Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 18, No. 2, 2026

Research on Blockchain Security Detection Based on Machine Learning

Yukang Fang *

School of Financial Technology, Shanghai Lixin University of Accounting and Finance, Shanghai, 201209, China
*** Corresponding author Email:** 3555484564@qq.com

Abstract: With the rapid development of blockchain technology, its decentralized, tamper-resistant and traceable characteristics have been widely adopted in fields such as finance, supply chain management and the Internet of Things. However, the openness of blockchain ecosystems also creates opportunities for criminals, posing serious threats to the stability of blockchain networks and the security of user assets. To address the difficulties in identifying on-chain malicious transactions, the low audit efficiency of smart contract vulnerabilities, and the complexity and poor reproducibility of existing models in traditional blockchain scenarios, this paper proposes a lightweight machine learning-based detection scheme. The Elliptic transaction dataset and a smart contract vulnerability dataset constructed from the Slither official test suite and SWC (Smart Contract Weakness Classification) standard vulnerability cases are adopted, and random forest, XGBoost, LSTM and CNN are selected to build detection models. Multi-model comparative experiments are conducted with accuracy, precision, recall, F1-score and AUC as evaluation metrics. Experimental results show that the proposed scheme can effectively detect malicious transactions and contract vulnerabilities, providing a reference for intelligent blockchain security protection.

Keywords: Blockchain Security; Malicious Transaction Detection; Smart Contract Vulnerability; Machine Learning; Lightweight Detection.

1. Introduction

With its decentralized, tamper-resistant, transparent and traceable characteristics, blockchain technology has been widely applied and rapidly developed in digital finance, supply chain management, the Internet of Things and other fields. However, the openness and anonymity of blockchain networks also create opportunities for security threats such as malicious transactions and smart contract vulnerabilities, seriously threatening network stability and user asset security [1]. Traditional detection methods such as manual audit, rule matching and static scanning suffer from limitations including low efficiency, high false positive rate and poor adaptability to unknown attacks; consequently, machine learning-based intelligent security detection has become an important research direction in the field of blockchain security.

Existing studies have achieved rich results in abnormal transaction identification and smart contract vulnerability detection [2-3]. However, most methods focus on a single detection task, suffer from high model complexity and weak experimental reproducibility, and show insufficient robustness in scenarios with class imbalance and scarce malicious samples, making it difficult to meet the practical requirements of lightweight, general and easily deployable detection.

To address the above problems, taking lightweight and reproducible detection as the core objective, this paper adopts four models, namely random forest, XGBoost, LSTM and CNN, and conducts comparative experiments on the Elliptic Bitcoin transaction dataset and a smart contract vulnerability dataset constructed from the Slither official test suite and SWC standard vulnerability cases. Through data preprocessing, feature engineering, model training and performance evaluation, an integrated lightweight detection scheme covering on-chain malicious transaction identification and smart contract vulnerability detection is constructed, and the effectiveness and practicability of the scheme are verified by multi-model comparative experiments.

2. Blockchain Security Related Concepts

2.1. Definition and Typical Types of On-chain Malicious Transactions

On-chain malicious transactions refer to illegal fund transfer activities carried out through anonymous addresses on the blockchain. In the early Bitcoin network, dark web money laundering and double-spending attacks were the main forms; after the emergence of the Ethereum DeFi ecosystem, various attack forms such as coin-mixing laundering, MEV front-running arbitrage and gang fraud transfers have appeared. Attackers generally employ methods such as splitting transfers into small amounts, multi-address layered relay and cross-chain transfer to evade basic rule-based screening. Since malicious samples account for an extremely low proportion of the dataset, detection is difficult. Typical malicious transactions in practice include three categories: first, the Kyberswap theft incident, in which hackers split tens of millions of dollars of stolen assets into multiple transfers into the Tornado Cash mixer and completed money laundering through multi-layer anonymous wallets and cross-chain transfers; second, MEV sandwich attacks, in which attackers seize the block packaging order and insert arbitrage orders before and after ordinary users' exchange transactions, encroaching on user returns through continuous sequential transactions; third, cryptocurrency pig-butchering fraud, in which fraud groups collect victims' assets and disperse them to a large number of blank addresses for layer-by-layer transfer without normal DApp interaction, forming obvious address cluster anomalies.

2.2. Common Smart Contract Vulnerabilities and Their Hazards

Since the code of a deployed smart contract cannot be modified, logic or syntax vulnerabilities directly lead to asset theft. There are four types of high-frequency vulnerabilities in the industry: the first is the reentrancy vulnerability, in which the contract fails to update the balance after transfer and the attacker recursively calls the withdrawal function to steal funds, as exploited in The DAO attack; the second is integer overflow/underflow, in which numerical operations exceeding the storage limit lead to incorrect asset values; the third is the access control vulnerability, in which the contract fails to restrict the identity of callers, allowing external accounts to arbitrarily execute high-risk transfer functions; the fourth is the transaction order dependence vulnerability, in which attackers seize the block execution order to tamper with contract returns. These vulnerabilities, individually or in combination, can cause large-scale asset loss; major security incidents such as Poly Network and Ronin Bridge are directly related to contract vulnerabilities.

2.3. Classification of Blockchain Data Features

The experimental data in this paper are divided into two feature systems. The first is account transaction statistical features, including structured numerical features such as the average interval of sending/receiving transactions, cumulative transfer amount, number of contract interactions and types of interacted tokens, which are suitable for traditional machine learning models such as random forest and XGBoost. The second is sequence features, including smart contract Opcode sequences and time-ordered continuous transaction records, which are one-dimensional temporal text features suitable for deep learning models such as CNN and LSTM. The two types of features jointly serve as model inputs to realize integrated transaction and contract detection.

3. Related Experimental Methods

3.1. Model Evaluation Metrics

For the binary classification tasks of on-chain malicious transaction detection and smart contract vulnerability detection, this paper adopts four core metrics, namely accuracy, precision, recall and F1-score, supplemented by AUC to comprehensively evaluate the ability of the model to distinguish positive and negative samples. All metrics are calculated based on the confusion matrix, which contains four basic sample categories: TP (true positive) is the number of samples that are actually malicious and predicted as malicious; FN (false negative) is the number of actually malicious samples misjudged as normal; FP (false positive) is the number of normal samples misjudged as malicious; and TN (true negative) is the number of actually normal samples predicted correctly.

Table 1. Confusion matrix.

	Predicted 0	**Predicted 1**
Actual 0	TN	FP
Actual 1	FN	TP

Accuracy reflects the proportion of correctly predicted samples, intuitively representing the global recognition ability of the model; precision focuses on the samples predicted as malicious and represents the proportion of real attack samples among them, measuring the false positive control ability of the model; recall focuses on the capture of all real malicious samples, representing the proportion of all attack samples successfully identified, reflecting the missed detection control level; F1-score is the harmonic mean of precision and recall, balancing the two metrics to avoid evaluation bias caused by a single high metric. AUC is the area under the ROC curve, ranging in (0, 1); the closer the value is to 1, the stronger the model's ability to distinguish imbalanced data, which is suitable for detection scenarios where blockchain malicious samples are scarce and the dataset is imbalanced.

3.2. Machine Learning Model Principles

3.2.1. Random Forest

Random forest is an ensemble learning model composed of a large number of independent decision trees. It randomly selects training samples by bootstrap sampling and randomly screens features to train each decision tree, and finally outputs classification results by majority voting. A single decision tree divides samples through feature splitting and can automatically mine nonlinear correlation features in transaction and contract data. The random sampling mechanism reduces the overfitting risk of a single tree, and the model is robust to imbalanced blockchain transaction data and discrete contract features. It can achieve stable detection without complex parameter tuning, and its lightweight characteristics fit ordinary hardware environments, making it the benchmark lightweight model compared in this paper [4].

3.2.2. XGBoost

XGBoost is an improved gradient boosting tree algorithm that iteratively generates multiple decision trees in a serial training manner. Each tree fits the prediction residual of the previous tree and minimizes the classification loss through gradient descent. The algorithm incorporates regularization terms to constrain model complexity and effectively alleviate overfitting, and supports automatic missing value handling and adaptation to multiple feature types. It achieves excellent fitting performance for structured features such as Ethereum account statistics and contract numerical features, and its training speed is superior to traditional gradient boosting models, showing good accuracy in fraud account and vulnerability sample identification tasks [5].

3.2.3. CNN (Convolutional Neural Network)

A one-dimensional convolutional neural network is suitable for sequence data. It extracts local features through sliding convolution kernels, reduces the number of model parameters through weight sharing, and completes feature dimensionality reduction with a max-pooling layer. For smart contract Opcode sequences and continuous time-series transaction records, CNN can automatically extract key abnormal features within segments, such as continuous dangerous instructions in contracts and high-frequency short-time transfers, without the need to manually design complex features, making it suitable for processing code sequences and time-series transaction data [6].

3.2.4. LSTM (long Short-term Memory)

LSTM is an improved recurrent neural network that solves the gradient vanishing problem of traditional RNN through three types of gating units, namely the forget gate, input gate and output gate, and has the ability to capture long-range temporal dependencies. Blockchain account transactions

have continuous temporal characteristics, and there are long-term correlations among account historical transfer intervals and fund flow sequences. LSTM can memorize long-distance transaction behavior changes and accurately identify hidden temporal attacks such as layered money laundering and staged malicious arbitrage; it can also be combined with CNN to take into account both local code features and long-term transaction behavior patterns [7].

4. Experimental Design

4.1. Overall Detection Framework

Aiming at two core tasks in the field of blockchain security, namely Bitcoin malicious transaction identification and smart contract vulnerability detection, this paper constructs a unified detection framework of "data input - preprocessing - model training - performance evaluation", as shown in Figure 1. The framework is divided into four layers from bottom to top: the data layer, preprocessing layer, model layer and evaluation layer.

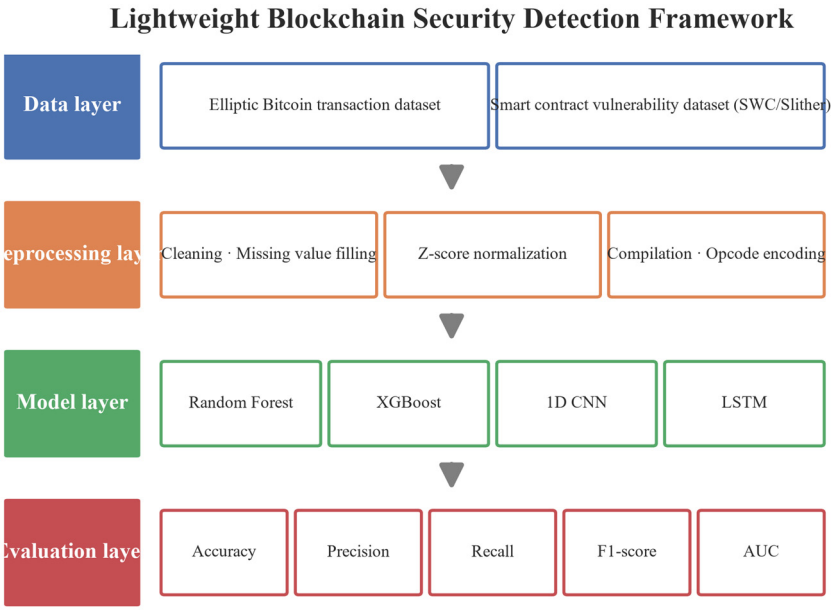


Figure 1. Overall detection framework

Data layer: the Elliptic Bitcoin transaction dataset and the smart contract vulnerability dataset are connected respectively, and the collection and label verification of raw data are completed.

Preprocessing layer: cleaning, missing value filling and normalization are performed for numerical transaction features; for smart contract source code, compilation, bytecode parsing and Opcode sequence encoding are performed to unify the input format.

Model layer: traditional ensemble learning models (random forest, XGBoost) and deep learning sequence models (one-dimensional CNN, LSTM) are deployed in parallel to adapt to the detection requirements of structured data and sequence data respectively.

Evaluation layer: accuracy, precision, recall, F1-score and AUC are adopted as core evaluation metrics, and multi-model horizontal comparison is completed from three dimensions: classification accuracy, generalization ability and detection efficiency.

The framework supports independent training and inference of the two tasks, and the detection target and model combination can be flexibly switched according to business scenarios.

In this paper, "lightweight" is defined operationally from the perspective of deployment feasibility: all models, including the deep learning models, are trained and inferred on an ordinary office-grade hardware platform (Intel Core i7 processor, 16 GB memory, no dedicated GPU, as described in Section 4.5.1), and the model structures are concise enough to be deployed on conventional devices without specialized inference hardware. As quantified in Tables 2 and 3, except for LSTM on the transaction detection task (about 19 minutes), the training time of all models is within 70 seconds, which demonstrates the low resource threshold of the proposed scheme and its reproducibility on ordinary equipment.

4.2. Dataset Selection

4.2.1. Elliptic Dataset

The Elliptic publicly available Bitcoin transaction graph dataset is adopted in this experiment; it is one of the most widely used benchmark datasets in the field of blockchain malicious transaction detection [8].

Data composition: it contains 203,769 Bitcoin transaction nodes, each corresponding to 166-dimensional features, covering transaction statistical features (number of transactions, transfer amount, fees, time intervals, etc.) and graph network features (in/out degree, neighbor node features, associated address statistics, etc.).

Label system: samples are divided into three categories: licit transactions (label 0, about 80%), illicit transactions (label 1, about 10%) and unlabeled samples; only samples with explicit labels are selected for training and testing. The proportion of each label category is shown in Figure 2.

Scenario value: the dataset naturally has class imbalance characteristics, highly matching the real scenario in which malicious samples account for an extremely low proportion of blockchain transactions, and the detection results have strong practical reference significance.

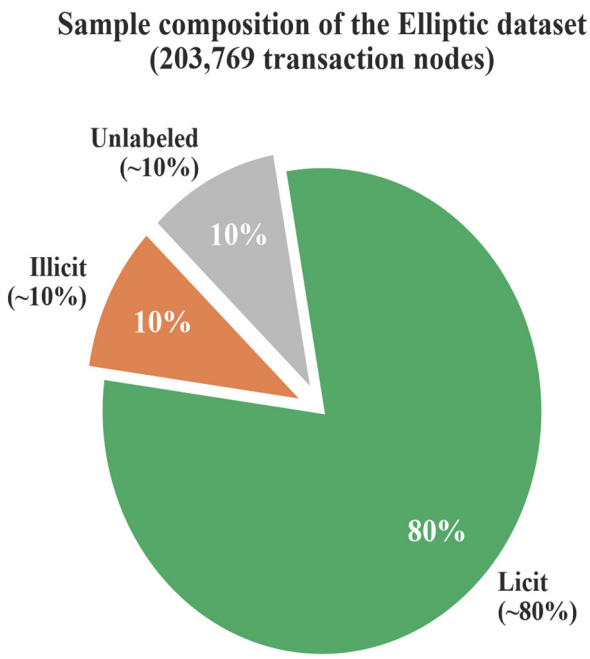


Figure 2. Sample composition of the Elliptic dataset

4.2.2. Smart Contract Vulnerability Dataset

The smart contract vulnerability detection dataset is constructed based on the Slither official test suite and SWC (Smart Contract Weakness Classification) standard vulnerability cases, covering typical vulnerability types such as reentrancy attacks, integer overflow, access control defects, unchecked transfer return values and uninitialized variables [9-10].

Positive samples (label 1): Solidity contract source code with explicit security vulnerabilities is selected, corresponding to the standard test cases of various vulnerabilities.

Negative samples (label 0): same-source safe contracts after vulnerability repair and benign business contracts without vulnerabilities are selected, ensuring that positive and negative samples differ only in vulnerability logic and avoiding detection distortion caused by data distribution differences.

Data processing: Solidity contracts are compiled into runtime bytecode, further mapped into Opcode sequences, and finally a space-separated sequence text dataset is formed to adapt to the input requirements of sequence models. After cleaning and deduplication, the final dataset contains 39 vulnerable contracts (label 1) and 43 benign contracts (label 0).

4.3. Data Preprocessing

4.3.1. Data Cleaning and Missing Value Handling

Cleaning operations are performed for the two datasets respectively. For the Elliptic dataset, unlabeled samples are removed, hidden spaces and invalid characters in column names are removed, median filling is adopted for a small number of missing statistical features to avoid the influence of extreme values, and label value ranges are verified to remove abnormal label samples. For the smart contract dataset, invalid samples with compilation failure or too-short bytecode are filtered, Opcode sequences are deduplicated to avoid overfitting caused by completely duplicated contract samples, opcode naming specifications are unified, and invalid metadata characters are removed.

4.3.2. Feature Normalization and Opcode Feature Extraction

Numeric feature normalization: Z-score standardization is applied to the numerical features of the Elliptic dataset, mapping the features to a distribution with mean 0 and standard deviation 1, eliminating the influence of dimension differences on algorithms other than tree models.

Opcode feature extraction and encoding: Solidity contracts are compiled to obtain runtime bytecode, and the hexadecimal bytecode is converted into semantic Opcode sequences through the opcode mapping table; an Opcode vocabulary is constructed and the sequences are integer-encoded; fixed-length truncation and zero padding are used to unify the input dimensions of all samples; the Embedding layer maps discrete opcodes into low-dimensional dense vectors as the input of deep learning models. The complete extraction and encoding process of Opcode sequence features is shown in Figure 3.

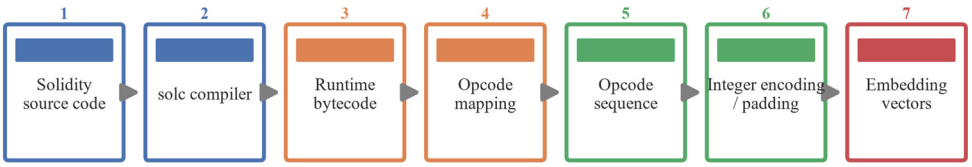


Figure 3. Opcode sequence feature extraction and encoding pipeline

4.3.3. Training Set and Test Set Partitioning

Both datasets adopt a stratified random partitioning strategy, dividing the training set and test set at a ratio of 7:3, ensuring that the proportion of positive and negative samples in the training set and test set is consistent with the original dataset and avoiding result distortion caused by partition bias. The training set is used for model fitting and parameter optimization, and the test set is used only for final performance evaluation and does not participate in the training process.

4.4. Model Selection and Construction Rationale

Four representative models are selected in this experiment, covering both traditional machine learning and deep learning, to realize horizontal comparison of different paradigms. Random forest (RF) is an ensemble learning model with multiple decision trees as base classifiers, which reduces the overfitting risk through the bagging mechanism, has strong adaptability to high-dimensional structured data, is stable in training, and serves as the baseline model of traditional machine learning. XGBoost is an optimized implementation of gradient boosting trees, which greatly improves training efficiency while ensuring accuracy through regularization, parallel computing and automatic missing value handling, performs well in class-imbalanced scenarios, and serves as the comparison benchmark of ensemble learning. One-dimensional CNN extracts local features of sequences

through sliding convolution kernels, is good at capturing local vulnerability patterns in Opcode sequences, has high computational efficiency and moderate parameter scale, and is suitable for sequence feature extraction with small samples. LSTM captures long-range temporal dependencies of sequences through gating mechanisms and is used to verify the gain of global temporal information of Opcode sequences for vulnerability detection, serving as the representative of deep learning sequence models.

4.5. Experimental Environment and Parameter Settings

4.5.1. Experimental Environment

Hardware environment: Intel Core i7 processor, 16 GB memory, Windows 11 operating system. Software environment: Python 3.10; the machine learning frameworks adopt Scikit-learn and XGBoost, the deep learning framework adopts TensorFlow 2.x, and the contract compilation adopts the py-solc-x toolchain.

4.5.2. Parameter Settings

Grid search combined with five-fold cross-validation is adopted to select the optimal parameters. The core parameter ranges of each model are as follows: random forest: the number of decision trees takes values in {50, 100, 200} and the maximum depth in {None, 8, 16}; XGBoost: the learning rate in {0.05, 0.1}, the maximum tree depth in {2, 3, 6} and the number of iterations in {50, 100, 200}; one-dimensional CNN: the convolution kernel size in {3, 5} and the number of filters in {64, 128}; LSTM: the number of hidden units in {64, 128}, the training epochs set to 50, and early stopping adopted to prevent overfitting.

5. Experimental Results and Analysis

5.1. Malicious Transaction Detection Results and Analysis

Following the evaluation metrics described in Section 3.1, with recall, F1-score and AUC as the core metrics and accuracy as an auxiliary reference, the malicious transaction detection results based on the Elliptic dataset are shown in Table 2:

Table 2. Malicious transaction detection results.

Model	Accuracy	Precision	Recall	F1-score	AUC	Training time /s
Random Forest	0.988	0.975	0.806	0.882	0.994	69.03
XGBoost	0.988	0.964	0.810	0.880	0.995	20.97
1D CNN	0.961	0.864	0.321	0.468	0.891	51.99
LSTM	0.963	0.882	0.357	0.509	0.930	1145.83

Analysis of the results:

Ensemble learning models are comprehensively superior: the F1-scores of random forest and XGBoost both exceed 0.88 and their AUC values are close to 0.99, significantly outperforming deep learning models in distinguishing malicious transactions. The reason is that the Elliptic dataset mainly consists of structured statistical features and graph features, where tree models have a stronger capability in mining nonlinear correlations among features, fitting the data characteristics.

XGBoost has the best overall cost-effectiveness: its detection accuracy is basically the same as that of random forest, but the training time is only 30% of random forest and the single-sample inference speed is about 6 times faster, showing obvious engineering deployment advantages in real-time transaction detection scenarios.

Deep learning models suffer from serious missed detection: the overall accuracy of CNN and LSTM is high, but the recall is below 40%. The models tend to classify boundary samples as the majority class (licit transactions) and cannot effectively identify minority-class malicious samples. The essential reason is that structured features are forcibly used as sequence inputs, failing to give play to the feature extraction advantages of deep learning models, and class imbalance further amplifies the missed detection problem.

High AUC conforms to the scenario characteristics: the AUC of all four models is higher than the accuracy, indicating that the models have a good ranking ability for positive and negative samples; the recall can be improved within an acceptable false positive range by lowering the classification threshold, adapting to different business risk preferences.

5.2. Smart Contract Vulnerability Detection Results and Analysis

The smart contract vulnerability detection results based on Opcode sequences are shown in Table 3:

Table 3. Smart contract vulnerability detection results.

Model	Accuracy	Precision	Recall	F1-score	AUC	Training time /s
Random Forest	0.880	0.909	0.833	0.870	0.987	4.47
XGBoost	0.960	0.923	1.000	0.960	0.968	4.48
1D CNN	0.920	1.000	0.833	0.909	0.981	1.66
LSTM	0.400	0.200	0.083	0.118	0.583	2.75

Analysis of the results:

The test set contains 25 samples in total (12 vulnerable contracts and 13 benign contracts), derived from a stratified 7:3 split of the 82-sample dataset. Given this limited sample size, the full-score metrics, namely the recall of 1.000 achieved by XGBoost and the precision of 1.000 achieved by CNN, should be interpreted with caution, as small-sample evaluation results may fluctuate considerably across different random splits. Nevertheless, the consistent performance of XGBoost across all four metrics and the stable relative ranking of the models still support the main conclusions drawn from the comparison.

XGBoost performs best: on the small-sample vulnerability dataset, XGBoost achieves 96% accuracy and 100% recall, realizing zero missed detection of vulnerability samples with only a few false positives, fully meeting the business

requirement of "rather false positives than missed detection" in vulnerability detection.

CNN has practical value: the precision of one-dimensional CNN reaches 100% with no false positives and only a small number of missed detections; in scenarios with low tolerance for false positives, the recall and precision can be balanced by threshold adjustment, and the overall performance is close to that of traditional ensemble models, verifying the ability of convolution operations to extract local vulnerability patterns of Opcode.

LSTM suffers from underfitting: the AUC of LSTM is only 0.583, close to the level of random guessing. The core reason is that the current dataset is limited in sample size, while LSTM has a large number of parameters and high data requirements, so it cannot fully learn the temporal dependencies of Opcode sequences; meanwhile, vulnerability features are more reflected in local patterns, and the gain of long-range temporal dependencies is limited.

5.3. Comprehensive Performance Comparison of Multiple Models

The comprehensive performance comparison of the four models in the two detection tasks is shown in Figure 4. XGBoost and random forest maintain leading F1-scores and AUC values in both tasks, while the performance of deep learning models fluctuates greatly with data scale and data type.

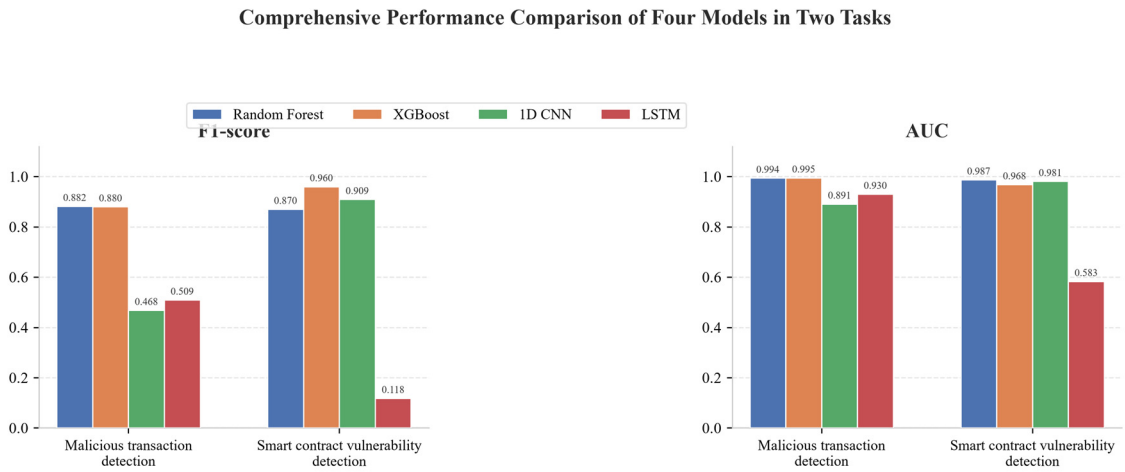


Figure 4. Comprehensive performance comparison of four models in two tasks

Combined with the experimental results of the two tasks, three general conclusions can be drawn.

Ensemble tree models have the strongest universality: XGBoost and random forest achieve the best comprehensive performance on both structured numerical data and Opcode sequence data, and their adaptability to data scale and data type is far superior to that of deep learning models, making them the first choice for blockchain security detection with small and medium samples.

The scenario boundary of deep learning models is clear: one-dimensional CNN is competitive on sequence data and can effectively capture local features; however, LSTM has extremely low cost-effectiveness on small samples and can only reflect temporal advantages on large-scale, long-sequence datasets.

There is a trade-off between efficiency and accuracy: XGBoost achieves the optimal balance between accuracy and speed in both tasks, with training and inference efficiency far higher than other models, making it more suitable for engineering deployment and real-time detection scenarios.

5.4. Experimental Problems and Optimization Suggestions

5.4.1. Existing Problems

Class imbalance: both datasets have imbalanced positive and negative sample ratios, causing the models to favor the majority class, and the recall of minority classes needs improvement.

Limited contract dataset scale: the small sample size of the smart contract vulnerability dataset prevents complex models such as LSTM from being fully trained, and their performance does not reach the theoretical upper limit.

Single feature dimension: the current vulnerability detection only uses Opcode sequence features without combining syntax tree features and business semantic features of contract source code, and the feature dimension needs to be enriched.

5.4.2. Optimization Suggestions

First, to address the problems of class imbalance and limited sample size, SMOTE oversampling and class weight strategies are introduced on the basis of the existing datasets, and the scale of smart contract vulnerability samples is further expanded, focusing on verifying the recall improvement of deep learning models such as LSTM under sufficient sample conditions [11].

Second, to address the problem of single feature dimension in vulnerability detection, abstract syntax tree features and function call graph features of contract source code are fused on the basis of Opcode sequence features, and an attention mechanism is introduced into LSTM to focus on key opcodes, improving the detection accuracy under small-sample conditions.

6. Conclusion

Focusing on two core tasks in blockchain security, namely malicious transaction identification and smart contract vulnerability detection, this paper constructs a multi-model comparative detection framework and completes a systematic comparative experiment between traditional ensemble learning and deep learning models based on the Elliptic dataset and a self-constructed vulnerability dataset. The experimental results show that gradient boosting tree models

represented by XGBoost achieve the best comprehensive performance in both tasks, combining high accuracy and high efficiency and possessing strong practical value; one-dimensional CNN performs well in Opcode sequence detection and can serve as a lightweight deep learning solution; LSTM performs poorly in the current small-sample scenario and needs to be combined with larger datasets to give play to its temporal advantages. Furthermore, the complete experimental pipeline, including data sources, preprocessing steps, model structures and parameter settings, is documented in detail in Sections 3-5 to support reproducibility, which serves as one of the core objectives of this work. Future work will further expand the dataset scale and enrich feature dimensions on the basis of the present framework to improve the accuracy and generalization ability of blockchain security detection.

References

[1] Su, X., Song, J. N., & Luo, S. (2025). User attack behavior detection based on PoS consensus. Artificial Intelligence and Robotics Research, 14(4), 855–867.

[2] Liu, D. H., & Xie, J. R. (2025). Artificial intelligence based mobile Internet information encryption technology and network security technology. Information Recording Materials, 26(8), 166–168.

[3] Qian, C. (2023). Smart contract vulnerability detection based on deep learning. Southeast University.

[4] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.

[5] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York.

[6] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444.

[7] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

[8] Weber, M., Domeniconi, G., Chen, J., et al. (2019). Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. In KDD Workshop on Anomaly Detection in Finance. Anchorage.

[9] Feist, J., Grieco, G., & Groce, A. (2019). Slither: A static analysis framework for smart contracts. In 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB) (pp. 8–15). Montreal.

[10] SWC Registry. (2019). Smart Contract Weakness Classification and Test Cases. https://swcregistry.io.

[11] Chawla, N. V., Bowyer, K. W., Hall, L. O., et al. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321–357.