E15	Changing block file offset of block (*) from (*) to (*) meta file offset to (*)
E16	(*):Transmitted block (*) to (*)
...	...
E28	BLOCK* NameSystem.addStoredBlock: addStoredBlock request received for (*) on (*) size (*) However, it does not belong to any file.
E29	PendingReplicationMonitor timed out block (*)

In this article, we first use the system log template to divide the unstructured log into several parts (e.g., date, time, content, etc.), and then further extract meaningful information (e.g., events) from these parts. Usually, the event consists of three parts (time stamp, signature and parameters). Figure 3 illustrates the HDFS log parsing process. The signature attribute is the log template. The specific algorithm is implemented as follows:

1. The initialization program first defines a log object (LCSObject) such as the log key (LCSseq) and line number list (lineIds), and defines a log object list (LCSMap), which is used to save each log object.
2. Enter the log file and read it line by line.
3. Read a line of log, and then traverse the LCSMap to see if there is already an LCSObject in the list that has the same LCSseq (log key). If such an LCSObject exists, add the lineIds of this log to the lineIds of the LCSObject. If not, then generate a new LCSObject to LCSMap.
4. Keep reading the log until the end.



```
081111 034456 18909 INFO dfs.DataNode$PacketResponder:
PacketResponder 0 for block blk_6137960563260046065
terminating
```

Data	081111	**Time**	034456
Pid	18909	**Level**	INFO
Component	dfs.DataNode$PacketResponder		
Content	PacketResponder 0 for block blk_6137960563260046065 terminating		

Signature	PacketResponder (*) for block (*) terminating
Parameters	["0"， "blk_6137960563260046065"]

Figure 3. Example of log parser.

The template extracted from the log in Figure 3 is "PacketResponder (*) for block (*) terminating", where "(*)" represents the variable parameter. If you enter a new log entry "PacketRespo-nder 2 for block blk_2529569087635823495 termina-ting", Spell's idea is not to extract the log key directly, but to extract it in comparison. After receiving the newly input log entry, the LCSMap is traversed and an LCSObject whose LCSseq is "PacketResponder 0 for block blk_6137960563260046065 terminating" is found. Then, the LCS is calculated as "PacketResponder for block terminating". When the length of the sequence is between 1/2 and 1 times the length of the input entry, it is judged that it belongs to the same log key, so it is merged, and the lineIds of this log entry is added to the lineIds attribute of the LCSObject.

Then, the obtained initial log template is manually filtered to obtain the final log event template. The log entries of the entire dataset are then processed to obtain log events of all log entries, which are used to form log event sequences and perform subsequent model training.

3.2. Anomaly Detection

HDFS logs are parsed to obtain the log template, and the event id of each log entry is obtained based on the log template. The value of some parameters can be used as an identifier for a specific execution sequence, such as block_id in HDFS logs. These identifiers can combine log entries together or unwrap log entries generated by concurrent processes to separate a single thread sequence. As shown in Table 3, the HDFS log event sequence in this article is generated based on block_id.

Table 3. The demo of HDFS log events sequences.

Sequence ID	**Log Events Sequences**
0	E5 E5 E5 E22 E11 E9 E11 E9 E11 E9 E26 E26 E26 E23 E23 E23 E21 E21 E21
1	E22 E5 E5 E5 E11 E9 E11 E9 E11 E9 E26 E26 E26
2	E22 E5 E5 E5 E26 E26 E26 E11 E9 E11 E9 E11 E9 E23 E23 E23 E23 E21 E21 E21
3	E22 E5 E5 E5 E11 E9 E11 E9 E11 E9 E26 E26 E26
4	E22 E5 E5 E5 E26 E26 E26 E11 E9 E11 E9 E11 E9 E4 E3 E3 E3 E4 E3 E4 E3 E3 E4 E3 E3 E23 E23 E23 E21 E21 E21
5	E5 E22 E5 E5 E11 E9 E11 E9 E11 E9 E26 E26 E26

System log detection will be performed on the log event sequence (shown in Table 3) obtained by the log parser. Assuming that K = $\{k_1, k_2, k_3, \ldots, k_n\}$ is a log event sequence transformed by a log block, each log key represents a log path command at a certain time, and the entire log event sequence reflects the sequential execution path of the log. To detect the entire sequence, it is necessary to check whether each log event is normal. Let k_i be one of the n K sequences, representing the log event to be detected. The model in DeepLog takes the influence of the forward sequence on k_i to determine whether it is abnormal. For HDFS logs, this article not only considers the impact of the forward sequence on the next log event but also considers the impact of the backward sequence on the previous log event and combines them to further determine whether the event is abnormal. Figure 4 summarizes the classification settings. Suppose t is the sequence id of the next log event, w1 is the set of h most recent log events in the previous sequence, and w2 is the set of h most recent log events in the subsequent sequence. In other words, w1 = $\{m_{t-h}, \ldots, m_{t-2}, m_{t-1}\}$, w2 = $\{m_{t+h}, \ldots, m_{t+2}, m_{t+1}\}$, where each mt is in K, and is the log event from the log entry et. The same log event in w1 and w2 may appear multiple times. The output of the training phase is two conditional probability distribution models Pr1 ($m_t = k_i \mid$ w1) and Pr2 = ($m_t = k_i \mid$ w2) for each $k_i \in K (i = 1, \ldots, n)$.

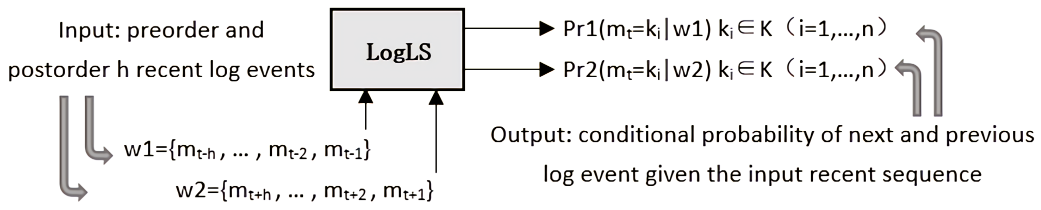

Input: preorder and postorder h recent log events
LogLS
Pr1(mt=ki|w1) ki∈K（i=1,...,n）
Pr2(mt=ki|w2) ki∈K（i=1,...,n）
w1={mt-h , ... , mt-2 , mt-1}
w2={mt+h , ... , mt+2 , mt+1}
Output: conditional probability of next and previous log event given the input recent sequence

Figure 4. An overview of log events anomaly detection model.

To extract context features and potential symmetry information [28] from sequence relationships, two long short-term memory networks (LSTMs) [29–31] are used in LogLS to train the preorder and postorder log event sequences. Each LSTM node has a hidden state h_{t-i} and a cell state C_{t-i}, both of which are passed to the next node to initialize its state. The purpose is to obtain the probability of the current log event k_i through the preorder and the subsequent log event k_i through the model and then set a probability limit (parameters g1, g2) to determine whether the current log event is anomalous.

The formula for forward propagation is:

Input gate:

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{1}$$

$$b_\iota^t = f(a_\iota^t) \tag{2}$$

Forget gate:

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{3}$$

$$b_\phi^t = f(a_\phi^t) \tag{4}$$

Output gate:

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^{t-1} \tag{5}$$

$$b_\omega^t = f(a_\omega^t) \tag{6}$$

In Formulas (1)–(6), x_t is the input at the current moment, s_c^{t-1} is the state of all cells at the last moment, b_h^{t-1} is the output of different LSTM memory blocks at the last moment, s_c^t is the state of all cells at the current moment, w is the weight of each gate, and f is the activation function.

The back propagation calculation formula is:

Input gate:

$$\delta_\iota^t = f'(a_\iota^t) \sum_{c=1}^{C} g(a_c^t) \epsilon_s^t \tag{7}$$

Forget gate:

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^{C} s_c^{t-1} \epsilon_s^t \tag{8}$$

Output gate:

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^{C} h(s_c^t) \epsilon_c^t \tag{9}$$

The process of backpropagation is actually the use of chain derivation to solve the gradient of each weight in the entire LSTM. In Formulas (7)–(9), δ_ι^t is the gradient of the input gate, δ_ϕ^t is the gradient of the forget gate, and δ_ω^t is the gradient of the output gate.

In the training phase, the normally executed log entries are used as the dataset to train the model. The purpose is to allow the model to fully learn the normal execution mode of the system log and avoid misjudgment as much as possible. Suppose a log event sequence is {E22, E5, E5, E5, E11, E9}. Given a window size of 2, the input preorder sequence and output label used to train the model are: {E22, E5→E5}, {E5, E5→E5}, {E5, E5→E11} and {E5, E11→ E9}. The input postorder sequence and output label are: {E9, E11→E5}, {E11, E5→E5}, {E5, E5→E5} and {E5, E5→E22}. The LogLS model is used to obtain the probability of the current log event based on the preorder and postorder sequences. The training phase needs to find an appropriate weight distribution so that the final output of the model produces the required labels and outputs them along with the input in the training dataset. In the training process, each input or output uses gradient descent to incrementally update these weights through loss minimization. In LogLS, the input includes h log event windows w1 and w2, and the output is the log event value immediately after w1 and the log event value before w2. We use categorical cross-entropy loss [32] for training. In the training phase, the length features and latent symmetry information of the log event sequence are counted for later use in the detection phase. After the model is trained, the verification set is used to adjust the parameters of the model to further obtain the optimal anomaly detection model parameter values.

The basic method of the detection phase is the same as the training phase, but a preliminary filtering process is added. For the newly added log event sequence, first a preliminary judgment is made based on the length features obtained in the training phase, and then how to combine the two models for anomaly detection is decided. The length of the tentative log sequence is represented by K, and the following three situations will occur in anomaly detection.

- When K < R (R represents the minimum length of the normal log event sequence counted during model training), the sequence is considered an anomaly sequence.
- When R < K < 2*W (W represents the final model training window size), only the previous log event sequence model is selected for anomaly detection. This involves a parameter g1 (g1 represents the first g1 digit of the log event probability predicted by the previous log event model).
- When K > 2*W, in this case, it is necessary to select the prelog event sequence model and the postlog event sequence model to predict together. It also needs to be subdivided, because the length of the last W of the log sequence cannot be used in the postsequence log event sequence model. Because there is no subsequent input, only the previous log event sequence model is selected at this time. Two parameters g2 and g3 are involved here (g2 represents the first g2 digits of the log event probability predicted according to the previous log event model, and g3 represents the first g3 digits of the log event probability predicted according to the subsequent log event model).

The log event sequence is input to the LogLS model, and the conditional probability of a log event to be detected is obtained. If the probability with the value of the parameters g1, g2, and g3 does not meet the parameter range, it means that it deviates from the normal log sequence and can be regarded as anomalous. Otherwise, other log events are continued to be judged until the entire log event sequence is determined. If no abnormality occurs, the log event sequence is normal.

For example, when the input log event sequence is {E22, E5, E5, E5, E11, E9, E11, E23} and the given window size is 3, the sequence length is 8. This situation belongs to the third situation mentioned above, and the {E5, E11} in the sequence are predicted using the preorder and postorder models. First, for {E5}, the input w1 = {E22, E5, E5}, w2 = {E11, E9, E11}. Suppose that the log event probability obtained according to w1 is {E5:0.8, E22:0.2} and that the event probability obtained according to w2 is {E5:0.6, E23:0.2, E9:0.2}. The parameter g2 is 1, and the parameter g3 is 2, indicating that the log event predicted by w1 is E5, and the log time predicted by w2 is E5 or E23. Because the actual log event is E5, it indicates that the current log event is normal. Because the detection result is normal, the detection work continues backward. If the actual log event is not in the two predicted results, it will be judged as an abnormal log event sequence. {E9, E11, E23} in this sequence cannot be predicted using the abnormal log event sequence model obtained in the subsequent sequence because there is no subsequent log sequence of the window size, so at this time, only the previous log event model is used to make the prediction. When the actual log event that needs to be predicted is {E23}, and the input w1 = {E11, E9, E11}, assuming that the event probability obtained according to this sequence {E5:0.6, E23:0.4}, the first g2 predicted result is taken as {E5}, which does not match the actual result. Therefore, it is regarded as an abnormal log event, and the sequence is finally judged as an abnormal sequence. The abnormal result is fed back to the user, and the user can judge whether a misjudgment is made. If a misjudgment occurs, the misjudgment sequence data are recorded, and the model is later adjusted through the update mechanism. If there was no misjudgment, the sequence was directly marked as abnormal.

3.3. Renewal Mechanism

Obviously, the training data may not cover all possible normal execution models. How to ensure the timeliness of the detection model and solve the emergence of new log execution models are problems that must be solved in system anomaly detection. For example, when the actual log event in the log event sequence is E12 and the predicted result according to w1 or w2 is E8, the event is regarded as abnormal. However, after manual inspection by the user, it is found that these two are normal log events. Therefore, LogLS provides users with a feedback mechanism [33] to relearn this new sequence using false positives to adjust its weight. When the sequence is entered again, it will be detected that both E12 and E8 will have the same probability and will be marked as normal.

4. Evaluation

LogLS is implemented using Keras [34] with TensorFlow [35] as the backend. In this section, we show evaluations of the overall performance of LogLS to show its effectiveness in finding anomalies from large system log data.

4.1. Dataset

This article uses the authoritative dataset commonly used in system log anomaly detection: the HDFS log dataset disclosed by Wei Xu et al. [5]. HDFS log datasets are generated by running Hadoop-based map-reduce jobs on Amazon EC2 nodes. It is marked by experts in the Hadoop field. The dataset contains 11,197,954 log entries, of which 16,838 log entries are abnormal, including events such as "write exceptions", accounting for approximately 2.9% of the total [36]. This dataset was constructed in 2009 and then widely used in the field of log anomaly detection. The dataset can be obtained in loghub. The specific information is shown in Table 4.

Table 4. Summary of the HDFS datasets.

System	#Time Span	#Data Size	#Nomalies	#Anomalies
HDFS	38.7 h	1.55 G	11,175,629	16,838

The HDFS dataset was eventually parsed into 575,059 log blocks, also known as log event sequences, of which 16,838 log blocks were marked as anomalies by Hadoop domain experts. In this paper, the HDFS dataset is divided into three parts, namely the training dataset, verification dataset and test dataset. The training dataset is used for the data samples for model fitting. Among them, there are only normal log event sequences, and 1% of the log event sequence dataset is selected, i.e., 4855 normal log event sequences. The validation dataset is a set of samples set aside separately during the model training process. It can be used to adjust the hyperparameters of the model and to make a preliminary assessment of the model's capabilities. Among them, there are both normal log event sequences and abnormal log events, and 1/3 of the remaining log event sequences are selected, i.e., 166,009 normal log time sequences and 5051 abnormal log event sequences. The test dataset is used to evaluate the final detection ability of the model, but cannot be used as a basis for algorithm selection such as parameter tuning and feature selection. It contains both normal log event sequences and abnormal log event sequences, which are the remaining 2/3 log event sequence [33,37]. The specific division is shown in Table 5.

Table 5. Set up of HDFS log datasets (unit: sequence).

Log Dataset	Number of Sessions			n: Number of Log Keys
	Training Data	**Validation Data**	**Test Data**	
HDFS	4855 normal	166,009 normal; 5051 anormal	387,357 normal; 11,787 anormal	29

4.2. Evaluation Metrics

To evaluate the effectiveness of the model, in addition to the number of false positives (FP) and false negatives (FN), we also use standard metrics, such as accuracy, precision, recall and F1-measure. Accuracy represents the percentage of logs that are correctly classified in the total logs. Precision represents the proportion of true anomalies among the detected anomalies. Recall represents the percentage of detected anomalies in the total anomalies in the dataset. The F1-measure is the weighted harmonic average of precision and recall and is a comprehensive evaluation index [38].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

$$F1 - measure = \frac{2 * Precison * Recall}{Precision + Recall} \tag{13}$$

In Formulas (10)–(13), TP is the number of normal classes predicted for normal samples, FP is the number of normal classes predicted for abnormal samples, and FN is the number of abnormal classes predicted for normal samples [39].

4.3. Result Analysis

We compare LogLS with four anomaly detection methods, namely PCA, IM, N-gram and DeepLog. Through the comparison of each evaluation data, the detection performance of this model in log anomaly detection is obtained. By default, we use the following parameter values for LogLS: g1 = 13, g2 = 4, g3 = 2, h = 10, L = 2, S = 64, and E = 300. g1, g2, and g3 are the same type of parameters g, and g determines whether the predicted log event is normal (the log event that appears next is considered normal among the g log events with the highest predicted probability). h is the window size used for training and detection, and L and S represent the number of layers in LogLS and the number of memory units in an LSTM block, respectively. E is the number of epochs to be trained, where each epoch is a single training iteration of all batches in the forward and backward propagation. If the number of epochs is too small, underfitting may occur, and if the number is too large, overfitting may occur.

Table 6 shows the number of false positives and false negatives for each method on the HDFS data (all data except the training set), and the accuracy rate. The false positives and false negatives of LogLS have a low level, and the accuracy rate of 99.84% is also the highest among the five methods. Figure 5 shows the experimental results of different methods on the HDFS dataset. Although the precision of the traditional PCA on the HDFS dataset is 0.98, the recall and F1-measure are relatively low, only 0.67 and 0.79, respectively. Among the three methods of IM, N-gram and DeepLog, DeepLog performs best. The precision, recall and F1-measure are 0.95, 0.96 and 0.96, respectively. However, we can also see that the LogLS method is better than several other methods as a whole. The three evaluation metrics reached 0.96, 0.98 and 0.97, indicating that this method has advantages.

Table 6. Number of FPs, FNs and Accuracy on HDFS log.

	PCA	**IM**	**N-Gram**	**DeepLog**	**LogLS**
False positive(FP)	277	2122	1360	833	657
False negative(FN)	5429	1226	739	615	280
Accuracy	99.00%	99.41%	99.63%	99.75%	99.84%

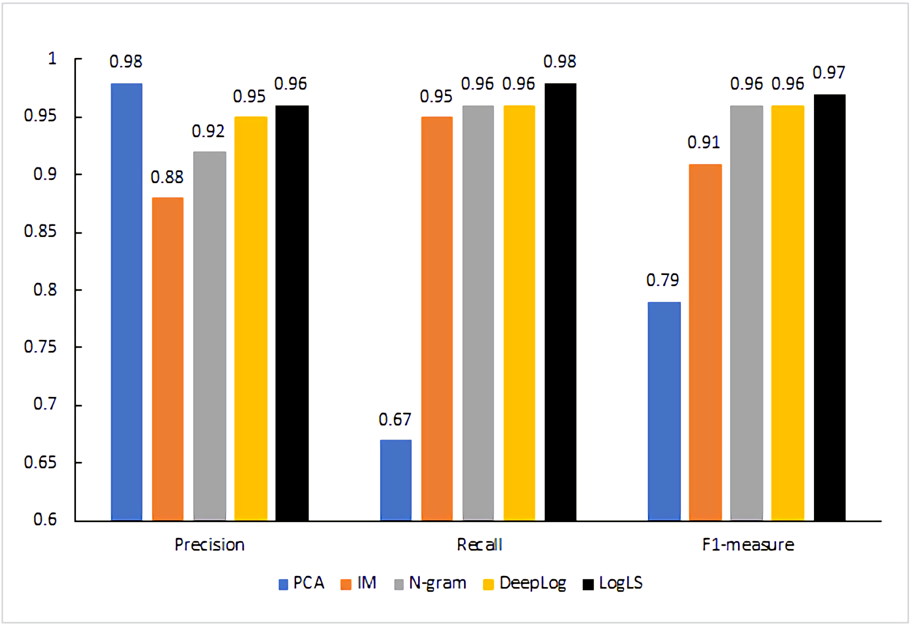


Figure 5. Evaluation on HDFS log.

4.4. Parameter Analysis

The performance of the model is tested by adjusting each parameter in the LogLS model. This article uses the controlled variable method to carry out the parameter adjustment experiment of LogLS. When one parameter is studied, the other parameters are controlled and remain unchanged. The adjusted parameters include g1, g2, g3, h, L, S, and E. In each experiment, only the value of one parameter is changed; the remaining parameters remain at the default values. Of course, g1, g2, and g3 belong to a class of parameters, so these three values are changed as variables, and the rest remain unchanged. After observation, the arithmetic sequence method is used to determine the optimal value of class G parameters. The value range of g1 is {5,7,9,11,13,15}. The value range of g2 and g3 is {1,3,5,7,9,11,13}. After adjusting these three parameters, a total of 294 groups of data are generated. The experimental data cannot be fully displayed in the chart, so Figure 6 shows 49 groups. The graph is obtained by assuming g1 = 13 and changing the values of g2 and g3. Figure 6 shows the changes in the three values of precision, recall and F1-measure when g2 and g3 change. The abscissa represents g2, and the ordinate represents g3. Figure 6 includes three sets of small graphs. Figure 6a shows the change in the precision value after changing these two parameters, Figure 6b shows the change in the recall value, and Figure 6c shows the change in the F1-measure.

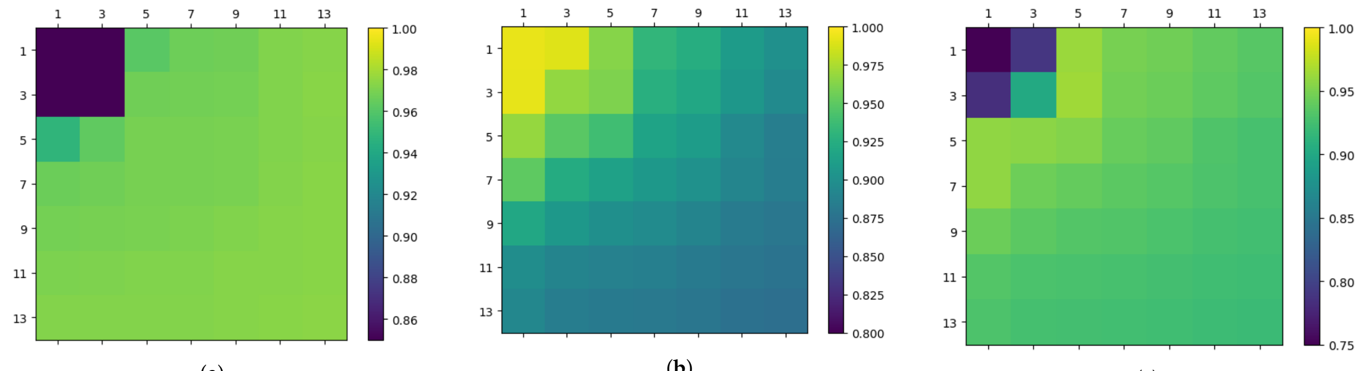


Figure 6. Performance comparison chart of changing parameters g2 and g3. (**a**) Precision. (**b**) Recall. (**c**) F1-measure.

Figure 6 shows that when the F1-measure of the model is the best, the values of g2 and g3 are 5 and 3, respectively. The detailed evaluation indicators corresponding to these two parameters are separately compared. The results for when g3 = 3 and only g2 is changed are shown in Table 7 and Figure 7.

Table 7. g2 size in LogLS.

g2 Size	Precision	Recall	F1-Measure
1	0.6476	**0.9923**	0.7837
3	0.8443	0.9673	0.9016
5	0.9677	0.9612	**0.9644**
7	0.9683	0.9260	0.9467
9	0.9689	0.9198	0.9437
11	0.9716	0.9060	0.9376
13	**0.9737**	0.8727	0.9204

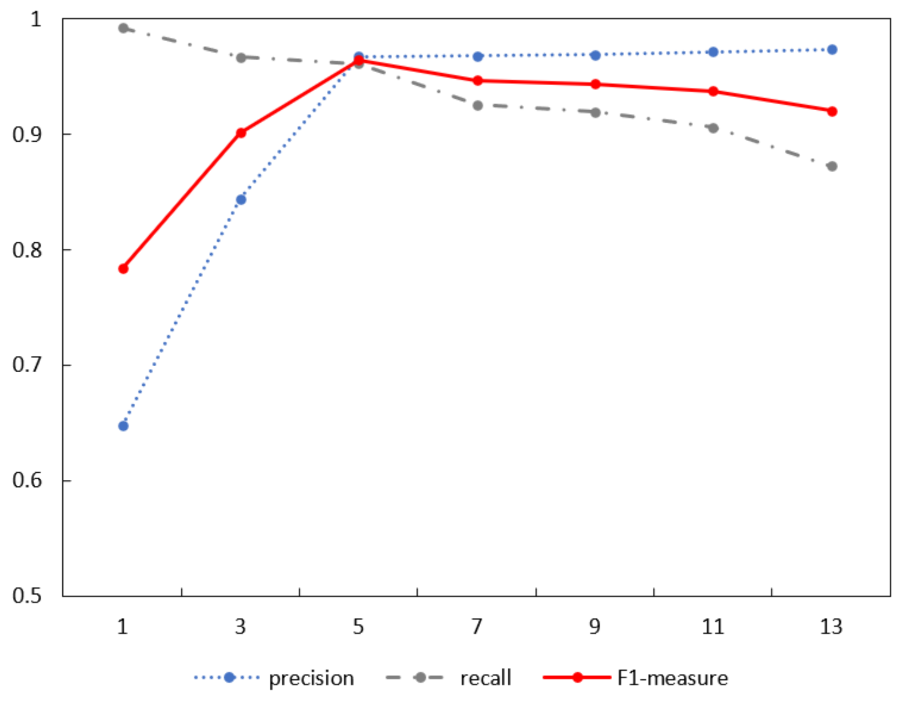


Figure 7. Performance comparison chart of changing parameter g2.

As seen from Table 7 and Figure 7, precision increases with increasing g2; the maximum precision value in the table is 0.9737. Recall decreases with increasing g2, and the maximum value of recall in the table is 0.9923. The value of the F1-measure first increases and then decreases as the value of g2 increases. When g2 = 5, the value of the F1-measure in the table reaches a maximum value of 0.9644, and at this time, the values of precision, recall and F1-measure are relatively balanced and high. Therefore, it is preliminarily confirmed that the optimal parameter value of g2 is 5.

Now we parameter g2 to 5 and use default values for other parameters. By changing the parameter value of g3, the changes of the three values of precision, recall and F1-measure are obtained. The results are shown in Table 8 and Figure 8.

Table 8. g3 size in LogLS.

g3 Size	Precision	Recall	F1-Measure
1	0.9619	**0.9636**	0.9627
3	0.9677	0.9612	**0.9644**
5	0.9693	0.9386	0.9537
7	0.9690	0.9147	0.9410
9	0.9699	0.8996	0.9334
11	0.9716	0.8878	0.9278
13	**0.9721**	0.8822	0.9250

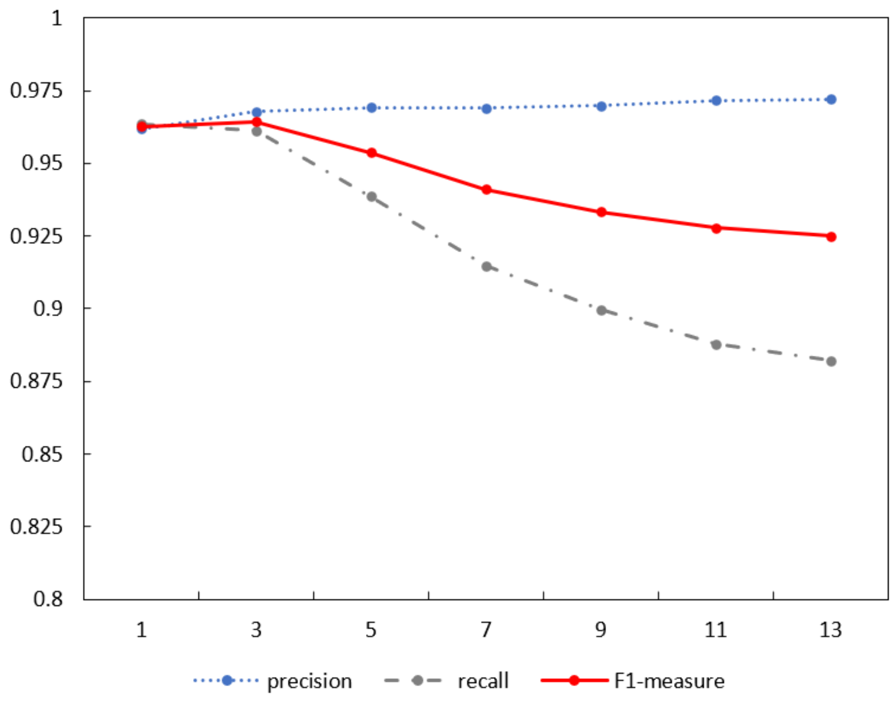


Figure 8. Performance comparison chart of changing parameter g3.

It can be seen from Table 8 and Figure 8 that precision increases with the increase of g3, recall decreases with the increase of g3, and that the F1-measure first increases and then decreases with the increase of g3. The F1-measure obtains the maximum value when g3 = 3, and the three values of precision, recall and F1-measure are relatively balanced and high. Therefore, the tuning parameter temporarily confirms that the optimal parameter value of g3 is 3. It can be concluded that the parameter values of g2 and g3 are temporarily 5 and 3, respectively. Then, the parameters of g1 are adjusted according to g2 and g3. As the value of g1 changes, the three values of precision, recall, and F1-measure are obtained. The results are shown in Table 9 and Figure 9.

Table 9. g1 size in LogLS.

g1 Size	Precision	Recall	F1-Measure
5	0.8919	**0.9966**	0.9414
7	0.9287	0.9830	0.9551
9	0.9464	0.9673	0.9567
11	0.9623	0.9654	0.9638
13	0.9677	0.9612	**0.9644**
15	**0.9690**	0.9339	0.9511

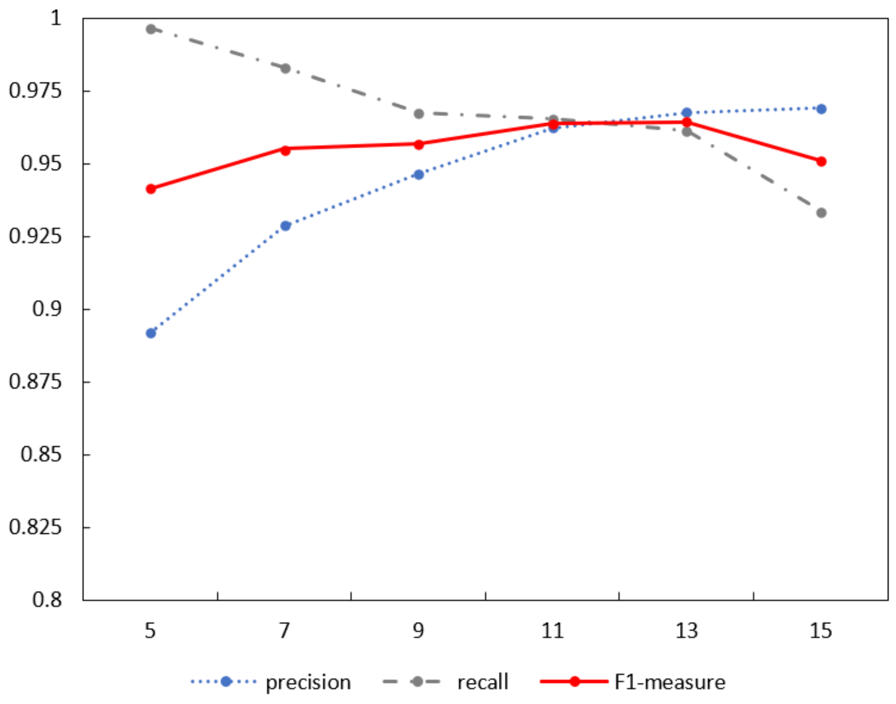


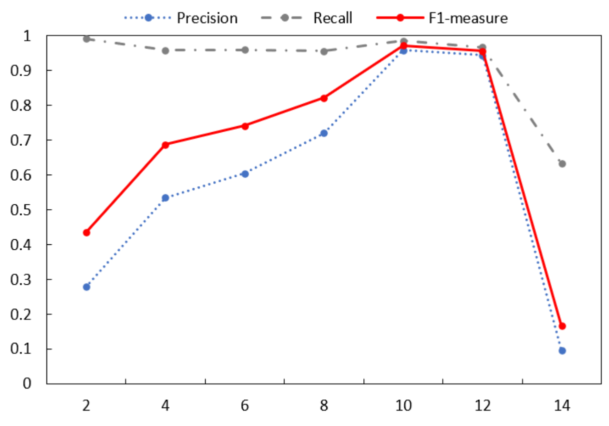
Figure 9. Performance comparison chart of changing parameter g1.

It is determined from Table 9 and Figure 9 that the parameter g1 can be temporarily set to 13. Combining the parameter results obtained above, it can be concluded that when g1 = 13, g2 = 5 and g3 = 3, the overall model prediction results are ideal. However, this parameter is adjusted by an arithmetic sequence, which is not complete. Moreover, it is found that the changes in precision, recall and F1-measure are correlated with g1, g2 and g3. Therefore, the adjacent parameter values are compared. The value range of g1 is {12,13,14}, while g2 is {4,5,6} and g3 is {2,3,4}. As shown in Table 10, the final values of parameters g1, g2 and g3 are 13, 4 and 2, respectively. At this time, precision, recall and F1-measure all perform well, being 0.9586, 0.9856 and 0.9719, respectively.

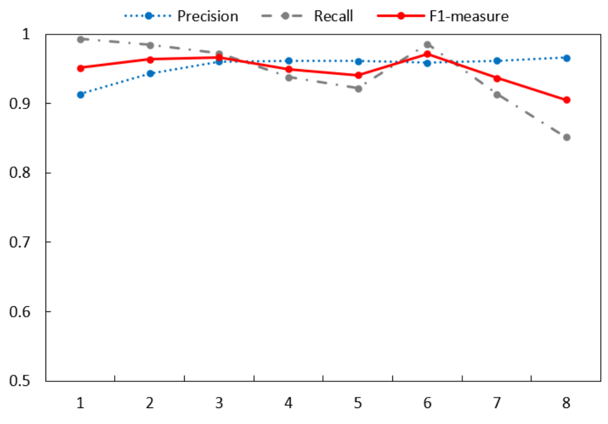
We then studied the impact of various other parameters on the detection performance during LogLS training, including h, L, S, and E. The results obtained are shown in Figure 10. In each experiment, we change the value of one parameter while using the default values of other parameters. Graph (a) in Figure 10 shows the performance change of the model by changing the parameter h. Because the LSTM network needs long dependence, within a certain range, when the selected window is larger, its performance is more obvious, but when it breaks this range, the performance drops sharply. Figure 10b shows the performance change of the model when the parameter E is varied. Within a certain range of training iterations, as the number of training iterations increases, the performance of the model becomes stronger, but the number of iterations is too large, and the performance of the model becomes weaker. Figure 10c shows the performance change of the model as the parameter L is changed. The change in the number of layers in LogLS has a relatively stable effect on model performance. Figure 10d shows the performance change of the model with changing parameters. When the number of memory units in an LSTM block is 64, the model performance is the best. In summary, of all the experimental results, it is found that the LogLS model is relatively stable when various parameters are adjusted reasonably, and a single change in parameters or a combination of adjustments has little effect on the performance of the model.

Table 10. Adjacent parameter.

g1	g2	g3	Precision	Recall	F1-Measure
12	4	2	0.9558	0.9856	0.9705
12	4	3	0.9596	0.9634	0.9615
12	4	4	0.9649	0.9586	0.9618
12	5	2	0.9609	0.9634	0.9621
12	5	3	0.9648	0.9616	0.9632
12	5	4	0.9664	0.9561	0.9612
12	6	2	0.9646	0.9442	0.9543
12	6	3	0.9653	0.9372	0.9511
12	6	4	0.9664	0.9333	0.9495
13	**4**	**2**	**0.9586**	**0.9856**	**0.9719**
13	4	3	0.9624	0.9630	0.9627
13	4	4	0.9678	0.9582	0.9630
13	5	2	0.9637	0.9630	0.9634
13	5	3	0.9677	0.9612	0.9644
13	5	4	0.9693	0.9555	0.9623
13	6	2	0.9675	0.9438	0.9555
13	6	3	0.9683	0.9368	0.9523
13	6	4	0.9693	0.9325	0.9506
14	4	2	0.9602	0.9778	0.9689
14	4	3	0.9640	0.9553	0.9596
14	4	4	0.9694	0.9473	0.9583
14	5	2	0.9654	0.9553	0.9603
14	5	3	0.9694	0.9531	0.9612
14	5	4	0.9709	0.9446	0.9576
14	6	2	0.9693	0.9361	0.9524
14	6	3	0.9699	0.9258	0.9473
14	6	4	0.9710	0.9212	0.9454

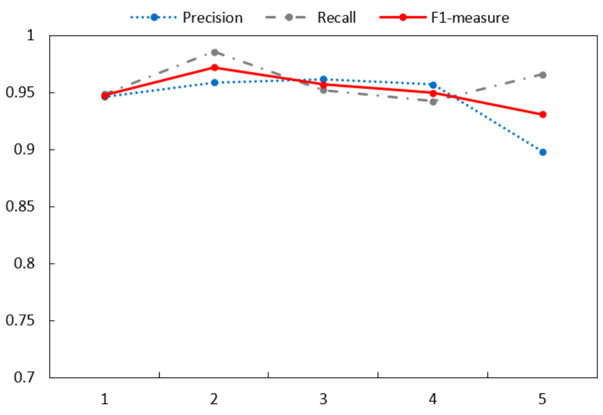


(**a**)

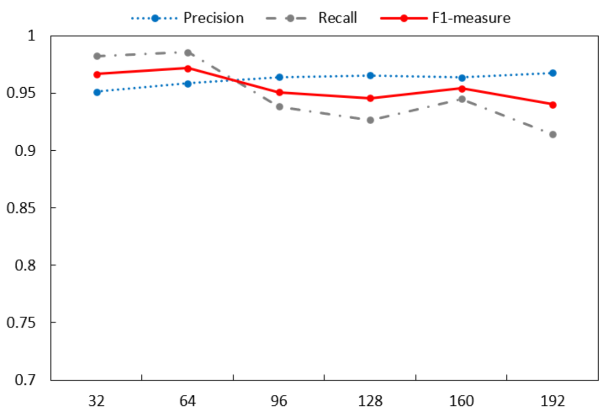


(**b**)

Figure 10. *Cont.*



(**c**)



(**d**)

Figure 10. LogLS performance with different parameters. (**a**) Window size: h. (**b**) Number of epochs: E. (**c**) Number of layers: L. (**d**) Number of memory units: S.

5. Online Update and Training of LogLS

Although the method in this paper has achieved good performance in the HDFS log anomaly detection experiment, problems may occur when dealing with more irregular logs (such as system logs). Many log keys only appear in a certain period, so the training set may not contain all the normal log keys, which will cause false predictions. The model update module can effectively solve this problem, which adjusts the weight parameters of the model in real time based on the online false-positive results. This section sets up a comparative experiment of model updates to verify its effectiveness.

The model update method adopts incremental update, and only uses false positives to update the model. Suppose h = 3, the input historical sequence is $\{k_1, k_2, k_3\}$, and LogLS predicts that the next log key is k_2 with probability 1, and the log key in the actual sequence is k_3, then the model marks it as an anomaly. However, after manual detection, it is known that this is a false positive. LogLS can use $\{k_1, k_2, k_3 \rightarrow k_3\}$ to update the weights of its model, therefore learning this new log pattern. The next time enter $\{k_1, k_2, k_3\}$, LogLS can output both k_2 and k_3 with updated probabilities. This method does not need to re-update LogLS from scratch. Updating the model with new experimental data. The weights of model are adjusted by minimizing the error between experimental output and actual observations from false-positive cases.

The log dataset selected in this experiment is the system log of the 708 M Blue Gene/L supercomputer [40], also known as the BGL log. This log is different from the HDFS log and is chosen because many logs in this dataset only appear in specific events, so the training set may not contain all the normal execution paths and log keys.

The log dataset contains 4,747,963 logs, of which 348,460 are marked as anomalies, including alarm and nonalarm messages identified by alarm category tags. In the first column of the log, "-" means nonalarm messages, while others are alarm messages. The label information facilitates alarm detection and prediction research. It has been used in many studies, such as log parsing, anomaly detection and failure prediction.

When using BGL to generate a log sequence, it is different from the HDFS method. For this type of log without a unique identifier, we use a sliding window to obtain the log sequence. In this experiment, we set the sliding window size to 40 and obtained 214,475 normal BGL log sequences and 20,657 abnormal BGL log sequences. We take 1% of normal BGL log entries as the training set, the remaining 1/3 as the validation set (if needed), and 2/3 as the test set. The model update uses the trained model to detect anomalies. Whenever the detected result is found to be a false positive, the input and output sequence of the result is used to update the model. Due to the characteristics of the BGL log, the settings on some parameters continue to use the values in the DeepLog method. In this experiment, g1 = 10, g2 = 4, g3 = 2, h = 3, L = 1, S = 64 and E = 300.

According to Table 11 and Figure 11, we have proved the effectiveness of the model update algorithm through experiments. After the model update, the detection accuracy of the model is improved. The model updating mechanism improves the detected F1 measure value from 29.89% to 80.94%, and the accuracy is also improved by 50.57%. This shows that updating the model can solve the situation where the training set cannot cover all normal execution paths.

Table 11. Evaluation on Blue Gene/L Log.

	No Update Model	**Update Model**
FP	64,440	6416
FN	31	47
Precision	17.58%	68.15%
Recall	99.78%	99.66%
F1-measure	29.89%	80.94%

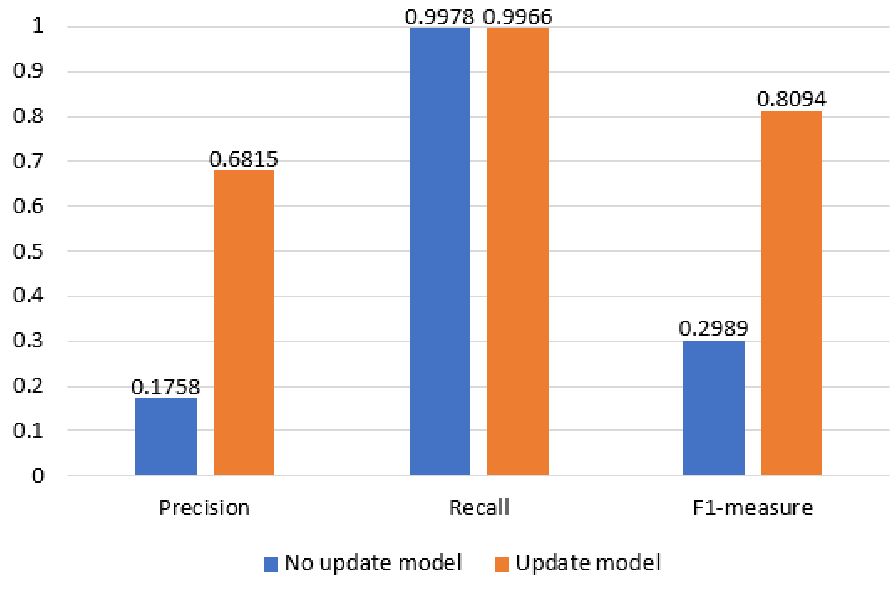


Figure 11. Evaluation on Blue Gene/L Log.

6. Conclusions

This paper proposes a system log anomaly detection method based on dual LSTM, which makes full use of the context of log events in log sequences. Referring to the Spell log parsing method, a filtering operation is added to obtain the log event template list more accurately, effectively solving the problem of inconsistent log structure in the traditional anomaly detection method. According to the log event context and latent symmetry information, we build two LSTM models from these two perspectives and make them cooperate with each other to detect log anomalies. To solve the problem that the LSTM model cannot handle unknown logs, we also added an updated model mechanism to improve the performance of the model in detecting new log rules. For logs with unique identifiers, such as HDFS logs, we can form log sequences based on unique identifiers. For logs without unique identifiers, such as BGL logs, we can select fixed windows to form log sequences. The experimental results show that the proposed method performs well on HDFS large log datasets, and the accuracy, recall rate and F1-measure are better than the current cutting-edge log anomaly detection methods. In addition, this paper fully analyzes the influence of parameter changes on the model performance, and verifies the effectiveness of the model update strategy, which has significant performance in system log anomaly detection, and is of great significance to system anomaly detection and optimization of model parameters.

In future work, the model will be improved to make it suitable not only for anomaly detection in the execution mode of the system log, but also for detection for each parameter in the log. We find a better way to solve the problem that the LSTM model cannot predict the log execution path that does not appear.

Author Contributions: Conceptualization, Y.C. and N.L.; methodology, Y.C., N.L. and D.L.; writing—original draft preparation, Y.C.; writing—review and editing, Y.C. and D.L.; project administration, N.L.; funding acquisition, N.L. All authors read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Innovation Environment Construction Special Project of Xinjiang Uygur Autonomous Region under Grant PT1811, and in part by the Key Grant Project of the National Social Science Fund of China(NSFC) under Grant 20&ZD293.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are openly available at ref. [15].

Acknowledgments: We thank the Innovation Environment Construction Special Project of Xinjiang Uygur Autonomous Region and NSFC for funding this research. We thank the anonymous reviewers for their contribution to this paper.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LSTM	Long Short-Term Memory Network
bi-LSTM	Bi-directional Long Short-Term Memory
RNN	Recurrent neural network
PCA	Principal component analysis
AST	Abstract syntax code
LCS	Longest common subalgorithm
BGL	Blue Gene/L
IM	Invariant Mining
MIT	Massachusetts Institute of Technology
HDFS	HDFS distributed file system

References

1. Fotiadou, K.; Velivassaki, T.H.; Voulkidis, A.; Skias, D.; De Santis, C.; Zahariadis, T. Proactive Critical Energy Infrastructure Protection via Deep Feature Learning. *Energies* **2020**, *13*, 2622. [CrossRef]
2. Wang, B.; Ying, S.; Cheng, G.; Li, Y. A log-based anomaly detection method with the NW ensemble rules. In Proceedings of the 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), Macau, China, 11–14 December 2020; pp. 72–82.
3. Rouillard, J.P. Real-time Log File Analysis Using the Simple Event Correlator (SEC). In Proceedings of the Conference on Systems Administration, Atlanta, GA, USA, 14–19 November 2004; pp. 133–150.
4. Kim, C.; Jang, M.; Seo, S.; Park, K.; Kang, P. Intrusion Detection Based on Sequential Information Preserving Log Embedding Methods and Anomaly Detection Algorithms. *IEEE Access* **2021**, *9*, 58088–58101. [CrossRef]
5. Xu, W.; Huang, L.; Fox, A.; Patterson, D.; Jordan, M. Detecting Large-Scale System Problems by Mining Console Logs. In Proceedings of the 26 th International Conference on Machine Learning, Haifa, Israel, 21–24 June 2010; pp. 37–46.
6. Lou, J.G.; Fu, Q.; Yang, S.; Xu, Y.; Li, J. Mining invariants from console logs for system problem detection. In Proceedings of the 2010 USENIX Annual Technical Conference, Boston, MA, USA, 23–25 June 2010; pp. 1–14.
7. Lou, J.G.; Fu, Q.; Yang, S.; Li, J.; Wu, B. Mining program workflow from interleaved traces. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 25–28 July 2010; pp. 613–622.
8. Lin, Q.; Zhang, H.; Lou, J.G.; Zhang, Y.; Chen, X. Log clustering based problem identification for online service systems. In Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, USA, 14–22 May 2016; pp. 102–111.

9. Yang, L.; Chen, J.; Wang, Z.; Wang, W.; Jiang, J.; Dong, X.; Zhang, W. Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 22–30 May 2021; pp. 1448–1460.
10. Bodik, P.; Goldszmidt, M.; Fox, A.; Woodard, D.B.; Andersen, H. Fingerprinting the datacenter: Automated classification of performance crises. In Proceedings of the ACM EuroSys Conference on Computer Systems, EuroSys' 10, New York, NY, USA, 13 April 2010; pp. 111–124.
11. Yang, R.; Qu D.; Zhu S.; Qian Y.; Tang Y. Anomaly detection for log sequence based on improved temporal convolutional network. *Comput. Eng.* **2020**, *46*, 50–57.
12. Phyo, P.P.; Byun, Y.C. Hybrid Ensemble Deep Learning-Based Approach for Time Series Energy Prediction. *Symmetry* **2021**, *13*, 1942. [CrossRef]
13. Wang, M.; Xu, L.; Guo, L. Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning. In Proceedings of the 2018 4th International Conference on Frontiers of Signal Processing (ICFSP), Poitiers, France, 24–27 November 2018; pp. 140–144.
14. Xia, B.; Bai, Y.; Yin, J.; Li, Y.; Xu, J. LogGAN: A Log-level Generative Adversarial Network for Anomaly Detection using Permutation Event Modeling. *Inf. Syst. Front.* **2021**, *23*, 285–298. [CrossRef]
15. Du, M.; Li, F; Zhang, G.; SriKumar, V. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the Acm Sigsac Conference on Computer & Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
16. Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; et al. Robust Log-Based Anomaly Detection on Unstable Log Data. In Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), Tallinn, Estonia, 26–30 August 2019; pp. 807–817.
17. Nedelkoski, S.; Bogatinovski, J.; Acker, A.; Cardoso, J.; Kao, O. Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs. In Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM), Sorrento, Italy, 17–20 November 2020; pp. 1196–1201.
18. Du, M.; Li, F. Spell: Online Streaming Parsing of Large Unstructured System Logs. *IEEE Trans. Knowl. Data Eng.* **2018**, *31*, 2213–2227. [CrossRef]
19. Du, M.; Li, F. Spell: Streaming Parsing of System Event Logs. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM), Barcelona, Spain, 12–15 December 2016; pp. 859–864.
20. Yu, X.; Joshi, P.; Xu, J.; Jin, G.; Zhang, H.; Jiang, G. CloudSeer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs. In Proceedings of the Twenty-First International Conference, New York, NY, USA, 25 March 2016; pp. 489–502.
21. Bao, L.; Li, Q.; Lu, P.; Lu, J.; Ruan, T.; Zhang, K. Execution Anomaly Detection in Large-scale Systems through Console Log Analysis. *J. Syst. Softw.* **2018**, *143*, 172–186. [CrossRef]
22. Fu, Q.; Lou, J.G.; Wang, Y.; Li, J. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, 6–9 December 2009; pp. 149–158.
23. Dai, H.; Li, H.; Chen, C.S.; Shang, W.; Chen, T.H. Logram: Efficient Log Parsing Using n-Gram Dictionaries. *IEEE Trans. Softw. Eng.* **2020**. [CrossRef]
24. He, P.; Zhu, J.; Zheng, Z.; Lyu, M.R. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In Proceedings of the IEEE International Conference on Web Services, Honolulu, HI, USA, 25–30 June 2017; pp. 33–40.
25. Zhu, J.; He, S.; Liu, J.; He, P.; Lyu, M.R. Tools and Benchmarks for Automated Log Parsing. In Proceedings of the Tools and Benchmarks for Automated Log Parsing, Montreal, QC, Canada, 1 May 2019; pp. 121–130.
26. He, P.; Zhu, J.; He, S.; Li, J.; Lyu, M.R. An evaluation study on log parsing and its use in log mining. In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Toulouse, France, 28 June–1 July 2016; pp. 654–661.
27. Alanis, A.Y.; Sanchez, O.D.; Alvarez, J.G. Time Series Forecasting for Wind Energy Systems Based on High Order Neural Networks. *Mathematics* **2021**, *9*, 1075. [CrossRef]
28. Nandanwar, A.K.; Choudhary, J. Semantic Features with Contextual Knowledge-Based Web Page Categorization Using the GloVe Model and Stacked BiLSTM. *Symmetry* **2021**, *13*, 1772. [CrossRef]
29. Ian G.; Yoshua B.; Aaron C. *Deep Learning*, 1st ed.; MIT Press: Cambridge, MA, USA, 2016; pp. 162–481.
30. Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
31. Understanding LSTM Networks. Available online: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 14 October 2020).
32. Rusiecki A. Trimmed categorical cross-entropy for deep learning with label noise. *Electron. Lett.* **2019**, *55*, 319–320. [CrossRef]
33. Oprea, A.; Li, Z.; Yen, T.F.; Chin, S.H.; Alrwais, S. Detection of Early-Stage Enterprise Infection by Mining Large-Scale Log Data. In Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, Brazil, 22–25 June 2015; pp. 45–56.
34. Keras. Available online: https://github.com/keras-team/keras (accessed on 2 October 2020).

35. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
36. Xu, W.; Huang, L.; Fox, A.; Patterson, D.; Jordan, M. Online System Problem Detection by Mining Patterns of Console Logs. In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, 6–9 December 2009; pp. 588–597.
37. Prewett, J.E. Analyzing cluster log files using Logsurfer. In Proceedings of the Annual Conference on Linux Clusters, 2003; pp. 83–95. Available online: https://www.semanticscholar.org/paper/Analyzing-cluster-log-files-using-Logsurfer-Prewett/d9a2a773348e6dc1c0bef303cf188145267bd8c1 (accessed on 2 February 2022).
38. Wang, Y.; Liu, P.; Wang, B. Research on system log anomaly detection based on deep learning. *Chin. J. Netw. Inf. Secur.* **2019**, *5*, 105–118.
39. Zhang, L.; Lu, R.; Liu, P. System Anomaly Detection Method Based on Bidirectional LSTM. *Comput. Appl. Softw.* **2020**, *12*, 303–309+339.
40. Oliner A.; Stearley J. What supercomputers say: A study of five system logs. In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Edinburgh, UK, 25–28 June 2007; pp. 575–584.