Available Online: 11 April 2025 DOI: 10.54254/2977-3903/2025.22029

A distributed system log anomaly detection framework based on graph-chain architecture

Mingyu Hu[1], Chen Song[1], Tao Xie[1]*

[1]College of Computer Science, National University of Defense Technology, Changsha, China

*Corresponding Author. Email: naturl_sty@163.com

Abstract. Enterprise alliances and government agencies typically deploy distributed systems across subsidiaries and subordinate departments, where each system node collaborates via a network to present a unified interface to users. Addressing the challenges of log data dispersion, heterogeneity, and lack of credibility in distributed systems, this paper proposes a log anomaly detection framework based on a graph-chain architecture. The framework leverages the sequence analysis capabilities of a distilled Transformer model to detect anomalies in system logs at each node. Finally, by integrating blockchain smart contracts, it ensures tamper resistance and traceability. Experimental results demonstrate that the proposed framework achieves an anomaly detection accuracy of 99.6%, surpassing traditional methods.

Keywords: distributed systems, blockchain, anomaly detection, artificial intelligence

1. Introduction

With the widespread application of distributed systems in finance, the Internet of Things (IoT), and other fields, system log data exhibits several characteristics: Multi-source heterogeneity-log formats vary across nodes; Temporal correlation-service invocation chains form complex spatiotemporal dependencies; Security vulnerability-centralized log servers are susceptible to tampering attacks (e.g., Log4j vulnerability leading to log injection). System logs record detailed information on computing events generated by computer systems, playing a crucial role in modern anomaly detection.

Traditional detection methods based on rule matching or isolated analysis suffer from high false positive rates and a lack of cross-node correlation. Emerging large-scale models, while effective, are difficult to deploy on nodes with limited computing resources due to their extensive parameter sizes. Various traditional machine learning models have been proposed to identify anomalies from log messages. These methods extract useful features from log messages and employ machine learning algorithms to analyze log data. However, due to data imbalance issues, training a binary classifier to detect anomalous log sequences is impractical. Therefore, numerous unsupervised learning models, such as Principal Component Analysis (PCA) [1] and one-class models like Support Vector Machines (SVM) [2, 3], have been widely used for anomaly detection. Nevertheless, traditional machine learning models, such as SVM, struggle to capture the temporal information of discrete log messages. In recent years, Recurrent Neural Networks (RNNs) in deep learning have gained widespread application in log anomaly detection due to their proficiency in processing sequential data. Du et al. [6] proposed DeepLog, a deep neural network model using Long Short-Term Memory (LSTM) to model system logs as natural language sequences. This enables DeepLog to learn log patterns from normal executions automatically and detect anomalies when log patterns deviate from expected sequences. Zhang et al. [7] introduced LogRobust, an attention-based Bi-LSTM model that detects anomalies by capturing contextual information in log sequences and autonomously learning the importance of different log events. This approach allows the model to recognize and process unstable log events and sequences. Le et al. [8] proposed NeuralLog, a novel log-based anomaly detection method that eliminates the need for log parsing. NeuralLog extracts the semantic meaning of raw log messages and represents them as semantic vectors. These representation vectors are then used for anomaly detection through a Transformer-based classification model, which effectively captures contextual information from log sequences. Experimental results demonstrate that this method accurately understands log message semantics and achieves precise anomaly detection.

However, these models still face significant challenges in practical applications: Although RNNs can capture sequence dynamics through temporal recurrence mechanisms, their unidirectional information propagation limits the ability to integrate bidirectional contextual semantics within a single log entry. This is particularly crucial for detecting malicious attacks, where

complete bidirectional context is more informative than historical data alone. Existing RNN training paradigms are typically based on an autoregressive prediction task, where normal behavior patterns are established by predicting subsequent entries from preceding logs. While this design captures intra-sequence correlations, it may overlook global common features when anomalies disrupt sequence coherence. Furthermore, optimizing solely for adjacent log predictions fails to systematically model the deeper statistical regularities of normal sequences. In Nakamoto consensus protocols (e.g., Bitcoin), the linear nature of chain-based ledgers constrains block generation speed, leading to low transaction throughput (e.g., Bitcoin processes approximately seven transactions per second). This limitation makes it unsuitable for high-frequency logging scenarios and fails to fully utilize the increasing network bandwidth of nodes. Existing blockchain protocols (e.g., Bitcoin) feature fixed block intervals and a single-chain structure, preventing dynamic optimization of network bandwidth usage. Given the widespread improvement in node bandwidth, static protocol designs lead to resource wastage.

To address these challenges in distributed system log anomaly detection, this paper proposes a graph-chain architecture-based distributed system log anomaly detection framework. We employ the DistilBERT model [4] to capture patterns in normal log sequences. Unlike traditional RNNs, Transformers eliminate sequential dependency constraints and directly compute relationships between any two positions within a sequence through self-attention, enabling global context capture. We aim for each log entry's contextual embedding to encompass the entire log sequence's information. Additionally, leveraging the ORIC graph-chain protocol [5] with its specialized mining mechanism allows nodes to concurrently generate multiple blocks within a single block time, thereby increasing block generation speed. This enhancement boosts maximum transaction throughput and improves bandwidth utilization. Experimental results indicate that the proposed log anomaly detection framework achieves superior performance on datasets. Furthermore, due to model compression, it can be deployed on lightweight system nodes efficiently.

2. Fundamental concepts

2.1. Transformer model

The Transformer model is an architecture based on the self-attention mechanism, consisting of an input module, encoder, decoder, and classifier. During the input stage, the input sequence is first tokenized, then each token is encoded into an embedding vector, and positional information is added to the vector. In some tasks, additional sentence-level information (such as whether a sentence is a question or an answer) may also be incorporated. Since the Transformer lacks a recurrent structure, positional encoding is introduced to represent the position of each element within the sequence. A common approach to positional encoding utilizes sine and cosine functions, defined as follows:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{1}$$

$$PE_{(pos,2i+i)} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{2}$$

After integrating the positional and sentence information, the input matrix is processed by the encoder. In the model used in this paper, the encoder consists of a multi-head attention mechanism and a feedforward neural network. In the self-attention mechanism, a set of matrices W_Q, W_K, W_V forms an attention head. These three weight matrices correspond to the Query matrix W_Q, the Key matrix W_K and the Value matrix W_V. The input word embeddings are multiplied by these matrices to obtain the query vector q_i, the key vector k_i and the value vector v_i. The attention values can be computed using the following formula:

$$Attention(q_i, k_i, v_i) = softmax\left(\frac{q_i k_i^T}{\sqrt{d_k}}\right)V \tag{3}$$

The multi-head attention mechanism is an extension of self-attention, where multiple attention heads are computed in parallel, and their outputs are concatenated. Given h attention heads, head_i is generated through $Attention(q_i, k_i, v_i), W^O$ is the output projection matrix, the multi-head attention mechanism can be expressed as:

$$MultiHead(Q, K, V) = Contact(head_1, \cdots, head_h)W^O \tag{4}$$

Apart from the attention mechanism, the Transformer architecture also includes a fully connected feedforward network, which applies a nonlinear transformation to each position. Given an input X, the feedforward layer with weights W_i and bias b_i, can be expressed as:

$$FFN(X) = ReLU(XW_1 + b_1)W_2 + b_2 \tag{5}$$

These are the fundamental components of the Transformer model. In practical applications, additional techniques such as residual connections, layer normalization, and dropout may also be applied to enhance performance.

2.2. Knowledge Distillation

Knowledge Distillation (KD) is a model compression technique designed to improve the performance of a smaller student model by training it on the predictions of a larger teacher model. Based on whether the teacher and student models are updated simultaneously, knowledge distillation can be categorized into three types:

Offline Distillation—The teacher model is pre-trained before knowledge distillation. The student model is then trained with additional guidance from the teacher's logits or intermediate features. This method focuses on designing effective knowledge representations rather than structural relationships between the models. However, the capacity gap between the teacher and student models persists, and performance heavily depends on the teacher model's quality.

Online Distillation—The teacher and student models are trained simultaneously. This approach is suitable when no well-trained teacher model is available, allowing for an end-to-end trainable distillation framework. However, existing online distillation methods still struggle with high-capacity teachers.

Self-Distillation—The model uses its own outputs as auxiliary references to guide its own learning iterations, thereby improving generalization performance. A variation of this is snapshot distillation, where early iterations of a model serve as a teacher to later iterations, enabling supervised learning within the same network architecture.

In knowledge distillation, the Softmax function is commonly used to represent model prediction probabilities. Temperature scaling adjusts the Softmax function's temperature parameter to control the smoothness of the predicted distribution:

$$P_i = \frac{exp(z_i/T)}{\sum_j exp(z_i/T)} \tag{6}$$

For the computed predicted probabilities, cross-entropy loss is used to measure the difference between the model's predictions and the target distribution. The cross-entropy loss formula for the prediction distribution q of the teacher model and the prediction distribution p of the student model is as follows:

$$Loss = -\sum_i q_i log(p_i) \tag{7}$$

The prediction results of the teacher model are referred to as soft targets, which are typically used to guide the training of the student model. Soft targets are based on the Softmax distribution of the teacher model and serve as a replacement for the original one-hot labels. The loss function of the student model usually incorporates both hard targets and soft targets. Here, λ represents the weight balancing hard and soft targets, CE denotes the cross-entropy loss, p_{hard} is the student model's hard target prediction, y_{hard} is the one-hot label, p_{soft} is the student model's soft target prediction, $p_{teacher}$ represents the teacher model's predictions. The specific formulation is as follows:

$$Loss = \lambda \cdot CE(p_{hard}, y_{hard}) + (1-\lambda) \cdot CE\left(p_{soft}, p_{teacher}\right) \tag{8}$$

These formulas form the core framework of knowledge distillation. While variations and extensions exist, they provide a fundamental understanding of how knowledge distillation enhances model compression and efficiency.

2.3. ORIC graph chain architecture

The ORIC consensus [5] protocol has the following characteristics: Decentralization and Permissionlessnessz: All nodes have the opportunity to add blocks to the chain at any given time. Many existing blockchain protocols, in pursuit of scalability, adopt hybrid consensus mechanisms. Some select a single node to lead block production, while others elect a committee at intervals to reach consensus. Having only a few block producers can lead to centralization and make the system more vulnerable to DoS attacks. The ORIC protocol allows consensus nodes to generate blocks at any time rather than at fixed intervals, maximizing decentralization. Additionally, nodes can join or leave the network without requiring identity verification. High Bandwidth Utilization: In a P2P network environment, the system's transaction throughput cannot exceed the average available bandwidth. Within the limits of available bandwidth, the protocol strives to maximize bandwidth utilization, approaching 50%; Theoretical Provability: The protocol is sufficiently simple to be formally proven for the security and liveness of the generated public ledger. Furthermore, the system's tolerance threshold for malicious nodes is close to 1/2; Bandwidth Adaptability: The protocol can automatically achieve the optimal current throughput without forking. Additionally, it can adjust throughput to some extent based on variations in node bandwidth availability with minimal impact on security.

Each valid block B_i on the chain can be represented by a five-tuple:

$$B_i = (h_{-1}, \eta, v, m, h) \tag{9}$$

The block identifier signifies that the block was produced by node B and i indicates its height in the blockchain. The corresponding elements of the five-tuple can be extracted using the symbol $\langle\cdot\rangle$. $B_i.h_{-1}$ represents the hash of the previous block. $B_i.\eta$ denotes a random value used for Proof-of-Work (PoW) computation, where each iteration involves modifying η. $B_i.v$ is a list of hashes connecting multiple other blocks. $B_i.m$contains additional payload information, such as protocol configuration parameters and transaction data. In the PoW process, miners must continuously attempt different values of η to ensure that the

block hash is below a certain threshold, referred to as the difficulty target D_p.Within the graph chain protocol, blocks are classified into primary blocks and secondary blocks based on the extent to which they meet the difficulty target:

A block $B_i$is a primary block if and only if the following conditions hold: $H(B_i.h_{-1}, B_i.\eta, B_i.v, B_i.m) < D_p, if\ i \geq 1, i \in \mathbb{N}, B_{i-1}.h = B_i.h_{-1}$, where B_{i-1} is also a primary block if $i = 0$, $B_i.h_{-1} = (0)^k$.

A block B_i is a secondary block if and only if the following conditions hold: $H(B_i.h_{-1}, B_i.\eta, B_i.v, B_i.m) < kD_p, where\ k \in \left[1, \frac{2^k}{D_p}\right] \subset \mathbb{R}$,if $i \geq 1, i \in \mathbb{N}, B_{i-1}.h = B_i.h_{-1}, where\ B_{i-1}$ is also a primary block if $i = 0$, $B_i.h_{-1} = (0)^k$.

According to the definition, if a block meets the primary block criteria, it automatically qualifies as a secondary block as well. Unless otherwise specified, "secondary block" typically refers only to those that are not primary blocks. In the definition, the parameter k is also known as the throughput parameter. By adjusting the value of k, the difficulty of block production can be modified, indirectly influencing the block generation rate and consequently the system's throughput.

2.3.1. Block generation mechanism

In summary, the block generation mechanism in the Graph Chain protocol allows a node to produce either a primary block or a secondary block when solving the Proof-of-Work (PoW) challenge. The classification is probabilistic. Meanwhile, through the incentive mechanism, blocks form linkage relationships and automatically organize into a graph structure. To better explain the formation of the graph structure and the block generation mechanism, consider a graph G_i as an example. The timeline progresses from top to bottom in sequential rounds, with time divided into slots. Blocks are generated within each slot and are received by all nodes in the following slot. Some blocks may experience network latency and are received in later slots. Slots serve as an auxiliary unit for segmenting time (rounds), and each node's view within a slot corresponds to a particular round's view.

2.3.2. Ordering mechanism

A blockchain consensus protocol must ensure that all nodes derive a unique block sequence from the same block view. Traditional "longest chain mechanisms" naturally produce a linear chain structure, eliminating the need for explicit ordering. The Graph Chain protocol, being an extension of the longest chain mechanism, naturally forms a graph structure sequence. Consequently, once this graph structure sequence is established, it is only necessary to sort the blocks within it to derive a final block sequence. If blocks are generated in different slots, their order is determined by their direct or indirect references via hash links, reflecting their absolute chronological order. Blocks generated earlier have higher priority. However, due to network latency and concurrent block production, multiple blocks may be created within the same slot without direct hash connections, making it impossible to infer their exact chronological order. For such cases, a predefined rule can be used for logical ordering, such as sorting based on hash difficulty.

2.3.3. Incentive mechanism

The incentive mechanism is the core principle driving rational nodes in public blockchain protocols to maximize their profits and is fundamental to ensuring protocol stability. This involves economic principles and game theory, which are beyond the scope of this discussion. However, a simple principle— "more work, more rewards"—is followed for designing a basic incentive structure. Miners, as block producers, expend significant computational power solving PoW challenges. The first transaction in each block, along with transaction fees, serves as a reward for successfully mining a block. A rational miner typically prioritizes transactions with higher fees for inclusion. In the Graph Chain protocol, it is assumed that receiving secondary blocks from other nodes during mining is merely an additional attempt. However, in real-world conditions, rational nodes are more inclined to dedicate computational power to mining rather than expending resources on receiving secondary blocks, as rejecting them does not impact their own profits. Therefore, incentives must be provided for processing secondary blocks. The reward should be proportional to the number of confirmed secondary blocks and must exceed the potential profit from using the same computational power for regular mining. This incentive mechanism ensures that rational nodes follow the protocol's predefined rules, thereby maintaining its efficiency and security.

2.3.4. Bandwidth adaptation mechanism

The bandwidth adaptation mechanism enables the protocol to automatically adjust its throughput in response to bandwidth variations while maintaining security. Security, in simple terms, refers to the probability that the protocol does not experience a fork—the higher this probability, the greater the security. Security is mainly related to the ratio of Block Propagation Delay (BPD) to the primary block interval time (Block-time):

$$\frac{BPD}{block-time} \tag{10}$$

A smaller BPD allows blocks to synchronize faster across the P2P network, reducing the probability of forks and increasing security. A larger Block-time provides more time for block propagation, lowering the probability of two blocks being generated within the same BPD period, thereby reducing fork probability and enhancing security.

Before introducing the bandwidth adaptation mechanism, it is necessary to define the bandwidth adjustment parameter and its calculation method:

$$\tau(l,w) = \frac{\sum_{i=l}^{l+w-1} |G_i B|}{w \cdot k} \tag{11}$$

$\tau(l,w)$ represents the ratio of two values. The numerator is the total number of blocks within the graph structure from height l to $l+w-1$, while the denominator $w \cdot k$ represents the theoretical number of blocks within this height range. If a block is discarded due to network delay or because a secondary block is generated within the same slot as the primary block but remains unconfirmed by the primary block, then as BPD increases or Block-time decreases, the probability of block discards rises. τ follows the relationship:

$$\tau \sim 1 - \frac{BPD}{block-time} \tag{12}$$

3. Solution process

3.1. Overall framework structure

The framework proposed in this paper is shown in the figure below. The frontend communicates with the backend using HTTP, WebSocket, and Ajax technologies. The backend includes a log collection module and an anomaly detection module. Once an anomaly is detected in the logs, the system identifies the corresponding log details using a system identifier. Then, the primary information is uploaded to the blockchain through a smart contract to ensure tamper-proofing and traceability. The log data is also persistently stored via IPFS. If a system anomaly needs to be traced later, the hash value of the anomalous log can be searched on the blockchain. The overall system framework is shown in Figure 1.

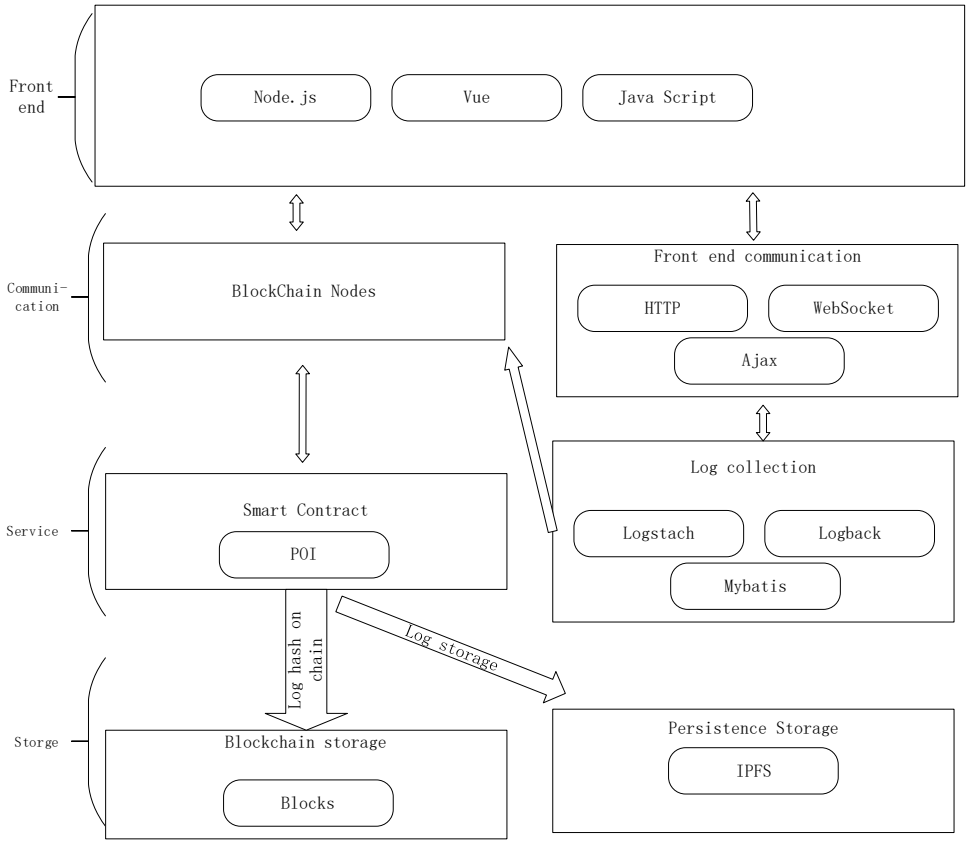

Figure 1. System framework diagram

3.2. Smart contract on blockchain

For the anomalous logs identified by the model, preprocessing is required, including data cleaning and feature extraction. The purpose of this step is to remove duplicate log entries and standardize the field formats (e.g., unifying timestamps to UTC time, converting IP addresses to CIDR notation). The SHA-256 hash of the raw log data is then calculated. A notarization contract is written using Solidity, and the data structure of the smart contract is shown in Table 1. The submitLog function is used to validate the log hash value. If the validation is successful, the correctly formatted log is pushed onto the blockchain.

Table 1. Log uplink data structure table

Data Format	Data Attribute	Descriptions
Uint256	Time Stamp	Uplink Time
String	Log Hash	Log Hash
Address	Submitter	Submitter
Uint8	Anomaly Status	Abnormal State
String	Encrypted Data	Encrypted Data

The smart contract is deployed on the test chain, and the contract address is obtained. Then, a transaction data packet is created, and the submitter's private key is used to sign the data packet using ECDSA. After calling the smart contract, the log is converted into a transaction and enters the memory pool, waiting for miners to package it.

3.3. Data traceability

By verifying the Merkle Root in the block header, the existence of the log can be confirmed. After searching for the log, the hash value is compared to decrypt and verify the log's storage location. The log's complete information is then parsed from the distributed database.

4. Experimental validation

4.1. Dataset

The dataset used for experimental validation in this framework is the BGL (BlueGene/L) dataset. This dataset is one of the benchmark datasets for supercomputing system logs and is widely used in research related to log analysis, anomaly detection, and fault prediction. It was generated by the IBM BlueGene/L (BGL) supercomputer, which is used in high-performance computing (HPC) fields. The dataset records hardware/software events during system operation, including normal operations, warnings, errors, and critical failures. The raw log size is approximately 4.7 GB, with a total of 4,747,963 log entries, of which 348,460 are anomaly logs, accounting for about 7.3% of the total. An example of an anomalous log is: "APPREAD 1117869876 2005.06.04 R27-M1-N4-I: J18-U01 2005-06-04-00.24.36.222560 R27-M1-N4-I: J18-U01 RAS APP FATAL ciod: failed to read message prefix on control stream (CioStream socket to 172.16.96.116:33370)." "APPREAD" refers to the type of event or source module recorded in the log. "1117869876 2005-06-04-00.24.36.222560" refers to the Unix timestamp and the system's detailed timestamp. "R27-M1-N4-I: J18-U01" represents the component identifier, pinpointing the hardware location where the failure occurred for physical troubleshooting or log correlation analysis. "RAS APP" refers to the system category, in this case, the Reliability, Availability, and Serviceability (RAS) module of the IBM system, typically responsible for error handling and log recording. APP indicates that the event is related to the application. "FATAL" represents the error severity level, indicating a critical error that may lead to system crash or disruption of key functions, requiring immediate attention. "ciod: failed to read message prefix on control stream (CioStream socket to 172.16.96.116:33370)" displays the error details, indicating the error type, network connection protocol, and target address.

4.2. Experiment process

First, we parse the raw BGL.log using regular expressions `r'^(?P<tag>-|\w+)\s+'` and `r'(?P<message>.*)$'`, which separates the log into normal and anomalous logs. Next, the data is converted into a DataFrame using the open-source Pandas library. The already initialized DistilBERT model's tokenizer is then used to tokenize the dataset, and the tokens are input to the model for fine-tuning. The fine-tuned model is then used to perform the anomaly log detection task. The model configuration parameters were as follows:

```
num_train_epochs=5;
per_device_train_batch_size=64;
per_device_eval_batch_size=128;
max_grad_norm=1.0;
learning_rate=2e-5;
warmup_steps=500;
weight_decay=0.01
```

Under these parameters, we achieved an accuracy of 99.68%, an F1 score of 97.9%, a precision of 95.8%, and a recall rate of 100%. The anomaly detection model's false positive rate and false negative rate were 0.34% and 0%, respectively. Below is a chart showing our experimental results:

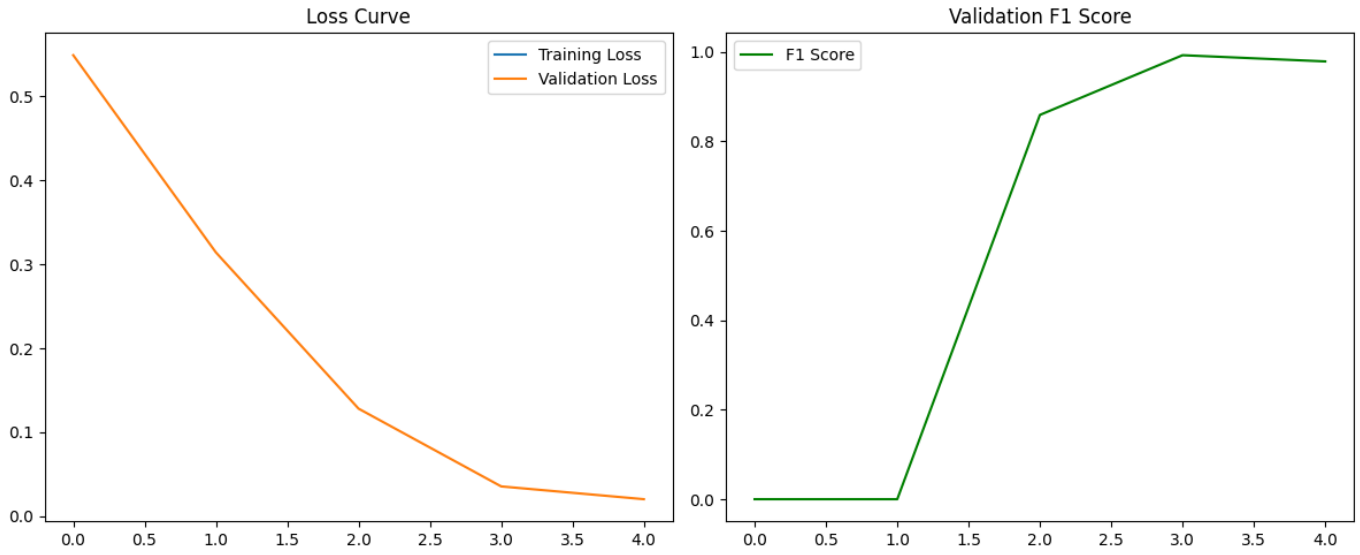

Figure 2. System training Loss vs. F1 score variation graphs

As shown in Figure 2, the Transformer architecture model can effectively fit the log sequence anomaly detection task, with the loss value dropping to a very small level by the fourth round.

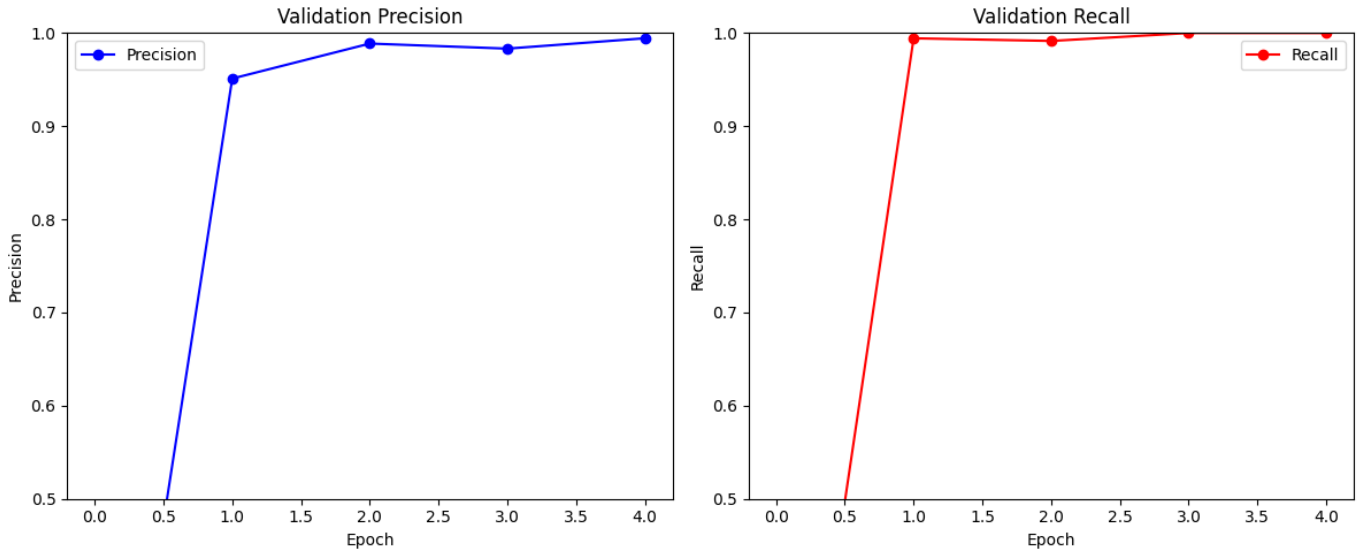

Figure 3. System accuracy vs. recall variation graphs

Figure 3 shows the trends of accuracy and recall rate throughout the training process.

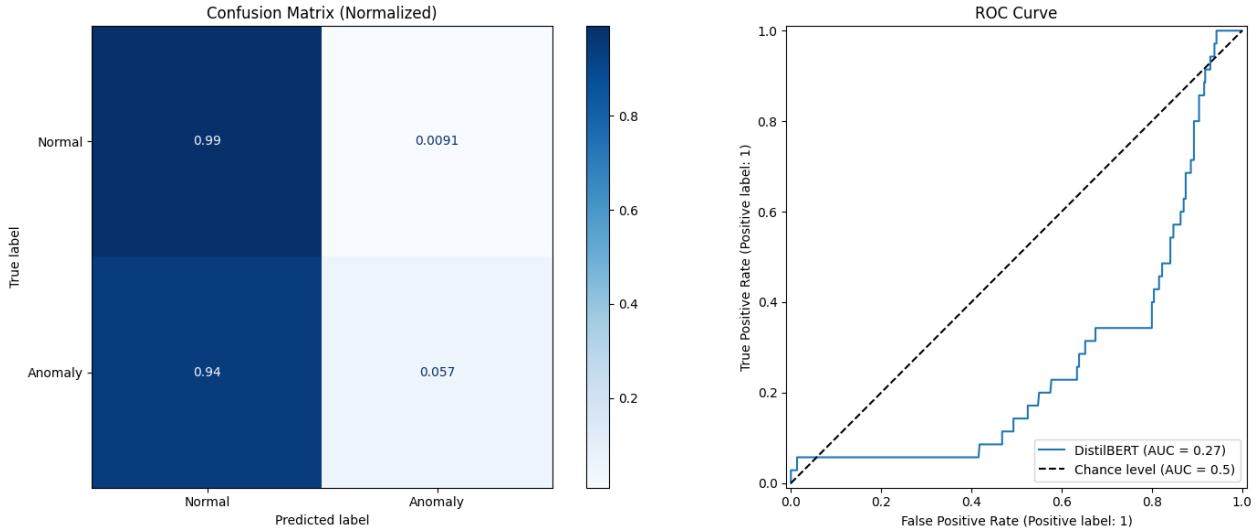

Figure 4. Schematic diagram of confusion matrix and ROC curve at the beginning of the training period

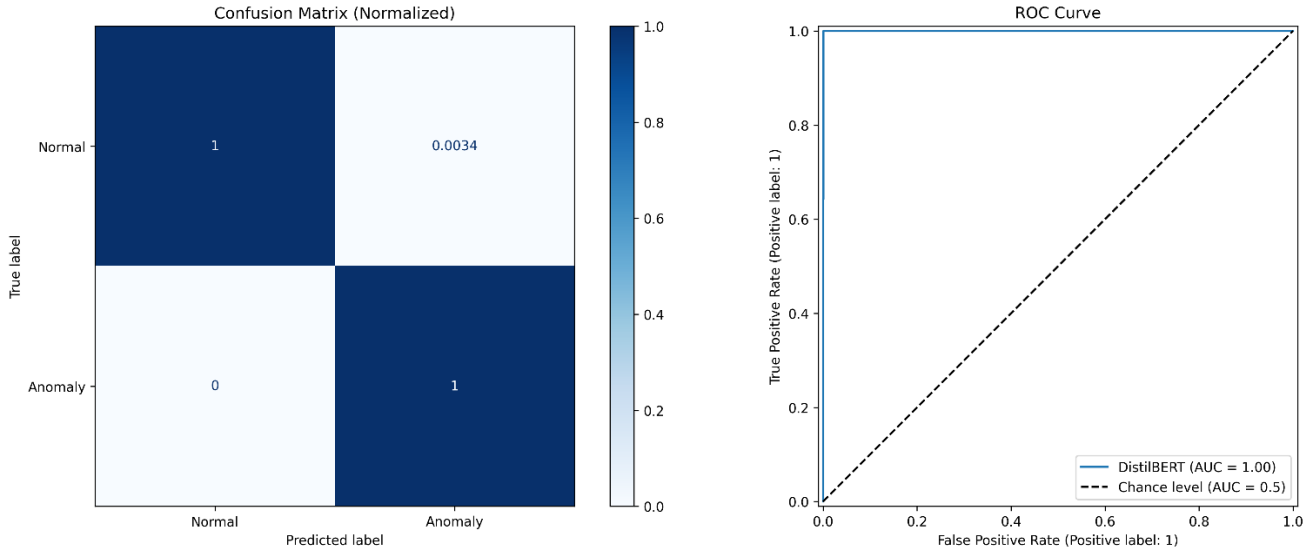

Figure 5. Schematic diagram of confusion matrix and ROC curve at the late stage of training

Figure 4 and Figure 5 display the confusion matrix and ROC curve at the beginning of the first round of training and after the training is completed. The confusion matrix is a matrix used to describe the performance of a classification model, particularly suitable for binary classification (and can be extended to multi-class). It compares the predicted results with the true labels. The ROC curve is a graphical tool used to evaluate the performance of a classification model by plotting the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR), reflecting the model's classification capability. At the start of training, the model performs poorly in recognizing anomalous logs, with a 100% probability of classifying anomalous logs as normal. After training, this probability drops to around 0%, although there are still very few instances where normal logs are misclassified as anomalous logs (0.34%).

5. Conclusion and future outlook

The log anomaly detection framework based on the graph-chain architecture proposed in this paper can effectively handle log anomaly detection tasks in distributed systems. With the high throughput provided by the graph-chain architecture, it can effectively cope with the sudden high data flow caused by large-scale service crashes. At the same time, the underlying blockchain

technology offers strong traceability and tamper resistance, making it suitable for applications in distributed systems that require high security.

In the future, consideration can be given to using more efficient, smaller parameter-based sequence classification models to handle log analysis tasks. This would address the issue of insufficient hardware performance in lightweight system nodes.

References

[1] Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (pp. 117–132). Association for Computing Machinery. https://doi.org/10.1145/1629575.1629587

[2] Li, K.L., Huang, H.K., Tian, S.F., & Xu, W. (2003). Improving one-class SVM for anomaly detection. *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics* (Vol. 5, pp. 3077-3081). IEEE. https://doi.org/10.1109/ICMLC.2003.1260106

[3] Wang, Y., Wong, J., & Miner, A. (2004). Anomaly intrusion detection using one class SVM. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop* (pp. 358–364). IEEE. https://doi.org/10.1109/IAW.2004.1437839

[4] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter*. arXiv. https://doi.org/10.48550/arXiv.1910.01108

[5] Xiong, T., Xie, T., Xie, J., & Luo, X. (2021). ORIC: A self-adjusting blockchain protocol with high throughput. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)* (pp. 1422–1434). IEEE. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00193

[6] Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1285-1298). Association for Computing Machinery.

[7] Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., Chen, J., He, X., Yao, R., Lou, J.-G., Chintalapati, M., Shen, F., Zhang, D. (2019). Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)* (pp. 807–817). Association for Computing Machinery. https://doi.org/10.1145/3338906.3338931

[8] Le, V. H., & Zhang, H. (2021). Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 492–504). IEEE. https://doi.org/10.1109/ASE51524.2021.9678773