Online ISSN: 3104-9338
Print ISSN: 3104-932X

A Binary Classification Detection Method for Smart Contract Honeypots Based on LSTM-Attention-CNN

Chenran Xi[1*], Handong She[2]

[1] Taiyuan Normal University

[2] Taiyuan University of Science and Technology

Received: April 13, 2026

Revised: April 14, 2026

Accepted: April 15, 2026

Published online: April 25, 2026

To appear in: *International Journal of Advanced AI Applications,* Vol. 2, No. 5 (May 2026)

* Corresponding Author: Chenran Xi (1215819301@qq.com)

Abstract. To address the technical bottlenecks of existing Smart Contract Honeypot detection methods, such as reliance on expert rules, low detection efficiency, and difficulty in identifying new honeypots, this paper proposes a binary classification detection method for Smart Contract Honeypots based on the fusion of Long Short-Term Memory (LSTM) network, Attention mechanism, and Convolutional Neural Network (CNN). Taking smart contract source code as the research object, this method converts code into trainable sequence data through data preprocessing, and designs a hybrid LSTM-Attention-CNN model to jointly capture temporal dependency features and local key patterns of code, so as to accurately distinguish Smart Contract Honeypots from normal contracts. The cross-entropy loss function and Adam optimization algorithm are introduced, combined with the early stopping strategy to avoid model overfitting and improve detection stability. Experiments are carried out based on the Ethereum smart contract dataset. The results show that the accuracy, recall rate and F1-score of the proposed model on the test set reach 98.97%, 98.82% and 98.89% respectively. Compared with single LSTM, CNN and BLSTM-ATT models, the detection performance is significantly improved, and the detection efficiency meets the requirements of large-scale batch contract detection, providing an efficient and reliable technical means for smart contract security audit.

Keywords: *Smart Contract; Smart Contract Honeypot; LSTM; Attention Mechanism; TextCNN*

1. Introduction

Since the release of the Bitcoin whitepaper in 2008 [1], blockchain technology has gradually

penetrated from the digital currency field to multiple fields such as Decentralized Finance (DeFi), supply chain management, and Non-Fungible Tokens (NFT), becoming an important support for the development of the digital economy. As the core component of blockchain technology, smart contracts have greatly expanded the application boundaries of blockchain with the characteristics of automatic execution, immutability, transparency and traceability [2]. According to DeFiPulse statistics, by the end of 2024, the total value of locked assets in various DeFi protocols around the world has exceeded 50 billion US dollars, highlighting the core position of smart contracts in the digital economy.

However, once deployed, smart contract code cannot be modified and directly manages a large number of digital assets, so its security vulnerabilities can easily lead to serious economic losses. The DAO attack in 2016 caused the theft of about 60 million US dollars worth of Ether, directly leading to a hard fork of the Ethereum blockchain; the Ronin Network cross-chain bridge attack in 2022 resulted in losses of more than 600 million US dollars [3]. Such security incidents not only damage users' property security, but also seriously undermine the credibility of the blockchain ecosystem. Among many smart contract security threats, Smart Contract Honeypots, as a new type of fraud, have become an important hidden danger endangering the blockchain ecosystem due to their strong concealment and high deception [4].

Smart Contract Honeypots lure attackers to invest funds to exploit "vulnerabilities" by deliberately embedding fake vulnerabilities, and finally steal attackers' funds through hidden logic [5]. Different from traditional vulnerability attacks, the deception mechanism of Smart Contract Honeypots relies on the characteristics of Solidity language and the underlying mechanism of Ethereum Virtual Machine (EVM), and constantly evolves new variants, bringing great challenges to detection work. Existing Smart Contract Honeypot detection methods are mainly divided into three categories: first, rule-based methods relying on experts, which require manual definition of detection patterns, lag behind new honeypots in rule update, and have a high missed detection rate [6]; second, symbolic execution-based methods, such as the HONEYBADGER tool [5], which have high detection accuracy but suffer from path explosion, low detection efficiency, and cannot adapt to large-scale contract detection scenarios; third, methods based on a single deep learning model, such as BLSTM-ATT vulnerability detection [12], the former loses code semantic information, and the latter is not optimized for Smart Contract Honeypots, resulting in limited detection performance.

Deep learning technology has unique advantages in sequence data processing and feature extraction. Among them, LSTM is good at capturing temporal dependencies of sequence data

and suitable for processing context associations of smart contract source code; Attention mechanism can adaptively focus on key code segments and improve the pertinence of feature extraction; TextCNN can effectively capture key patterns of local features of code [7]. Based on this, this paper fuses the three to design a hybrid LSTM-Attention-CNN model specially used for binary classification detection of Smart Contract Honeypots and normal contracts, solving the problems of low efficiency and insufficient accuracy of existing methods. Experimental verification shows that this method outperforms existing mainstream methods in both detection performance and efficiency, providing a new technical path for smart contract security detection.

2. Related Technologies

2.1. Core Characteristics of Smart Contract Honeypots

A Smart Contract Honeypot is a type of malicious smart contract disguised as having exploitable vulnerabilities for fraud purposes, and its core characteristics are reflected in two aspects: "deceptiveness" and "concealment". According to the classification by Torres et al. [5], Smart Contract Honeypots are mainly divided into 8 types, among which hidden state update type (accounting for 46.6%), balance disorder type, and inheritance disorder type are the most common. The common feature of such contracts is that there are obvious "exploitable vulnerabilities" on the surface (such as abnormal balance check logic and function call ambiguity), but in fact, hidden code (such as inline assembly and concealed transfer logic) is used to tamper with contract state or steal funds. Their source code contains specific keywords and syntax patterns, providing a feature basis for deep learning-based detection.

2.2. Principle of LSTM Network

Long Short-Term Memory (LSTM) network is an improved model to solve the problems of gradient disappearance and gradient explosion of traditional Recurrent Neural Network (RNN). It realizes effective memory and screening of sequence historical information by introducing three control gates: input gate, forget gate and output gate [8]. The mathematical expressions of its core structure are as follows:

$$f_t=\sigma\left(W_f\cdot[h_{t-1},x_t]+b_f\right)$$

$$i_t=\sigma(W_i\cdot[\mathrm{h_{t-1}},\mathrm{x_t}]+b_i)$$

$$\tilde{C}_t=tanh(W_C\cdot[\mathrm{h_{t-1}},\mathrm{x_t}]+b_C)$$

$$C_t=f_t\odot C_{t-1}+i_t\odot\tilde{C}_t$$

$$o_t=\sigma(W_o\cdot[h_{t-1},x_t]+b_o)$$

$$h_t=o_t\odot tanh(C_t)$$

Where f_t, i_t and o_t are the outputs of forget gate, input gate and output gate respectively; C_t and $\tilde{C}_t$ are the current cell state and candidate cell state respectively; h_t and h_{t-1} are the current and previous hidden states respectively; x_t is the input at the current moment; W_f, W_i, W_C and W_o are weight matrices; b_f, b_i, b_C and b_o are bias terms; σ is the sigmoid activation function; tanh is the hyperbolic tangent activation function; $\odot$ is element-wise multiplication.

2.3. Attention Mechanism

Attention mechanism calculates the weight of each element in the input sequence to focus on features more important to the task and suppress interference from irrelevant information [9]. This paper adopts the additive attention mechanism. Its core is to map hidden states to the same dimension through linear transformation, and then calculate attention weights through the softmax function. The mathematical expressions are as follows:

$$e_{it}=v_a^T tanh(W_a h_i+U_a h_t)$$

$$\alpha_{it}=\frac{exp(e_{it})}{\sum_{k=1}^{T} exp(e_{ik})}$$

$$h_{att}=\sum_{i=1}^{T}\alpha_{it}h_i$$

Where e_{it} is the correlation score between the i-th hidden state and the t-th moment; v_a, W_a and U_a are trainable parameters; α_{it} is the attention weight; h_{att} is the fused feature output by the attention mechanism; T is the sequence length.

2.4. TextCNN Model

TextCNN captures local features of sequence data through convolution operations, and is suitable for extracting local key patterns in smart contract source code (such as hidden transfer and assembly instructions) [10]. Its core process is: input the embedded sequence data into convolution kernels of different sizes, extract local features through convolution and pooling operations, and finally splice them into a global local feature vector. The mathematical expression of convolution operation is as follows:

$$c_j=tanh(W_k\cdot x_{i:i+k-1}+b_k)$$

Where c_j is the output of the j-th convolution kernel; W_k is the weight of the convolution

kernel with size k; $x_{i:i+k-1}$ is a local segment of length k in the input sequence; b_k is the bias term of the convolution kernel.

3. Binary Classification Detection Method for Smart Contract Honeypots Based on LSTM-Attention-CNN

3.1. Overall Framework of the Method

The overall Smart Contract Honeypot binary classification detection method proposed in this paper is divided into four stages: data preprocessing, feature extraction, model training and classification detection. The framework flow chart is shown in Figure 1.

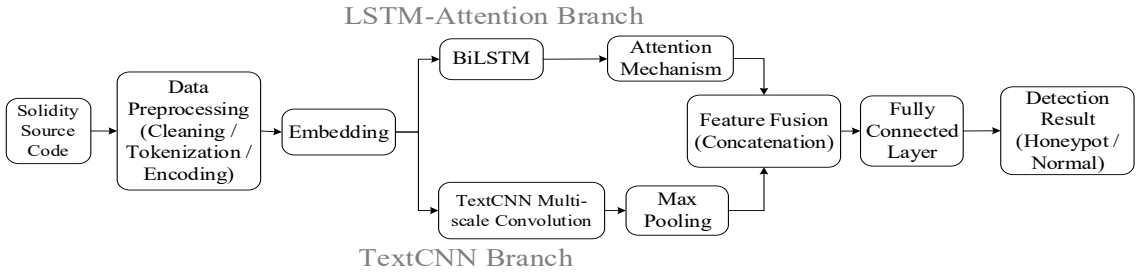

Figure 1. Overall framework flow chart of the method.

The data preprocessing stage converts the original smart contract source code into sequence data trainable by the model; the feature extraction stage captures the temporal dependency features of the code through the LSTM-Attention branch, captures the local key features through the TextCNN branch, and then performs feature fusion; the model training stage improves the model performance through optimization algorithms and regularization strategies; the classification and detection stage performs binary classification judgment on the input contract and outputs the detection results.

3.2. Data Preprocessing

Smart contract source code contains irrelevant information such as comments, blank lines and redundant characters, which needs to be converted into standardized sequence data through preprocessing. The specific steps are as follows:

(1) Source code cleaning: Remove comments, blank lines and redundant spaces, retain core code logic; unify the format of variable names and function names (such as converting camel case naming to underscore naming) to eliminate the impact of format differences.

(2) Lexical analysis and tokenization: Use Solidity lexical analysis tools to decompose the cleaned source code into tokens, including keywords (such as contract, function), identifiers, operators, constants, etc.; filter meaningless tokens (such as semicolons, brackets) to obtain

pure token sequences.

(3) Vocabulary construction and sequence encoding: Count the occurrence frequency of all tokens, retain tokens with occurrence times ≥ 3 to construct a special vocabulary; convert token sequences into numerical sequences using One-Hot encoding, and use a unified "unknown token" encoding for tokens not in the vocabulary; unify all sequences to a fixed length L (L=200 in this paper) through padding or truncation to obtain standardized input sequence X=$[x_1, x_2, \ldots, x_L]$, where x_i is the encoding value of the i-th token.

3.3. Hybrid Model Architecture Design

The hybrid LSTM-Attention-CNN model proposed in this paper is composed of several essential components, including an embedding layer, an LSTM-Attention branch, a TextCNN branch, a feature fusion layer, and a classification layer. This meticulously designed architecture facilitates the effective integration of sequential data processing and attention mechanisms with convolutional neural networks, allowing for improved contextual understanding and feature extraction. The overall structure of the model is clearly illustrated in Figure 2, which provides a comprehensive visual representation of the various layers and their interactions, effectively demonstrating how these components work together to enhance performance on the specified tasks. This layered approach aims to leverage the strengths of each individual component, ultimately leading to superior results in the targeted applications.

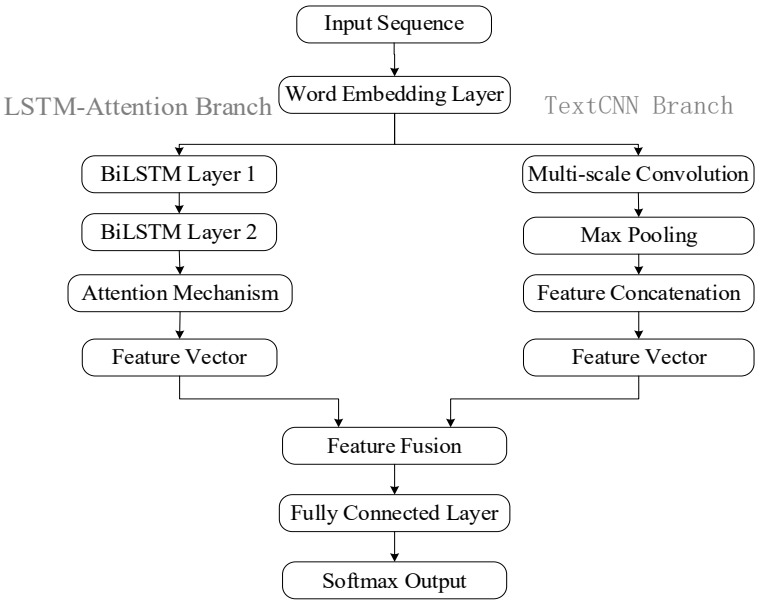

Figure 2. Structure diagram of hybrid LSTM-Attention-CNN model.

3.3.1. Embedding Layer

The embedding layer maps standardized numerical sequences into low-dimensional dense

vectors to reduce dimensionality disaster and retain semantic associations between tokens. Let the vocabulary size be V and the embedding dimension be d. The embedding layer converts the input sequence X into an embedded vector sequence $X_{emb} = [x_{emb1}, x_{emb2}, \dots, x_{embL}]$ through the trainable matrix $W_{emb} \in \mathbb{R}^{V \times d}$, where $x_{embi} \in \mathbb{R}^d$ is the embedding vector of the i-th token.

3.3.2. LSTM-Attention Branch

This branch is used to extract temporal dependency features of smart contract source code and capture associations between code contexts (such as function call order and state variable modification logic). Input the embedded vector sequence X_{emb} into the bidirectional LSTM layer to obtain the bidirectional hidden state sequence $H_{LSTM} = [h_1, h_2, \dots, h_L]$, where $h_i = h_i^{forward}, h_i^{backward}$, which are the hidden states of forward and backward LSTM respectively.

Input H_{LSTM} into the attention mechanism layer, calculate attention weights through formulas, and obtain the vector $h_{att} \in \mathbb{R}^{2n}$ fused with key temporal features (n is the dimension of LSTM hidden layer).

3.3.3. TextCNN Branch

This branch is used to extract local key features in the source code (such as assembly, sstore and other honeypot feature token combinations). Three convolution kernels of different sizes (3, 4, 5) are adopted, with 32 convolution kernels for each size, to perform convolution operations on the embedded vector sequence X_{emb} to obtain local features of different dimensions; perform global max pooling on each convolution output to retain the most representative local features; splice all pooled features to obtain the local feature vector $h_{cnn} \in \mathbb{R}^{3 \times 32}$.

3.3.4. Feature Fusion and Classification

Splice the temporal feature vector h_{att} and the local feature vector h_{cnn} to obtain the fused feature vector $h_{fusion} = [h_{att}, h_{cnn}]$; input the fused feature into the fully connected layer, perform feature mapping through the ReLU activation function, and introduce a Dropout layer (dropout rate=0.5) to prevent overfitting; finally output the binary classification probability through the sigmoid activation function. When the probability ≥ 0.5, it is judged as a Smart Contract Honeypot, otherwise it is a normal contract.

3.4. Model Training Strategy

3.4.1. Loss Function

The cross-entropy loss function is used to measure the difference between the model predicted value and the real label, which is suitable for binary classification tasks. The mathematical expression is as follows:

$$L = -\frac{1}{N}[y_i log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)]$$

Where N is the number of samples; y_i is the real label of the i-th sample (0=Normal Contract, 1=Smart Contract Honeypot); $\hat{y}_i$ is the probability that the model predicts the i-th sample is a Smart Contract Honeypot.

3.4.2. Optimization Algorithm and Early Stopping Strategy

The Adam optimization algorithm is adopted to minimize the loss function. This algorithm combines momentum gradient descent and adaptive learning rate adjustment, with fast convergence speed and good stability [11]. The learning rate is set to 0.001, the attenuation coefficients $\beta_1 = 0.9$, $\beta_2 = 0.999$, and the regularization parameter $\epsilon = 1e-8$.

The early stopping strategy is introduced to avoid model overfitting: divide the dataset into training set and validation set (ratio 8:2), monitor the validation set loss during training, stop training when the validation set loss does not decrease for 5 consecutive epochs, and save the current optimal model parameters.

3.4.3. Hyperparameter Selection

The optimal hyperparameters of the model are determined through grid search. The core hyperparameters are set as follows: embedding dimension d=128, LSTM hidden layer dimension n=64, TextCNN convolution kernel sizes 3/4/5, number of convolution kernels for each size 32, number of neurons in fully connected layer 128, dropout rate=0.5, batch size=32, maximum training epochs=50.

4. Experiments and Result Analysis

4.1. Experimental Environment and Dataset

4.1.1. Experimental Environment

Experimental hardware environment: CPU is Intel Core i7-12700H, memory 32GB, GPU is NVIDIA RTX 3060 (6GB); software environment: operating system is Ubuntu 20.04, programming language is Python 3.8, deep learning framework is PyTorch 1.12.0, lexical

analysis tool is SolidityParser, data processing tools are Pandas and Numpy.

4.1.2. Dataset Construction

The experimental dataset is derived from public contracts on the Ethereum mainnet and existing research datasets [5,12]. A total of 10,000 smart contract source codes are collected, including 3,000 Smart Contract Honeypots (covering 8 main types) and 7,000 normal contracts (including DeFi protocols, NFT contracts, ordinary transfer contracts, etc.). All contracts are written in Solidity language, with versions covering 0.4.x-0.8.x, ensuring the diversity and representativeness of the dataset.

The dataset is divided into training set (8,000), validation set (1,000) and test set (1,000) according to the ratio of 8:1:1. Random shuffling and deduplication strategies are adopted to avoid model deviation caused by uneven data distribution.

4.1.3. Evaluation Metrics

Accuracy, Recall, Precision and F1-score are used as model evaluation metrics. The calculation formulas are as follows:

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

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

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

$$F1=2\times\frac{Precision\times Recall}{Precision+Recall}$$

Where TP (True Positive) is the number of contracts correctly judged as honeypots; TN (True Negative) is the number of contracts correctly judged as normal; FP (False Positive) is the number of normal contracts misjudged as honeypots; FN (False Negative) is the number of Smart Contract Honeypots misjudged as normal.

4.2. Experimental Results and Analysis

4.2.1. Model Performance Comparison

To verify the superiority of the proposed hybrid LSTM-Attention-CNN model, it is compared with the single LSTM model, single TextCNN model and BLSTM-ATT model (for reentrancy vulnerability detection [12]) in comparative experiments. The experimental results are shown in Table 1.

It can be seen from Table 1 that the LSTM-Attention-CNN model proposed in this paper

outperforms the comparison models in all evaluation metrics: the accuracy is increased by 6.62 percentage points compared with the single LSTM and 3.09 percentage points compared with BLSTM-ATT; the F1-score reaches 98.89%, indicating that the model has excellent binary classification performance. Although the single contract detection time is slightly longer than that of single models, the detection speed of 23.1 ms can still meet the demand of large-scale batch contract detection (about 43 contracts can be detected per second).

Table 1. Performance comparison of different models on the test set.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Single contract detection time (ms)
Single LSTM	92.35	91.82	92.17	92.00	18.6
Single TextCNN	93.72	93.15	93.58	93.36	12.3
BLSTM-ATT	95.88	95.42	95.67	95.54	21.5
Proposed LSTM-Attention-CNN	98.97	98.75	98.82	98.89	23.1

Analysis reasons: the single LSTM model can only capture temporal features and cannot effectively identify local key patterns; the single TextCNN model is good at local feature extraction but lacks the capture of context dependencies; the BLSTM-ATT model introduces the attention mechanism but is not optimized for Smart Contract Honeypot features and does not fuse local features; while the proposed model combines temporal features and local features, and focuses on key honeypot code segments through the attention mechanism, significantly improving detection accuracy.

4.2.2. Ablation Experiment

To verify the effectiveness of each component of the model, ablation experiments are carried out by removing the Attention mechanism and TextCNN branch respectively, and the model performance changes are compared. The results are shown in Table 2.

Table 2. Ablation experiment results.

Model Variant	Accuracy (%)	Recall (%)	F1-score (%)
Complete model (LSTM-Attention-CNN)	98.97	98.82	98.89
Remove Attention mechanism (LSTM-CNN)	96.53	96.38	96.45
Remove TextCNN branch (LSTM-Attention)	95.92	95.77	95.84

It can be seen from Table 2 that after removing the Attention mechanism, the model accuracy decreases by 2.44 percentage points and the F1-score decreases by 2.44 percentage points, indicating that the attention mechanism can effectively focus on key honeypot features and improve model discrimination ability; after removing the TextCNN branch, the model accuracy decreases by 3.05 percentage points and the F1-score decreases by 3.05 percentage points, indicating that the local features extracted by TextCNN play an important role in honeypot

detection. The fusion of the two components can achieve complementary advantages and significantly improve model performance.

4.2.3. Comparison with Existing Methods

The proposed method is compared with existing mainstream honeypot detection methods (HONEYBADGER [5], Camino et al.'s method [13]). The results are shown in Table 3.

It can be seen from Table 3 that the proposed method is significantly superior to existing methods in accuracy and detection efficiency: the accuracy is increased by 13.97 percentage points compared with HONEYBADGER; the detection efficiency reaches 43.3 contracts/second, much higher than HONEYBADGER. It does not rely on transaction history data and is suitable for real-time detection of newly deployed contracts, with stronger practical application value.

Table 3. Performance comparison with existing methods.

Method	Core Technology	Accuracy (%)	Detection Efficiency (contracts/second)	Application Scenario
HONEYBADGER [5]	Symbolic execution + heuristic rules	85.00	0.8	Small-scale accurate detection
Camino et al. [13]	Transaction behavior analysis + machine learning	89.50	12.5	Contracts with transaction history
Proposed method	LSTM-Attention-CNN + source code	98.97	43.3	Large-scale batch detection

5. Conclusion and Prospect

5.1. Conclusion

Aiming at the problems of low efficiency, insufficient accuracy and difficulty in adapting to large-scale scenarios of existing Smart Contract Honeypot detection methods, this paper proposes a binary classification detection method for Smart Contract Honeypots based on LSTM-Attention-CNN. The main work is as follows:

(1) A set of data preprocessing process for smart contract source code is designed. Through cleaning, lexical analysis and sequence encoding, the original code is converted into standardized training data, eliminating the interference of format differences and irrelevant information and laying a foundation for model training.

(2) A hybrid LSTM-Attention-CNN model is constructed, which combines the temporal

feature extraction ability of LSTM, the key feature focusing ability of Attention mechanism and the local feature extraction ability of TextCNN to comprehensively capture Smart Contract Honeypot features and improve binary classification detection accuracy.

(3) Adam optimization algorithm, cross-entropy loss function and early stopping strategy are introduced to optimize the model training process, avoid overfitting, and improve the stability and generalization ability of the model; the optimal hyperparameters are determined through grid search to further optimize model performance.

Experimental results show that the accuracy of the proposed model on the test set reaches 98.97% and the F1-score reaches 98.89%. The detection efficiency meets the requirements of large-scale batch detection. Compared with existing methods, it has significant advantages and can effectively identify various Smart Contract Honeypots, providing technical support for smart contract security audit.

5.2. Research Limitations and Future Prospect

There are still some limitations in this research: first, model training relies on a large amount of labeled data, and the detection performance for new unlabeled Smart Contract Honeypots needs to be improved; second, the model is only designed for Solidity language contracts and has insufficient adaptability to other smart contract languages (such as Vyper); third, the impact of contract obfuscation technology on detection performance is not considered.

Future research directions mainly include:

(1) Introduce semi-supervised learning or transfer learning technology to reduce the dependence on labeled data and improve the detection ability of new Smart Contract Honeypots;

(2) Expand model adaptability, optimize data preprocessing and feature extraction modules to support multi-language smart contract detection;

(3) Research confrontation methods of contract obfuscation technology to improve the anti-interference ability of the model through feature enhancement;

(4) Combine binary classification detection with fine-grained type judgment to further improve the Smart Contract Honeypot detection system.

References

[1] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Working Paper.*
[2] Buterin, V. (2014). Ethereum: A next-generation smart contract and decentralized application platform. *White Paper.*
[3] Ronin Network. (2022). Ronin bridge exploit post-mortem. *Technical Report.*

[4] Torres, C. F., Steichen, M., & State, R. (2019). The art of the scam: Demystifying honeypots in Ethereum smart contracts. *In Proceedings of the 28th USENIX Security Symposium* (pp. 1591–1607). USENIX Association.
[5] Torres, C. F., & Steichen, M. (2019). The art of the scam: Demystifying honeypots in ethereum smart contracts. *In 28th USENIX Security Symposium (USENIX Security 19)* (pp. 1591-1607).
[6] Liu, Z., Jiang, M., Zhang, S., Zhang, J., & Liu, Y. (2023). A smart contract vulnerability detection mechanism based on deep learning and expert rules. *IEEE Access*, 11, 77990-77999.
[7] Kim, Y. (2014). Convolutional neural networks for sentence classification. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (pp. 1746–1751).
[8] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
[9] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv*:1409.0473.
[10] Zhang, Y., & Wallace, B. (2015). A sensitivity analysis of convolutional neural networks for sentence classification. *arXiv preprint arXiv*:1510.03820.
[11] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv*:1412.6980.
[12] Rahardja, U., & Aini, Q. (2026). Enhancing Blockchain Security Through Smart Contract Vulnerability Classification Using BiLSTM and Attention Mechanism. *Journal of Current Research in Blockchain*, 3(1), 28-45.
[13] Otoum, Y., Asad, A., & Nayak, A. (2025). Blockchain meets adaptive honeypots: A trust-aware approach to next-gen iot security. *IEEE Transactions on Network Science and Engineering.*