W2V-SA: A Deep Neural Network-based Approach to Smart Contract Vulnerability Detection

Zhigang Xu [1], Chaojun Li [1], Hongmu Han [1], Xinhua Dong [1] , Zhiqiang Zheng [2], Haitao Wang[2], Jiaxi Zhang [2] , Xingxing Chen [1] and Orest Kochan [1]

[1] *Hubei University of Technology, Wuhan,430000, China*
[2] *Narcotics Control Bureau of Department of Public Security of Guangdong Province, Guangzhou, 510050, China*

Abstract

As blockchain applications increase continuously, the number of smart contracts has exploded. However, an increasing number of hackers are mining the vulnerabilities hidden in smart contracts and using them to attack blockchain networks, causing serious consequences. To solve the problem of smart contract vulnerability, we propose W2V-SA, a static analysis method based on deep neural networks for vulnerability detection in smart contracts. This approach appraises the smart contracts before they are deployed to the blockchain network, to detect vulnerabilities timely. Firstly, converts smart contracts into vectors as input data by the word embedding method. Secondly, the hybrid deep neural network model is used to extract and classify features from the input data. Finally, efficient and accurate detection is achieved for six vulnerability types on real smart contracts. The experimental results indicate that the average accuracy of this method is more than 94% in smart contract vulnerability detection. The experimental results demonstrate the effectiveness of W2V-SA.

Keywords

Smart Contracts, Vulnerability Detection, Neural Networks, Blockchain,

1. Introduction

Smart contracts [1], as a core component of blockchain technology, play an important role in the application of blockchain technology. In the early stage, smart contracts were a scripting language for Bitcoin, mainly used to limit the input and output of transactions and implement some simple logical judgments, and they were very difficult to write and use. With the emergence of the Ethereum blockchain, Ethereum adopted smart contracts based on the solidity programming language, making it easy for developers to write and use smart contracts and implement more complex functions [2]. At this stage, more and more industry sectors are introducing blockchain technology, for example, in the logistics industry [3], blockchain technology makes the traceability of express delivery faster and more credible. In the field of information security [4], blockchain technology can ensure that evidence is not tampered with and is permanently preserved. Blockchain smart contracts have not only brought great innovation to the industry but also brought great convenience to human life [5].

However, blockchain has been under attack at any time since its inception. In recent years, more and more hackers have exploited vulnerabilities in smart contracts to attack blockchain networks, causing huge economic losses and trust crises in blockchain networks [6]. In 2016, hackers exploited vulnerabilities in smart contracts, leading to attacks on DAO projects on the Ether platform and the theft of more than $50 million in digital currency [7].Smart contracts on the CoinBene exchange were

COLINS-2023: 7th International Conference on Computational Linguistics and Intelligent Systems, April 20–21, 2023, Kharkiv, Ukraine
EMAIL: xzg@hbut.edu.cn (Zhigang Xu); lcj@hbut.edu.cn (Chaojun Li); 20181073@hbut.edu.cn (Hongmu Han); xhdong@hbut.edu.cn (Xinhua Dong); zzq1881@163.com (Zhiqiang Zheng); 13925088659@139.com (Haitao Wang); 13922336486@139.com (Jiaxi Zhang); cstar@hbut.edu.cn (Xingxing Chen); orest.v.kochan@lpnu.ua (O. Kochan)
ORCID: 0000-0003-4007-4779 (Zhigang Xu); 0009-0009-7578-7614 (Chaojun Li); 0000-0002-6909-5242 (Hongmu Han); 0009-0008-2491-1593 (Xinhua Dong); 0009-0006-0638-1440 (Zhiqiang Zheng); 0009-0006-3234-7377 (Haitao Wang); 0009-0008-4425-9942 (Jiaxi Zhang); 0009-0004-0922-4198 (Xingxing Chen); 0000-0002-3164-3821 (O. Kochan)

vulnerability, leading to the theft of a large amount of digital currency [8]. 2020, the UniSwap platform smart contract vulnerability was exploited by hackers, stealing over $500,000 in digital currency [9].The security of smart contracts is closely related to the security of the blockchain network. On the one hand, smart contracts as an important part of the blockchain ecosystem, and the security of smart contracts directly affects the security and stability of the blockchain ecosystem. On the other hand, with the rapid development of blockchain technology, the application scenarios of smart contracts are expanding, and only by ensuring the security of smart contracts can they be better applied. Therefore, it is necessary to conduct security audits on smart contracts before deploying them to blockchain networks [10].

To prevent further smart contract security incidents, researchers, scholars, and companies are actively researching and developing more secure and reliable smart contract detection technologies and tools to ensure the security and stability of blockchain systems. However, most smart contract vulnerability detection methods still suffer from significant drawbacks, including low efficiency, inability to perform high-volume smart contract vulnerability detection, low automation, and long detection times. This paper examines the literature published by domestic and foreign scholars on smart contract vulnerability detection, identifies the shortcomings of existing research, and proposes an automated vulnerability detection method for (Ethereum) smart contracts named W2V-SA (Word2vec-Solidity Attention) based on the deep neural network method. By integrating the benefits of different neural networks, W2V-SA accurately and comprehensively extracts features, surpassing single smart contract vulnerability detection models, with higher detection efficiency and the capability of detecting a large number of smart contract vulnerabilities. The primary contributions of this paper are as follows:

1. This paper proposes a deep neural network-based vulnerability detection method for smart contracts (W2V-SA), which develops specific vulnerability identification checking models for different vulnerability types of smart contracts.
2. Combining different neural network models and word embedding methods, the scheme proposed in this paper can effectively improve the efficiency and accuracy of smart contract vulnerability detection.
3. The results of extensive experimental comparisons indicate that the W2V-SA model, which combines the advantages of several neural network models, surpasses individual neural network models and machine learning algorithms in detecting smart contract vulnerabilities. The three performance evaluation metrics- precision, recall, and F1-score, all exhibit an improvement of over 4%, signifying superior detection performance.

The structure of this paper comprises five sections. The first section serves as an introduction, presenting the background and motivation of the research. In the second section, an overview of related work is provided. The third section outlines the proposed method and architecture. The results of the experiment are presented in the following section. Finally, the fifth section is the conclusion and provides an outlook for future research.

2. Related Work

This section covers several fundamental concepts related to the research, such as Ethereum smart contract vulnerabilities, neural network models, and existing methods for detecting smart contract vulnerabilities are examined.

2.1. Smart Contracts Vulnerability

The smart contract is essentially a piece of code written by a human that runs automatically on a blockchain network, which completes the given operations according to predefined rules [11]. However, the code written by humans based on known knowledge is difficult to guarantee perfect application in different scenarios, and it is also difficult to avoid the emergence of smart contract vulnerabilities. Smart contract vulnerabilities are flaws or weaknesses in the design, implementation, or use of smart contracts. These weaknesses can be exploited by malicious attackers and lead to unexpected behavior in smart contracts, such as theft of funds and network crashes [6]. The reasons for vulnerabilities arising from smart contracts can be summarized as follows.

- Code quality issues

A smart contract is essentially a piece of code, and if the code is not of high quality, it is susceptible to a variety of vulnerabilities. For example, there can be unvalidated input in the code, dead loops, stack overflows, and other issues that can lead to contract execution failures or attacks.

- Logic vulnerabilities

The writing of smart contracts often involves a lot of complex business logic, and if the writer does not consider all possible scenarios, logic vulnerabilities may arise. For example, when writing a multi-signature contract, if the writer does not consider the case where everyone is unable to sign, it may lead to an attack on the contract.

- External dependency issues

Smart contracts may need to rely on external services and libraries, such as cryptographic libraries, random number generators, etc. If these external dependencies are vulnerable, the security of the smart contract can be compromised.

2.2. Smart Contracts Vulnerability Detection

Currently, the mainstream detection methods of smart contract vulnerability can be divided into two types, one is static analysis, and another method is dynamic analysis. Static analysis is an analysis method of vocabulary, syntax, control flow, and data flow information before the source code is run. It does not need to compile or run smart contract code to identify common vulnerabilities. With the rise of artificial intelligence, a large number of methods of machine learning and deep learning have realized static analysis. Liu et al. [12] propose a smart contract vulnerability detection method based on a combination of deep learning and expert patterns, which converts the code into a semantic graph to extract deep graph features and later fuses local expert patterns for prediction. Slither [13] is a static analysis framework that converts smart contract code to SlithIR, which is the method that can simultaneously be a platform for finding bugs, suggesting code optimizations, and improving the understanding of a given smart contract code. Contractward [14] converts smart contract source code into opcodes and builds models using five machine learning algorithms and two sampling algorithms to achieve efficient detection of six smart contract vulnerabilities. In summary, static methods often rely on vulnerability models constructed by human experts and vulnerability-type criteria developed by experts [15]. The high cost of manual construction and human subjectivity seriously affect the false positive rate and omission rate of smart contracts as the complexity of the code program increases. Static methods are generally only applicable to small-scale code programs and cannot cope with large and complex programs and diverse vulnerabilities.

Dynamic analysis is a method of debugging and checking the program when it running. Oyente [16] is a well-known tool for detecting smart contract vulnerabilities based on symbolic execution. This tool uses symbolic execution to traverse execution paths on the control flow graph to analyze vulnerabilities in smart contracts, including error-handling exceptions, transaction order dependencies, timestamp dependencies, and reentrancy. ConFuzzius [17] is the first tool to detect smart contract vulnerabilities using hybrid fuzzy testing, which combines evolutionary fuzzy, lightweight EVM, symbolic taint analysis, and genetic algorithms to generate inputs that satisfy complex conditions to achieve coverage of more program paths and uncover deeper vulnerabilities. Manticore [18] examines the security of smart contracts by enumeration all execution paths of the contract. The tool can detect vulnerabilities of integer overflow and uninitialized memory. Ding et al. [19] proposed a fuzzy technology-based approach to detect vulnerabilities in Hyperledger Fabric smart contracts. The method is well combined with the fuzzing tool go fuzz and has achieved better results in practice. The current mainstream dynamic analysis methods, such as symbolic execution and fuzzy testing, have great challenges. First, fuzzy testing has difficulties in finding complex vulnerabilities with long execution paths and concealment. Second, most programs need to judge the reasonableness of the input data, and blind random variation will lead to inefficiency in generating test cases. Third, in the process of fuzzy testing, different test cases may execute the same path and trigger the same vulnerabilities. The challenge with symbolic execution is path explosion. Dynamic analysis methods require exploring all executable paths in a smart contract and analyzing the dependency graph of a smart contract [20]. It has problems such

as high time overhead and unsuitability for high-volume contract detection, and these problems can lead to reduced efficiency of smart contract vulnerability detection.

2.3. Deep Neural Network Hybrid Model

In recent years, there has been a growing preference for deep neural network hybrid models. In these models, each component compresses different feature values based on its unique characteristics, resulting in more comprehensive extracted feature values compared to individual neural network models. SAMF-BiLSTM [21] is a hybrid neural network model that integrates a self-attention mechanism with a bidirectional LSTM featuring multichannel features. This model performs better in sentiment analysis tasks. Jin et al. [22] proposed a hybrid neural network model, Bi-LSTM-CRF with Self-Attention, for Korean named-entity recognition. Xu et al. [23] proposed a hybrid neural network model, Self-Attention Bi-LSTM combined with ALBERT, for detecting spam.

Since CNN does not require manual feature selection [24], feature extraction is fast and effective, and has strong generalization ability [25]. Compared with other network models, the self-attention mechanism has lower model complexity, fewer parameters, and less arithmetic power requirement, and its computation at each step does not depend on the results of the previous step, which can be processed in parallel as CNN [26]. In addition, the self-attention mechanism is not limited by the length of the text. Even if the text is long, the self-attention mechanism can obtain the global and local connections and capture the key parts of the text without losing the important information. Therefore, the paper combines these two models with a good-performing word embedding method to build a method for smart contract vulnerability detection. To demonstrate that the performance of the hybrid model is better than that of the single model, the CNN and fastText models are chosen to compare with the performance of W2V-SA. FastText [27] is a fast text classification model that is close to deep neural network classification models in terms of precision. Moreover, no pre-trained word embeddings are required in fastText compared to the deep neural networks model, speeding up training and testing while maintaining high precision. The CNN modeluses [28] word embeddings for word pre-training, which was first proposed by Kim for sentiment classification in the field of NLP, and has achieved better results.

3. Overall Framework

In order to achieve efficient and accurate detection of smart contract vulnerabilities, this paper introduces a self-attention mechanism in the neural network model to improve the feature extraction rate of the model and enhance the performance of smart contract vulnerability detection. This subsection mainly introduces the structural design of W2V-SA and the functions of each layer of W2V-SA.

3.1. Overall Method Structure

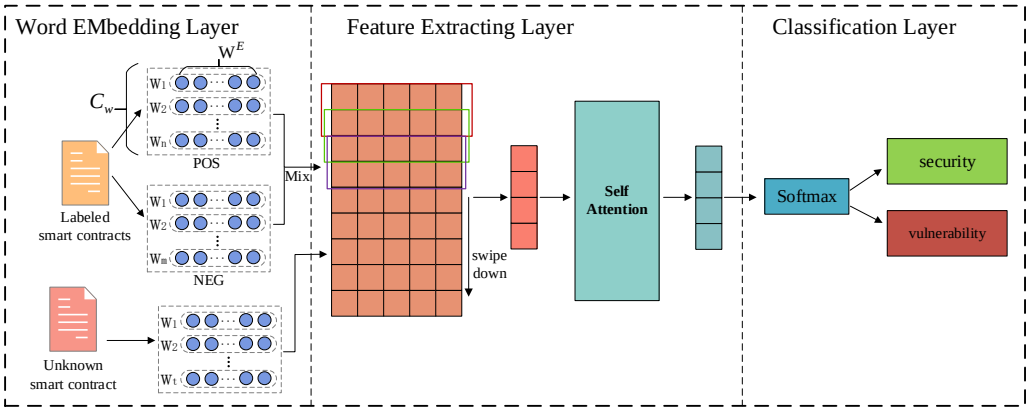

Figure 1: Structure of W2V-SA

As shown in Figure 1, the W2V-SA hybrid model consists of three layers, which are the word embedding layer, feature extraction layer, and classification layer. After the conversion of smart contract source code into opcodes, each smart contract C_w takes word w_i as the basic unit and forms a sequence of words: $\{w_1, w_2, w_3, \ldots \ldots, w_n\}$. In the word embedding layer, the word2vec model maps each word $w_i$to a multidimensional vector w_i^E. The word embeddings in the sequence are concatenated to obtain a word embedding matrix representation of the entire smart contract: $C_w = w_1^E + w_2^E + w_3^E + \cdots + w_i^E$. After converting smart contracts into word embeddings, smart contracts are classified and labeled into two classes: with vulnerability (positive) and without vulnerability (negative), and they are first mixed and disrupted, and then their features are extracted using a hybrid neural network model. In the feature extraction layer, the features extracted by the convolutional neural network are input to the self-attention layer to calculate the attention weights, analyze the relationship between the mining contexts, and continuously train the smart contracts with and without vulnerabilities for feature extraction. Finally, the training results are classified and the model is saved. To verify the validity of the model, unknown smart contracts are fed into the model for prediction to determine whether they contain some type of vulnerability.

3.2. Word Embedding Layer

Before word embedding training of smart contracts, the source code of smart contracts needs to be compiled into bytecodes and then de-compiled into opcodes. According to the specification of the Ethereum [29], each bytecode has its corresponding opcode, and there are 144 bytecodes in total corresponding to the opcode. The mapping between bytecode and opcode is shown in Table 1.

Table 1
The relationship mapping table of byte code to opcode

Bytecode	Opcode
0x00	STOP
0x01	ADD
0x02	MUL
0x03	SUB
.....	.....
0xfd	REVERT
0xfe	INVALID
0xff	SELFDESTRUCT

Table 2
Word embedding training results

Opcode	Word Embedding
PUSH1	0.27597266 0.005130794 0.00354396 0.062392443 -0.31270385 0.028506324 -0.02617409 0.09100188 0.08294053 0.13363415 -0.18524168 -0.20330547 0.07561363 -0.034679547 0.54183996 0.14461869 0.03434112 -0.20602745 -0.38273817 -0.043624587 0.0013151914 -0.2788695 0.1741333 -0.0054611214 0.037056547 0.009860475 0.21438162 0.20419425 0.08955717 -0.15953913

The opcode is a text message and cannot be entered directly into the model, we need to convert the opcode into a digital representation. We choose to use the skip-gram model in the word2vec word

embedding model to vectorize the text. In this section, 144 opcodes are trained by the skip-gram model, and after continuous testing, we choose the most suitable representation of text conversion into vectors: each opcode is mapped into a 30-dimensional vector. Taking the opcode PUSH1 as an example, the word vector training results are shown in Table 2.

3.3. Feature Extraction Layer

After the skip-gram model finishes training the smart contract word embeddings, as shown in Figure.2, the hybrid neural network model starts feature extraction of the input word embeddings.

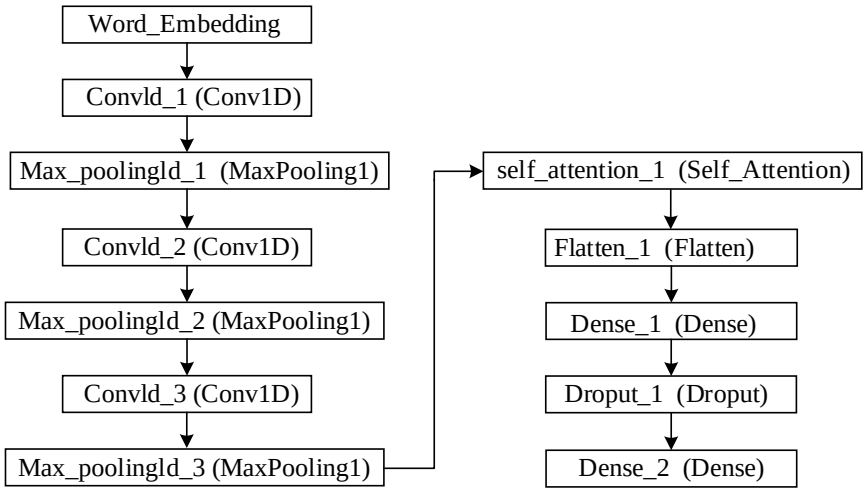

Figure 2: Feature extraction layer

In the convolutional neural network branch, As seen in Figure 2, the data output from the word embedding layer is passed into the convolutional layer, which extracts some primary features through local connectivity and weight sharing. Then it is passed to the pooling layer for down-sampling. The pooling layer can effectively reduce overfitting and improve the fault tolerance of the model. This study utilized three layers of convolutional neural networks for local feature learning. The advantage of using three layers of convolution is to reduce the number of network parameters, thereby reducing the amount of data that needs to be learned. As the number of layers increases, the model can extract more complex and abstract information during training, thus improving learning efficiency.

The self-attention mechanism model is another branch of the feature extraction layer, which is an enhanced version of the attention mechanism that relies less on external information and is proficient at capturing data features or correlations within features. By learning the correlation between different words, the self-attention mechanism selects the more relevant words from a vast amount of information. It enables better comprehension of the connections between local contexts and extraction of essential feature information while reducing the data quantity.

The text representation sequence obtained from the convolutional layer is input to the self-attention layer for attention weight calculation, where the larger the weight is, the more focused its corresponding information, and the weight represents the importance of the information. The computing process of the self-attention mechanism is shown in Figure 3. a_i in the formula denotes the input vector and w^q, w^k, w^v denotes the weight matrix. Take input vectors a_1 and a_2 as an example. first, multiply a_1 by w^q, w^k, w^v three weight matrices respectively to get q^1 (query), k^1 (key), v^1 (value), and input vector a_2 similarly, the calculation formula can be expressed as

$$q^i = a_i \cdot w^q \quad (1)$$

$$k^i = a_i \cdot w^k \quad (2)$$

$$v^i = a_i \cdot w^v \quad (3)$$

Second, the relevance between input vectors is calculated. $\alpha_{i,j}$ is represented as the relevance between input vectors a_i and a_j. According to Equation (4), vector $\alpha_{1,2}$ is obtained by multiplying

matrix q^1 and k^2. Third, the attention scores between input vectors are calculated. Third, the attention scores between the input vectors are computed. according to Equation (5), $\alpha_{1,2}$ is computed by the activation function Soft-max, and then the input vectors a_1 and a_2 are obtained as the attention scores $\alpha'_{1,2}$. Finally, the output vector b_2 is obtained by multiplying the attention scores $\alpha'_{1,2}$ with the value v^2 according to Equation (6).

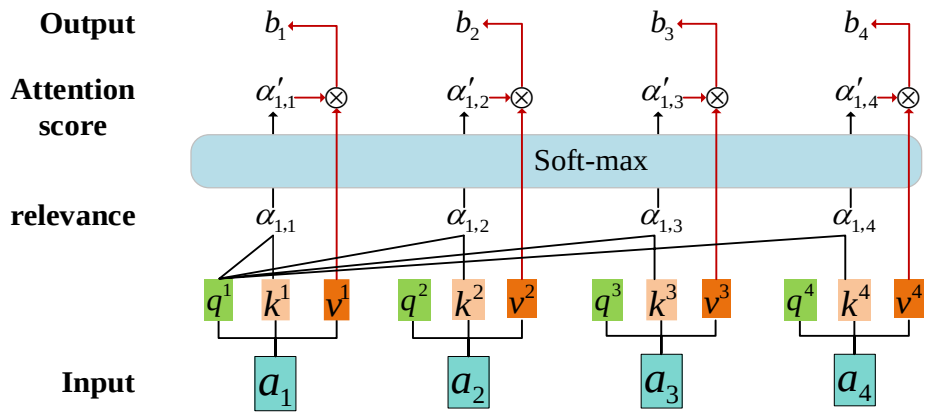

Figure 3: The self-attention mechanism computation process

$$\alpha_{i,j} = q^i \cdot k^j \tag{4}$$

$$\alpha'_{i,j} = \frac{exp(\alpha_{i,j})}{\sum_j exp(\alpha_{i,j})} \tag{5}$$

$$b_i = \sum_j v^i \cdot \alpha'_{i,j} \tag{6}$$

The subsequent layer after the self-attention layer is the Flatten layer, this layer helps to decrease the dimensionality of the final output vector. The Dropout layer is positioned after the Flatten layer, and it disregards a certain number of neurons with a certain probability in each training batch. This can significantly reduce overfitting. The final layer is the Dense layer, which maps the feature space computed in the previous layer to the sample label space. Its primary role is to integrate feature representation into a single value, which reduces the influence of feature position on classification results and enhances the entire network's robustness.

The W2V-SA model, which combines the word embedding model with the hybrid deep neural network model, is proposed in this paper. By comprehensively extracting features, W2V-SA can effectively enhance the efficiency of smart contract vulnerability detection. In the following section, we employ the W2V-SA model in actual smart contract vulnerability detection and establish the effectiveness of the proposed model through numerous experiments.

4. Experiments and Results

This subsection encompasses a series of comprehensive experiments that evaluate the efficiency of the hybrid neural network model proposed in this paper for detecting vulnerabilities in smart contracts.

4.1. Dataset

The data utilized in this paper comprises authentic smart contracts sourced from the official Ethereum website. After verification and removal of duplicate and incomplete smart contracts, we acquired a total of 98,919 smart contracts. In this study, the dataset was categorized and labeled into two groups, namely smart contracts with vulnerabilities and smart contracts without vulnerabilities. The detection of six types of smart contract vulnerabilities was the focus of this paper, including AssertFail

vulnerability, BlockTimestamp vulnerability, CheckEffects vulnerability, LowLevelCalls vulnerability, Reentrancy vulnerability, and IntegerUnderFlow vulnerability. Table 3 illustrates the distribution of the smart contract dataset.

Table 3 indicates that the distribution of smart contracts with and without vulnerabilities is imbalanced, and direct usage could lead to severe overfitting. Therefore, the dataset requires balancing before performing word embedding transformation. For imbalanced datasets, this paper utilizes a technique called ADASYN, which is an adaptive synthetic sampling method proposed by He et al. [30]. ADASYN translates the difficulty of samples into weights to facilitate the learning process and provides a higher proportion of the few difficult samples to simulate the data distribution. Table 4 displays the balanced dataset, and the distribution of the balanced dataset is more reasonable than the original dataset.

Table 3
Distribution of smart contract data sets

Types of vulnerability	Positive (With vulnerability)	Negative (Without vulnerability)	Total
AssertFail	7296	9960	17256
BlockTimestamp	5624	11785	17409
CheckEffects	6851	10557	17408
LowLevelCalls	5153	12257	17410
Reentrancy	5130	6897	12027
IntegerUnderFlow	10882	6527	17409

Table 4
Distribution of balanced smart contract dataset

Types of vulnerability	Positive (With vulnerability)	Negative (Without vulnerability)	Total
AssertFail	7296	9960	17256
BlockTimestamp	5624	6231	11855
CheckEffects	6851	7945	14796
LowLevelCalls	5153	6034	11187
Reentrancy	5130	6897	12027
IntegerUnderFlow	6697	6527	13224

4.2. Comparison of Results

The fundamental concept of the W2V-SA model is to merge word embedding with a hybrid deep neural network model to enhance the model's performance. Figure 4 illustrates the training process of the W2V-SA model, and all four curves of the model gradually stabilize after reaching 15 training iterations. Throughout the training phase, the training and test sets exhibit similar curves and are in close proximity, indicating that the model does not overfit during the training process.

The W2V-SA model is evaluated for detecting common smart contract vulnerabilities, including AssertFail vulnerability, BlockTimestamp vulnerability, CheckEffects vulnerability, LowLevelCalls vulnerability, Reentrancy vulnerability, and IntegerUnderFlow vulnerability. The detection outcomes are presented in Table 5.

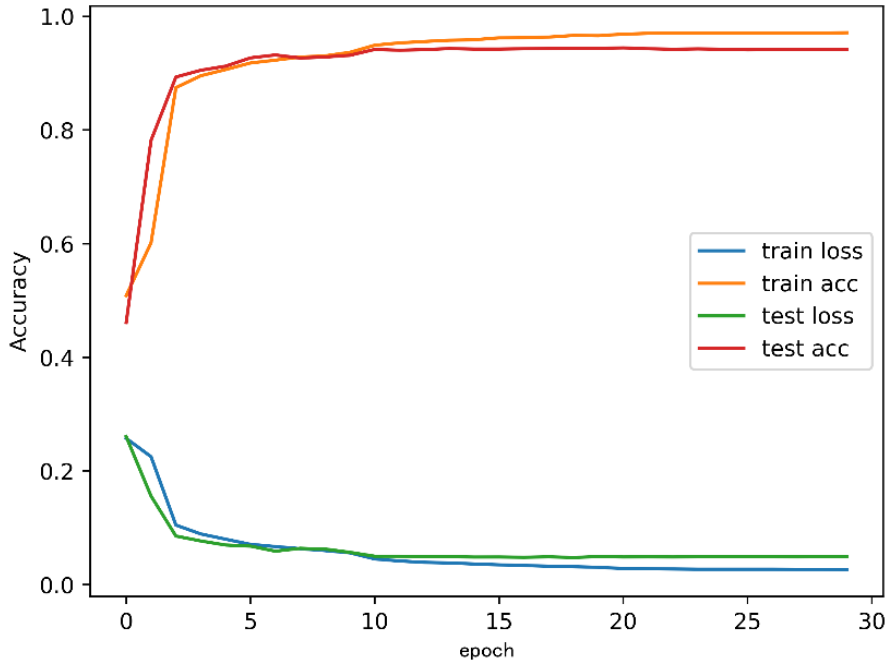

Figure 4: Training process of W2V-SA model

Table 5
Test results

Types of vulnerability	Precision	Recall	F1-score
AssertFail	94.50%	94.40%	94.45%
BlockTimestamp	95.56%	95.47%	95.51%
CheckEffects	94.99%	94.98%	94.98%
LowLevelCalls	94.46%	94.46%	94.46%
Reentrancy	97.71%	97.70%	97.70%
IntegerUnderFlow	91.86%	91.74%	91.80%

Table 5 demonstrates that the W2V-SA model presented in this paper achieves over 90% for the three evaluation metrics, Precision, Recall, and F1-score, for the six aforementioned vulnerabilities. To establish the superiority of the W2V-SA model, this paper compares it with the fastText and CNN models to evaluate the performance of detecting smart contract vulnerabilities under the same experimental environment. The results of the comparison between the W2V-SA model proposed in this paper and fastText and CNN are depicted in Figure 5 to Figure 7.

As shown in Figure 5 and Figure 6, in terms of Precision, the best detection for the six smart contract vulnerability types is the W2V-SA model, with Precision exceeding 90%, reaching 94.85% on average. The next best model is CNN, with an average Precision of 88.34%. The last is fastText, with a Precision average of only 84.99%. In terms of Recall, the best detection effect is the W2V-SA model, with Recall exceeding 90% and the average value reaching 94.79%. The next is CNN, with a recall average of 88.08%. The last is fastText, with a recall average of only 84.99%. This means that the W2V-SA model is more accurate in detecting smart contract vulnerabilities, and is more suitable for smart contract vulnerability detection than fastText and CNN. Among the other smart vulnerability types, the best Reentrancy vulnerability detection is W2V-SA, with Precision and Recall metrics exceeding 97%. Among the models compared, the CNN model performs relatively in the detection of Reentrancy vulnerabilities, with Precision and Recall metrics exceeding 93%, and the worst result is fastText, with Precision and Recall metrics not exceeding 90%. The performance of CNN with fastText in the detection task of Reentrancy vulnerabilities is still lower than that of the W2V-SA model.

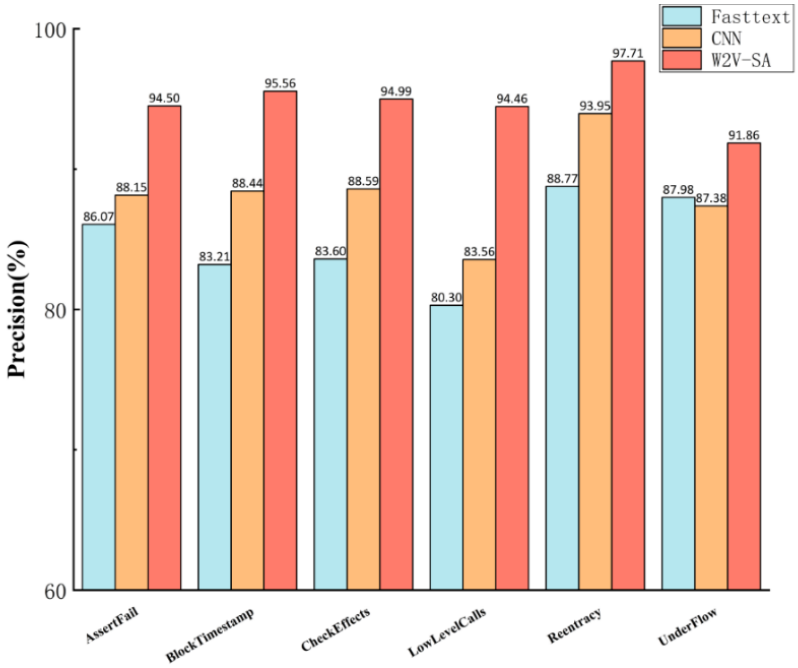

Figure 5: Comparison results of Precision for different models

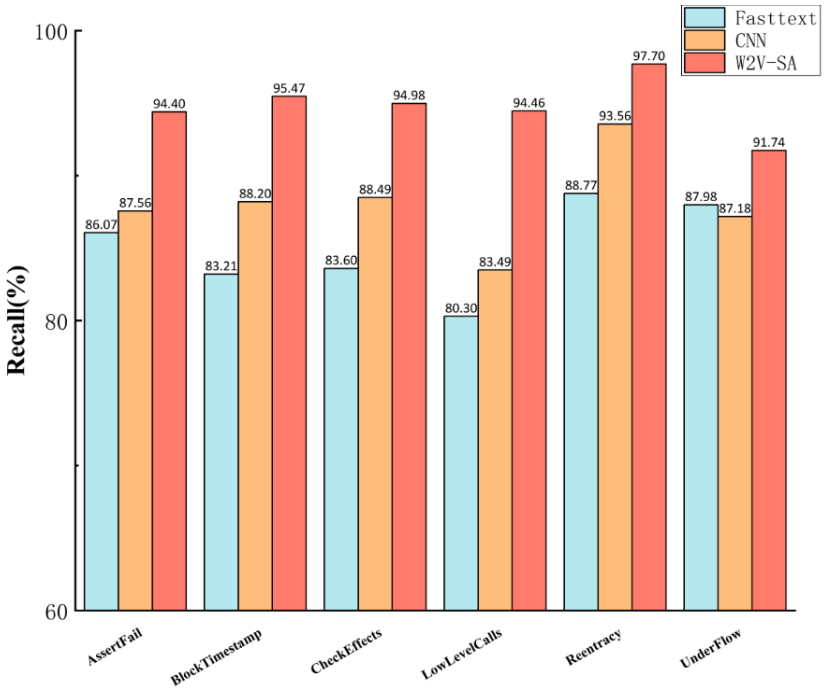

Figure 6: Comparison results of Recall for different models

The F1-score is a metric utilized to assess the comprehensive performance of the model, and a higher F1-score value indicates a superior model. As illustrated in Figure 7, the average F1-score values of W2V-SA exceed 94%, followed by CNN with an average F1-score of over 88.21%, while fastText has the lowest average F1-score value of only 84.99%.When compared to CNN and fastText, both of which are single-model deep neural network models and algorithms, the W2V-SA model proposed in this paper combines the advantages of several deep neural network models. The comparison results demonstrate that the average value of all metrics for the W2V-SA model exceeds that of the CNN model by more than 6%, and that of the fastText model by more than 10%, under the same experimental environment. Among all the models involved in the comparison, the F1-score of the W2V-SA model is higher, indicating the effectiveness of the vulnerability detection method used in the model. In terms of performance metrics, the Recall and Precision of the CNN and fastText models are inferior to those of

the W2V-SA model. As a result, the detection outcomes of the W2V-SA model can identify more samples containing smart contract vulnerabilities.

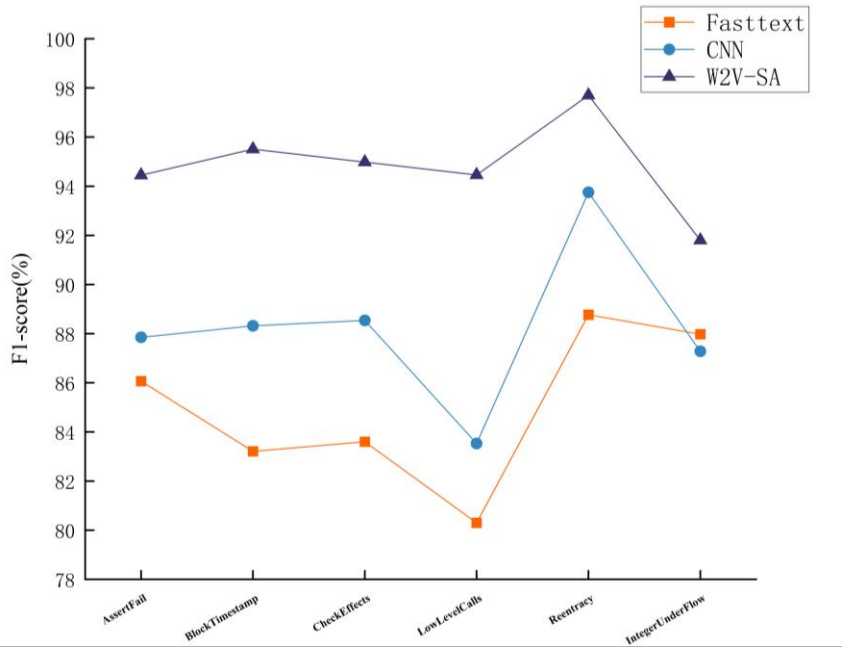

Figure 7: F1-score comparison

The results suggest that the W2V-SA model proposed in this paper is highly effective in detecting smart contract vulnerabilities, with all metrics surpassing 90%. Additionally, it outperforms single-model neural network models and algorithms with an exceptional level of performance.To summarize, in contrast to previous smart contract vulnerability detection models, the W2V-SA model can update the detection model based on different types of vulnerabilities, maintaining a high accuracy rate. Nevertheless, there are limitations to the smart contract vulnerability detection model proposed in this paper. Firstly, the W2V-SA model can only identify whether a smart contract contains a certain type of vulnerability and cannot detect if a smart contract contains multiple types of vulnerabilities. Secondly, the Precision of the W2V-SA model is higher for detecting smart contract vulnerabilities with clear features, whereas it is lower for detecting vulnerabilities with obscure features. For example, the Precision of the W2V-SA model in detecting integer underflow vulnerabilities is 91.86%, whereas in detecting reentrant vulnerabilities, it is 97.71%.

5. Conclusions

The W2V-SA model proposed in this paper is a deep neural network-based smart contract vulnerability detection model that amalgamates the advantages of various deep neural network models. By meticulously considering contextual information during vulnerability detection, this model accurately and rapidly extracts features. To evaluate the performance of the W2V-SA model, we conducted comparative tests with other smart contract vulnerability detection methods under the same experimental environment. The results showed that the W2V-SA model possesses excellent smart contract vulnerability detection capabilities and is more efficient than other models in detecting common smart contract vulnerabilities. Furthermore, this paper demonstrated the effectiveness of the self-attention mechanism in the model through comparative experiments.

In future work, we will focus on studying vector representation methods and feature extraction methods for text, improving model structures, and increasing the efficiency of smart contract vulnerability detection. The next work will delve into two aspects: (1) Using sentence vectors or other word embedding models to improve the model structure. (2) Combining different attention mechanisms to improve our approach. In order to cope with the constantly updated smart contract vulnerabilities

and to face the serious harm caused by smart contract vulnerabilities, we need to explore more efficient methods for detecting smart contract vulnerabilities.

6. Acknowledgments

This work is supported by the Key-Area Research and Development Program of Guangdong Province 2020B1111420002, the Key-Area Research and Development Program of Hubei Province 2022BAA040, the Science and Technology Project of Department of Transport of Hubei Province 2022-11-4-3, and the Innovation Fund of Hubei University of Technology BSQD2019027、BSQD2019020 and BSQD2016019. We deeply appreciate your consideration of our manuscript, and we look forward to receiving comments from the reviewers.

7. References

[1] N. Szabo, Smart contracts: building blocks for digital markets, EXTROPY: The Journal of Transhumanist Thought 18 (1996) 28.

[2] V. Piantadosi, G. Rosa, D. Placella, S. Scalabrino, R. Oliveto, Detecting functional and security-related issues in smart contracts: A systematic literature review, Software: Practice and Experience 53 (2023) 465-495. doi:10.1002/spe.3156.

[3] M. Pournader, Y. Shi, S. Seuring, S. C. L. Koh, Blockchain applications in supply chains, transport and logistics: a systematic review of the literature, International Journal of Production Research 58 (2020) 2063-2081. doi: 10.1080/00207543.2019.1650976.

[4] Z. Xu, S. Zhang, H. Han, X. Dong, Z. Zheng, H. Wang, W. Tian, Blockchain-aided searchable encryption-based two-way attribute access control research, Security and Communication Networks 2022 (2022) 2410455. doi:10.1155/2022/2410455.

[5] M. Barboni, A. Morichetta, A. Polini, Smart contract testing: challenges and opportunities, in: Proceedings of the 2022 IEEE/ACM 5th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2022, pp. 21-24. doi:10.1145/3528226.3528370.

[6] S. M. Beillahi, E. Keilty, K. Nelaturu, A. Veneris, F. Long, Automated auditing of price gouging TOD vulnerabilities in smart contracts, in: Proceedings of the 2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2022, pp. 1-6. doi:10.1109/ICBC54727.2022.9805509.

[7] E. M. Sifra, Security vulnerabilities and countermeasures of smart contracts: a survey, in: Proceedings of the 2022 IEEE International Conference on Blockchain (Blockchain), 2022, pp. 512-515. doi: 10.1109/Blockchain55522.2022.00080.

[8] S. Kushwaha, S. Joshi, D. Singh, M. Kaur, H-N. Lee, Ethereum smart contract analysis tools: a systematic review, IEEE Access 10 (2022) 57037-57062. doi:10.1109/ACCESS.2022.3169902.

[9] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, H-N. Lee, Systematic review of security vulnerabilities in ethereum blockchain smart contract, IEEE Access, 10 (2022) 6605-6621. doi: 10.1109/ACCESS.2021.3140091.

[10] R. Pise, S. Patil, A deep dive into blockchain-based smart contract-specific security vulnerabilities, in: Proceedings of the 2022 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS), 2022, pp. 1-6. doi:10.1109/ICBDS53701.2022.9935949.

[11] F. Sparbrodt, M. García-Valls, Digesting smart contracts in Ethereum blockchain networks, in: Proceedings of the 2022 5th Conference on Cloud and Internet of Things (CIoT), 2022, pp. 60-66. doi: 10.1109/CIoT53061.2022.9766685.

[12] Z. Liu, P. Qian, X. Wang, L. Zhu, Q. He, S. Ji, Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion, arXiv preprint arXiv:2106 09282 2021. doi:10.48550/arXiv.2106.09282. doi: arxiv-2106.09282.

[13] J. Feist, G. Grieco, A. Groce, Slither: a static analysis framework for smart contracts, in: Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2019, pp. 8-15. doi:10.1109/WETSEB.2019.00008.

[14] w. Wang, J. Song, G. Xu, Y. Li, H. Wang, C. Su, Contractward: automated vulnerability detection models for ethereum smart contracts, IEEE Transactions on Network Science and Engineering 8 (2020) 1133-1144. doi:10.1109/TNSE.2020.2968505.
[15] Piantadosi V, Rosa G, Placella D, Scalabrino S, Oliveto R. Detecting functional and security-related issues in smart contracts: A systematic literature review. Software: Practice and Experience 2023, 53(2): 465-495. doi: 10.1002/spe.3156
[16] L. Luu, D-H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security; 2016. pp. 254-269. doi:10.1145/2976749.2978309.
[17] C. F. Torres, A. K. Iannillo, A. Gervais, R. State, ConFuzzius: a data dependency-aware hybrid fuzzer for smart contracts, in: Proceedings of the 2021 IEEE European Symposium on Security and Privacy (EuroS&P), 2021, pp. 103-119. doi:10.1109/EuroSP51992.2021.00018.
[18] M. Mossberg, F. Manzano, E. Hennenfent, A. Groce, G. Grieco, J. Feist, T. Brunson, A. Dinaburg, Manticore: a user-friendly symbolic execution framework for binaries and smart contracts, in: Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1186-1189. doi:10.1109/ASE.2019.00133.
[19] M. Ding, P. Li, S. Li, H. Zhang, Hfcontractfuzzer: fuzzing hyperledger fabric smart contracts for vulnerability detection, Evaluation and Assessment in Software Engineering, 2021, pp 321-328. doi:10.48550/arXiv.2106.11210.
[20] Ivanov N, Li C, Sun Z, Cao Z, Luo X, Yan Q. Security Threat Mitigation For Smart Contracts: A Survey. arXiv preprint arXiv:2302 07347 2023. doi: 10.48550/arXiv.2302.07347
[21] W. Li, F. Qi, M. Tang, Z. Yu, Bidirectional LSTM with self-attention mechanism and multi-channel features for sentiment classification, Neurocomputing 387 (2020) 63-77. doi: 10.1016/j.neucom.2020.01.006.
[22] G. Jin, Z. Yu, A Korean named entity recognition method using Bi-LSTM-CRF and masked self-attention, Computer Speech & Language 65 (2021) 101134. doi: 10.1016/j.csl.2020.101134.
[23] G. Xu, D. Zhou, J. Liu, Social network spam detection based on ALBERT and combination of Bi-LSTM with self-attention, Security and Communication Networks 2021 (2021) 5567991. doi: 10.1155/2021/5567991.
[24] Cong S, Zhou Y. A review of convolutional neural network architectures and their optimizations. Artificial Intelligence Review 2023, 56(3): 1905-1969. doi: 10.1007/s10462-022-10213-5
[25] M.-T. Fang, K. Przystupa, Z.-J. Chen, et al, Examination of abnormal behavior detection based on improved YOLOv3, Electronics, 10 (2021) 197. doi: 10.3390/electronics10020197.
[26] Pan X, Ge C, Lu R, Song S, Chen G, Huang Z, Huang G. On the integration of self-attention and convolution. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition; 2022. pp. 815-825. doi: 10.48550/arXiv.2111.14556
[27] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification, in: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, 2017, pp. 427–431. doi: 10.48550/arXiv.1607.01759
[28] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP): Association for Computational Linguistics, 2014, pp. 1746-1751. doi: 10.3115/v1/D14-1181.
[29] G. Wood, Others. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151 (2014) 1-32.
[30] H. He, Y. Bai, E. A. Garcia, S. Li, ADASYN: adaptive synthetic sampling approach for imbalanced learning, in: Proceedings of the 2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence), 2008, pp. 1322-1328. doi: 10.1109/IJCNN.2008.4633969.