



Article

Smart Contract Vulnerability Detection Model Based on Siamese Network (SCVSN): A Case Study of Reentrancy Vulnerability

Ran Guo ¹, Weijie Chen ², Lejun Zhang ^{2,3,4,*} , Guopeng Wang ^{4,*} and Huiling Chen ⁵ ¹ School of Physics and Materials Science, Guangzhou University, Guangzhou 510006, China² College of Information Engineering, Yangzhou University, Yangzhou 225127, China³ The Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510006, China⁴ Engineering Research Center of Integration and Application of Digital Learning Technology, Ministry of Education, Beijing 100039, China⁵ Department of Computer Science and Artificial Intelligence, Wenzhou University, Wenzhou 325035, China

* Correspondence: zhanglejun@gzhu.edu.cn (L.Z.); wangguopeng@sohu.com (G.W.)

Abstract: Blockchain technology is currently evolving rapidly, and smart contracts are the hallmark of the second generation of blockchains. Currently, smart contracts are gradually being used in power system networks to build a decentralized energy system. Security is very important to power systems and attacks launched against smart contract vulnerabilities occur frequently, seriously affecting the development of the smart contract ecosystem. Current smart contract vulnerability detection tools suffer from low correct rates and high false positive rates, which cannot meet current needs. Therefore, we propose a smart contract vulnerability detection system based on the Siamese network in this paper. We improved the original Siamese network model to perform smart contract vulnerability detection by comparing the similarity of two sub networks with the same structure and shared parameters. We also demonstrate, through extensive experiments, that the model has better vulnerability detection performance and lower false alarm rate compared with previous research results.

Keywords: smart contract; deep learning; siamese network



Citation: Guo, R.; Chen, W.; Zhang, L.; Wang, G.; Chen, H. Smart Contract Vulnerability Detection Model Based on Siamese Network (SCVSN): A Case Study of Reentrancy Vulnerability. *Energies* **2022**, *15*, 9642. <https://doi.org/10.3390/en15249642>

Academic Editors: João Martins and Keke Gai

Received: 13 November 2022

Accepted: 16 December 2022

Published: 19 December 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Smart contracts, a computer protocol designed to disseminate, validate or enforce contracts informally, allowing trusted transactions without a third party, were proposed by cryptographer Nick Szabo in the 1990s [1], but they were not used or developed due to the lack of a trusted execution environment at the time. Blockchain has the characteristics of openness, transparency, decentralization and invariance. So, blockchain can be used as a vehicle for smart contracts [2]. The structure of a smart contract is shown in Figure 1. Compared to traditional contracts, smart contracts are participants and executors of the contract, so the process of contract execution does not require third party involvement.

Due to their convenience, smart contracts are beginning to be widely used in power systems and Internet of Things (IoT). For example, using a blockchain-based big data management approach to integrate smart grids with the Internet of Things (IoT) [3]. Ethereum [4] is a global, decentralized platform for financial and new applications; essentially a public chain that supports smart contracts. There are many smart contracts deployed on Ethereum, and these smart contracts control billions of dollars' worth of Ether. In 2016, hackers exploited the reentrancy vulnerability to attack The Dao, resulting in the loss of over 60 million USD [5]. Security incidents like this can create a serious trust crisis in the blockchain, so we need to build an efficient and smart contract vulnerability detection tool. In this paper, we propose a smart contract vulnerability detection model based on Siamese networks named

SCVSN model, which performs vulnerability detection by calculating similarity, and has the advantages of simple structure, high correct rate and low false positive rate. Through experiments, we demonstrate that our proposed SCVSN model has better performance in vulnerability detection compared with previous smart contract vulnerability detection tools. The main contributions of this paper are described as follows:

1. Applying the natural language processing side to smart contract processing, and proposing a smart contract embedding method that reduces the impact of irrelevant content on smart contract vulnerability detection.
2. This is the first model that applies the Siamese network to smart contract vulnerability detection at the source code level. SCVSN improves the accuracy of smart contract vulnerability detection and reduces the false positive rate of smart contract vulnerability detection by calculating the similarities. To ensure that the number of positive and negative sample pairs is balanced during training, we also propose a classification algorithm for positive and negative sample pairs.
3. The SCVSN model proposed in this paper is demonstrated to have excellent performance in smart contract vulnerability detection tasks through extensive experiments on the existing smart contract dataset, and the SCVSN model is compared with the deep learning-based smart contract vulnerability detection methods proposed in previous studies.

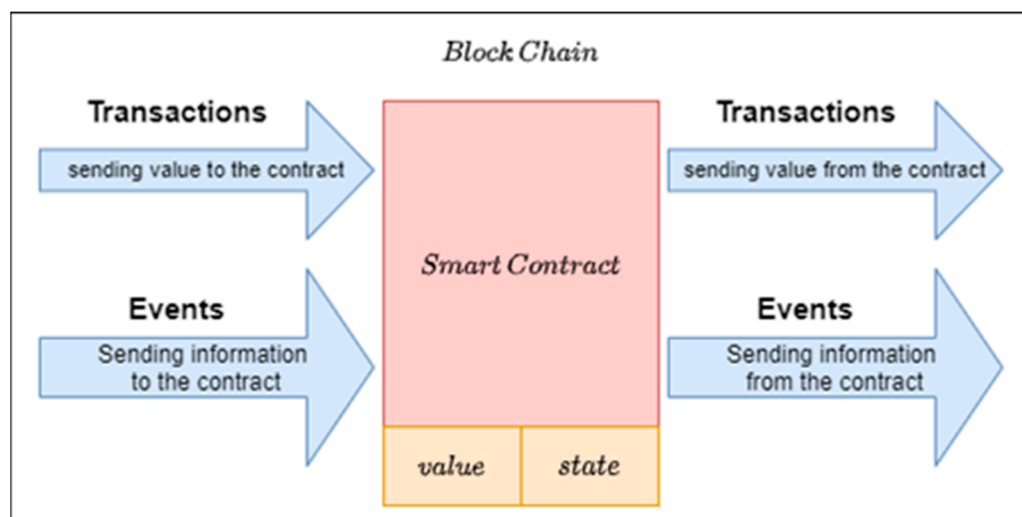


Figure 1. Smart contract structure.

The paper is organized as follows: We present current research on smart contract vulnerability detection, Siamese network and our motivation in Section 2. We discuss our methodology for our study in Section 3. We present the experimental procedure which includes dataset processing and performs performance comparisons in Section 4. Finally, we conclude the whole paper and future work in Section 5.

2. Related Work

2.1. Blockchain and Power Systems

With the development of blockchain technology and power systems, blockchain technology is gradually being applied to power systems, using smart contracts to control centralized power transactions, and using blockchain technology to establish a secure electronic information trading platform [6]. For example, Ferrag MA et al. propose DeepCoin, a blockchain-based energy framework for enabling peer-to-peer energy transactions [7]. Therefore, it is important to ensure the security of the power system network. For example, Dehghani M et al. applied deep learning techniques with wavelet singular values approach to DC microgrid network attack detection [8]. Ghiasi M et al. proposed to apply

blockchain technology and Hilbert Huang Transform to grid network attack detection and enhance grid security, where Hilbert Huang Transform is applied to FDI attack detection and blockchain technology is used for data exchange [9]. Liang G et al. [10] propose a new blockchain-based distributed protection framework to enhance the self-defense of power systems against cyber attacks, and experimentally demonstrate the effectiveness of the proposed framework.

From the above study, it can be seen that blockchain technology is well referenced in the power system, and it is also important to ensure the security of the power system network, but due to the tamper-evident nature of the smart contracts on the blockchain, the security of the power system network cannot be guaranteed when there are loopholes in the smart contracts, so it is also important to ensure the security of the smart contracts used.

2.2. Smart Contract Vulnerability Detection

With the development of deep learning, deep learning is starting to be applied to smart contract vulnerability detection. Yu X et al. developed a framework named DeeSCV Hunter [11], proposing the concept of vulnerability candidate slice (VCS), and this tool mainly targets the detection of reentrancy vulnerabilities and timestamp vulnerabilities. Wu H [12] et al. proposed a Wu method to improve the correctness of smart contract vulnerability detection by capturing critical data flow information in smart contracts. Mi F et al. proposed the VSCL framework to automatically detect vulnerabilities in smart contracts on the blockchain [13], they utilize novel feature vector generation techniques from bytecode of smart contract. Qian P [14] et al. combined the Bidirectional long short-Term memory (BiLSTM) neural network with attention mechanism for the detection of smart contract reentrancy vulnerability and obtained better accuracy. Xing C [15] et al. proposed slicing matrix, a new method to extract vulnerability features. Zhuang et al. [16] have proposed a Degree-Free Graph Convolutional Neural network (DR-GCN) and a Temporal Message Propagation network (TMP).

The above researches shows that deep learning can be applied to smart contract vulnerability detection and can achieve excellent results. Bromley

2.3. Siamese Network

The Siamese network was first introduced in the 1990s [17]. The current application of Siamese networks is also mainly to calculate similarity. The Siamese network has been used in medical, target, tracking, and code similarity comparison due to excellent performance. Amin-Naji M [18] et al. apply the Siamese network in the study of Alzheimer's disease. They used convolutional neural networks as sub-networks of the Siamese network to determine whether a patient has Alzheimer's disease by calculating the similarity of structural MRI to the Normal Control and the Alzheimer's Disease. Li Z et al. [19] combined a Siamese network with LSTM neural network for detecting audio signal similarity detection. Wu Y et al. [20] combined convolutional neural networks with Siamese network and applied them to code similarity computation and achieved good results. Wang Q et al. [21] Combining twin networks with cross-contrast neural networks for identifying the hepatocellular carcinoma (HCC) and the intra-hepatic cholangiocarcinoma (ICC). the experimental results showed that Siamese network has excellent feature extraction ability. Eryong Tang et al. [22] combined a Siamese network with CNN and added attention mechanism in feature extraction process, and their proposed model was used for target tracking and showed significant improvement in tracking success and tracking accuracy. Visual et al. [23] combined the attention mechanism with Siamese networks, and they proposed a method to calculate attention weights to improve the matching ability through a fractional network of Siamese networks, which solves the search problem by locating the attention part.

The above research shows that the Siamese network has an excellent performance in computing similarity, so the Siamese networks are chosen for similarity computation in this paper.

2.4. Research Motivation

In the above researches, the process of detecting vulnerabilities in smart contracts based on deep learning is similar. The detection process is shown in Figure 2. The general training process of the model is to first process the original data, then perform data labelling, extract the feature through the neural network model and then perform classification, and finally produce the results. When a neural network model performs classification, the accuracy of the result is related to the result of feature extraction. The more adequate the feature extraction, the more accurate the final classification result will be, but this will inevitably lead to an increase in the complexity of the neural network, and such a network requires a huge amount of computation both in training and in use. And in the previous researches, the general approach to reduce the false positive rate of the model is to process the dataset in advance and extract the key parts of the data before training to reduce the influence of irrelevant parts on the model when extracting features, and VCS proposed by Yu X [11] et al. is based on this concept. However, the main cause of smart contract vulnerabilities is incorrect external calls between functions, and vulnerabilities may appear anywhere in the smart contract, so performing critical part extraction before training may lead to inadequate model feature extraction and thus reduce the accuracy of vulnerability detection.

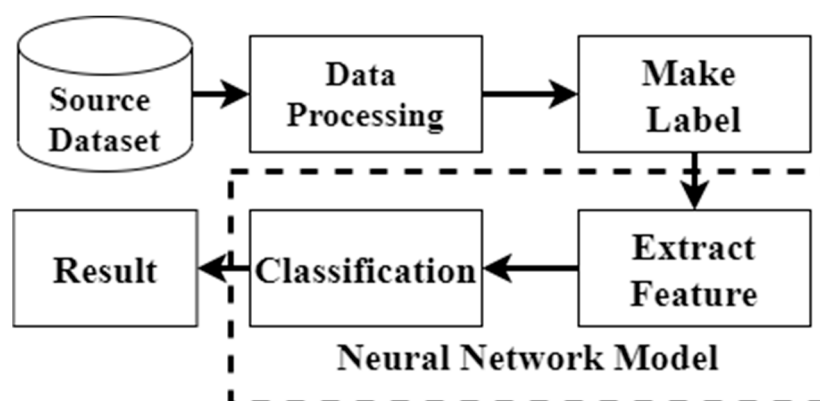


Figure 2. General vulnerability detection process.

So, we propose a new neural network model SCVSN for detecting smart contract vulnerability by combining Siamese networks for similarity calculation with a deep learning model. Our proposed SCVSN model uses only few neural network layers to achieve a fast vulnerability detection by similarity calculation. When feature extraction is performed, the parameters are updated between the subnetworks according to the different situations of the two smart contracts in pairs, which ensures the model feature extraction is adequate and reduces the data processing time before training.

In addition, smart contracts currently have multiple vulnerabilities. Among these vulnerabilities the detection of reentrancy vulnerability has a high rate of false positive, this is because the reason for the occurrence of the reentrancy vulnerability is that the use of the fallback function to call the function itself leads to the formation of a recursive call, at the same time, a transfer operation is performed during the recursive call, resulting in a circular transfer. Although there is a judgment statement in the code, the state update statement is after the function call, so the state update will not be executed because of the loop call. Figure 3 shows a real-world instance of a smart contract reentrancy attack. When the attacker calls the attack function in contract Contract2 (line 6), this function will execute the withdraw function in contract Contract1 (line 7). When contract Contract2 executes the withdraw function (line 3), it will use a call statement to send Ether to contract Contract2 (line 6). As mentioned above, when an externally owned account or other contract sends Ether to the contract address, the fallback function of the called contract will be executed. So, the fallback function in contract Contract2 is executed (line 9) and the fallback function

of contract Contract2 calls the withdraw function of contract Contract1 to get Ether again. Using this method, contract Contract2 will continuously get Ether from contract Contract1. Through reentrancy vulnerability, all the money in the “victim’s” contract account will eventually be withdrawn, but it will only be recorded once in the block. From the above cases, we can see that the reentrancy vulnerability is very harmful. However, existing vulnerability detection tools rely too much on hard rules defined by experts, which cannot give correct detection results while having a high rate of false positives. Therefore, it is urgent and important to solve the problem of smart contract reentrancy vulnerability. So, we choose the reentrancy vulnerability as a case study for this research.

Contract1(Victim)	
1	Contract Contract1{
2	mapping (address => unit) private balances;
3	function withdraw (unit _amount) {
4	if (balance [msg.sender] >= _amount) {
5	msg.sender. call.value (_amount) ();
6	balance [msg.sender] -= _amount;
7	}
8	}
9	function () public payable ()
10	}

Contract2(Attack)	
1	Contract Contract2{
2	Contract1 public entrance;
3	constructor (address _target) public {
4	entrance = entrance Contract1 (_target);
5	}
6	function attack () payable {
7	entrance. withdraw (0.5 ether);
8	}
9	function () public payable {
10	entrance. withdraw (0.5 ether);
11	}
12	}

Figure 3. A real-world instance of smart contract reentrancy attack.

3. Siamese Network Structure in SCVSN Model

From the perspective of model structure, the SCVSN model has the following main steps: (1) The first step is to input the sol files. (2) The second step is to process the sol files and use word2vec for word embedding. (3) The third step SCVSN model performs feature extraction on the word vectors, calculates the Euclidean distance, and obtains the final result by comparing it with the boundary value (m). The architecture of the SCVSN model is shown in Figure 4. The first and third steps in the SCVSN model structure are discussed in detail in the Section 3.1 of the paper. The second step, since it is not related to the siamese network structure, is described in detail in Section 4.1 in the experimental section.

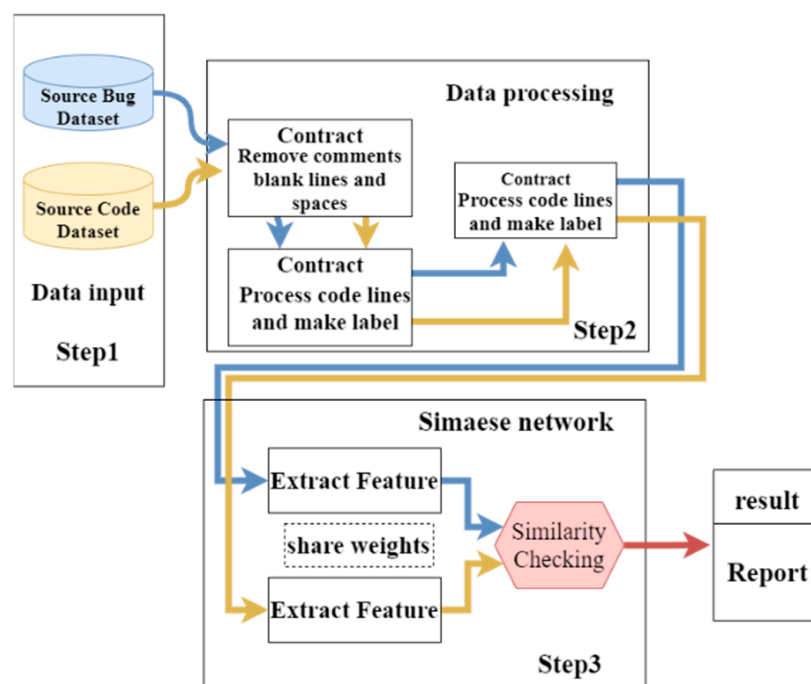


Figure 4. SCVSN model structure.

We have summarized and explained the symbols that will appear in the following, as shown in Table 1.

Table 1. The symbol table.

Symbol	Explain
C	sample smart contracts
X	the feature values
d	the distance between two smart contracts
D	the database used for training models
l	The label of dataset
a	a smart contract
b	a smart contract
y	the label of weather two smart contract samples match
m	the threshold of the distance between two samples

3.1. Siamese Network Structure in SCVSN Model

In the third step of our proposed model, feature extraction using the SCVSN model is required to calculate the Euclidean distance d . Then the Euclidean distance d is compared with the edge value margin to finally derive the result. The Siamese network structure used in our study is the combination of the Siamese network and LSTM neural network and the structure of the Siamese network used in this paper is shown in Figure 5. It can be seen that two subnetworks have the same structure and the weights are shared between the two subnetworks. The subnetworks in the Siamese network used in SCVSN model consisted of only the input layer, LSTM neural network layer, Dropout layer, ReLU layer and fully connected layer, the structure of the subnetwork is simple. The input data goes through the LSTM neural network layer to extract the features and then passes through the Dense layer, which nonlinearized the previously extracted features and extracts the association between them, and then passes through the Dropout layer to avoid overfitting, the data are then processed using the nonlinear ReLU activation function. For large-scale data, ReLU has a better fitting ability compared to tanh activation function and sigmoid activation

function, and also better enhances the nonlinearity of the model, making the neural network more discriminative. The data goes through the Dropout layer and ReLU layer once more to further increase the stability of the model, followed by the vector transformation through the Dense layer, and finally the feature extraction results are output. After the two subnetwork models extract the feature values, the Siamese network determines whether the two smart contracts are similar by calculating the Euclidean distance between the output values of the two subnetworks. The processing of the Siamese network structure used in this paper can be represented by the following equation.

$$\text{word2vec}(C_1) \Rightarrow C_1 \quad (1)$$

$$\text{word2vec}(C_2) \Rightarrow C_2 \quad (2)$$

$$\text{LSTM}(C_1) \Rightarrow X_1 \quad (3)$$

$$\text{LSTM}(C_2) \Rightarrow X_2 \quad (4)$$

$$|\text{Distance}(X_1, X_2)| \Rightarrow d \quad (5)$$

where C_1 and C_2 denote sample smart contracts, C_1 and C_2 denote the input matrices obtained after word embedding, X_1 and X_2 denote the feature values obtained after processing by the LSTM network in the Siamese network, d denotes the distance between two smart contracts.

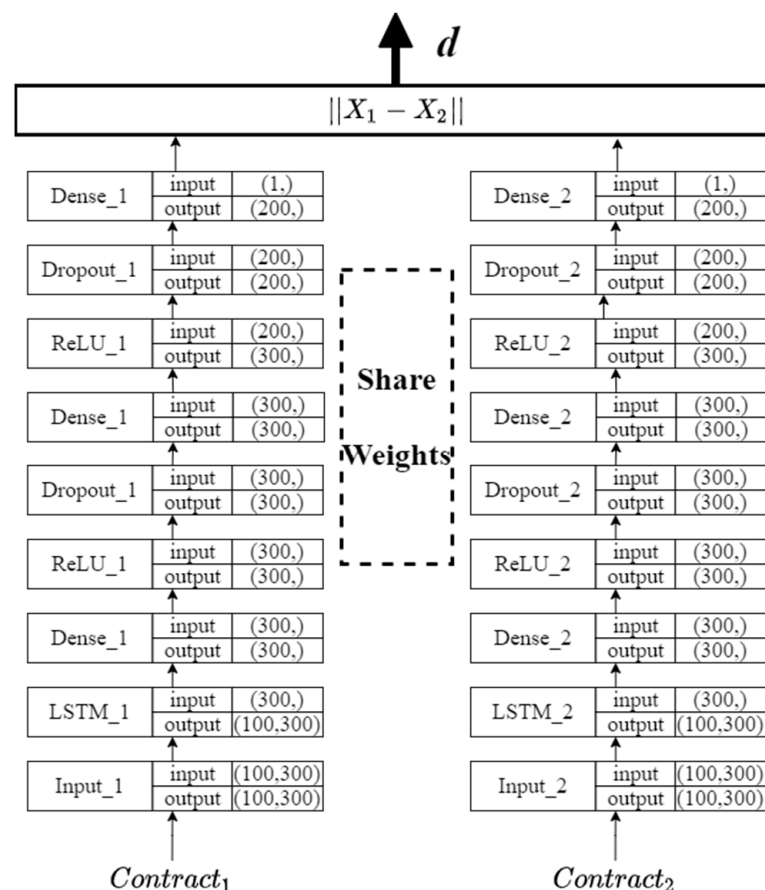


Figure 5. Siamese network structure.

LSTM can make full use of temporal relationship and semantic information among data. The benefits of using LSTM for our subnetwork are that LSTM is able to establish connections between units in a loop and remember previous inputs through their internal states, LSTM neural networks have specialized gate control structures capable of controlling

the memory and forgetting of data [24], and LSTM neural networks do not suffer from gradient disappearance and gradient explosion. So, the LSTM neural network is used as the subnetwork of the Siamese network structure in the research.

3.2. The Core Idea of SCVSN Model

Our proposed approach uses Siamese network model to compute the similarity of two smart contracts, and based on comparing the similarity, thus performing vulnerability detection of smart contracts. The core idea of our proposed model can be briefly described as two subnetworks of SCVSN model, one of which extracts the feature of a smart contract A and other extracts the feature of a smart contract B. Smart contract A contains the vulnerability and is used as a reference sample, smart contract B may or may not contain vulnerability. First of all, the features extracted from the two subnetworks are subjected to Euclidean distance calculation, and then the Euclidean distance is compared with the threshold value. If the value of Euclidean distance is lower than the threshold, it means that the two smart contracts are similar and smart contract B contains vulnerability, if the Euclidean distance is higher than the threshold, it means that the two contracts are not similar and thus it is judged that smart contract B does not contain the vulnerability. In order to implement the idea, we proposed, the following issues need to be addressed:

1. How to implement smart contract vulnerability detection via SCVSN model.
2. How to train SCVSN model to improve their feature extraction capabilities.
3. How to reduce rate of false positives.

First question. Our proposed SCVSN model performs features extraction through subnetworks, After the training of the SCVSN model is completed (the training process described in detail in second question), the subnetwork that specifically extracts features of the smart contract containing the vulnerability saves the trained parameters. In testing or practical use, the sample to be tested is passed through another sub-network of SCVSN model to extract features, then the Euclidean distance between the two feature value is calculated, followed by a comparison of the Euclidean distance with a threshold value to determine whether the sample to be tested contains vulnerabilities. The structure of our proposed SCVSN model is simple, using only 2 LSTM neural network layers and 4 ReLU layers, but achieves good detection results and small computational effort. After the network is trained, the features of the reference samples (smart contracts containing vulnerabilities) can be extracted in advance, and the samples to be tested only need to extract their features and then calculate the Euclidean distance and compare it with the threshold value. The vulnerability detection process is shown in Figure 6.

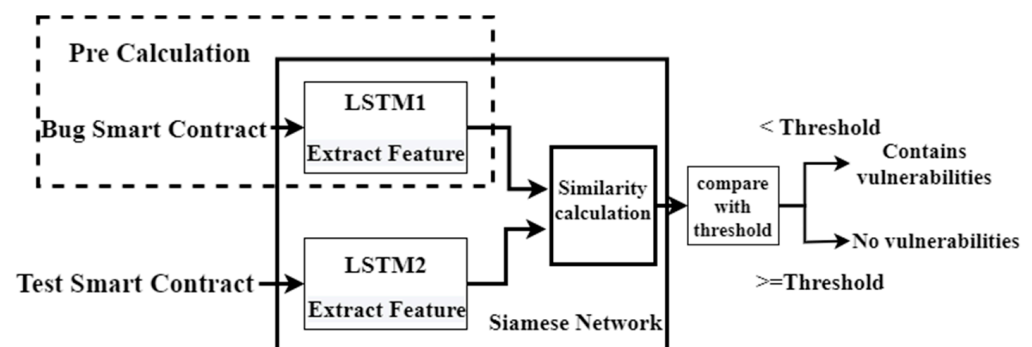


Figure 6. Vulnerability detection process.

Second question. Our approach is to first use the 50 smart contract samples that have been processed in the Source Bug Dataset as the reference samples, since these 50 smart contract samples are certain to contain vulnerabilities, then 50 smart contract reference samples containing vulnerabilities are combined with 2918 smart contract samples already processed in Source Code Dataset to form positive and negative sample pairs respectively.

The positive sample is composed of a sample of smart contracts containing vulnerabilities in the Source Bug Dataset and a sample of smart contracts containing vulnerabilities in the Source Code Dataset, and this sample pair corresponds to a label of 1, negative sample is a sample pair consisting of a sample of smart contracts with vulnerabilities in the Source Bug Dataset and a sample of smart contracts without vulnerabilities in the Source Code Dataset, and this sample pair corresponds to a label of 0. After constructing a large number of positive and negative samples, the samples are input into the network model, which is used to train the network model and improve the ability of the SCVSN model to extract features. The algorithm for forming positive and negative sample pairs is shown in Algorithm 1.

Algorithm 1. Positive and negative sample classification

Input:

D_1 : bug smart contract dataset;
 D_2 : source smart contract dataset;
 $label_1$: label of D_1 ;
 $label_2$: label of D_2 ;

Output:

Pairs: dataset of sample pairs consisting of positive and negative samples in a specific order

Pairs_label: label of the dataset

```

1: int  $l_1 = 0$ ;
2: int  $l_2 = 0$ ;
3: for  $l_1 = 0; l_1 < D_1.length; l_1++$  do
4:   for  $l_2 = 0; l_2 < D_2.length; l_2++$  do
5:     Pairs.append( $D_1[l_1], D_2[l_2]$ )
6:     if  $label_1[l_1] == label_2[l_2]$  then
7:       Pairs_label.append(1)
8:     else
9:       Pairs_label.append(0)
10:    end if
11:  end for
12: end for
13: return Pairs, Pairs_label;
```

Third question. Since smart contract vulnerability can appear anywhere in a smart contract, the fact that two smart contracts are similar does not necessarily mean that the tested smart contract contains the smart contract vulnerabilities. As shown in Figure 4, where Contract1 is the one containing reentrancy vulnerability. When the attacker uses reentrancy vulnerability to attack, Contract1 will not execute according to the logic intended by the contract creator, where the vulnerability appears in line 12. Contract2 does not contain reentrancy vulnerability. These two smart contracts are similar, but one contains reentrancy vulnerability and the other does not. If Contract1 is used as the reference sample, the result must be that the two smart contracts are similar by the usual method of calculating similarity, thus proving that Contract2 contains reentrancy vulnerability, but this is not the actual case. By training the SCVSN model, the false positive rate of smart contract vulnerabilities in this case can be reduced. The previous subsection illustrates that SCVSN model uses two subnetworks together to extract features, and the parameters are shared between the subnetworks, providing a stronger feature extraction capability than a single network, so the loss function used in SCVSN model needs to be able to handle the relationship among sample pair labels, Euclidean distance and threshold. Therefore, the loss function used in our proposed SCVSN model is Contrastive Loss which shown in Equation (6)

$$loss = \frac{1}{2N} \sum_{n=1}^N yd^2 + (1 - y)\max(m - d, 0)^2 \quad (6)$$

In the formula, d denotes the Euclidean distance between the features of two smart contract samples, which can also indicate the degree of similarity between two smart

contract samples, with the expression $d = ||a_n - b_n||^2$, where a and b denote two different smart contracts. y indicates the label of whether two smart contract samples match, $y = 1$ means they are similar, $y = 0$ means they are not similar, and this labelling rule is also used when constructing the labels of positive and negative sample pairs, m indicates threshold. The advantage of our choice to use Contrastive Loss as the loss function is that the overall loss function ensures that sample pairs that are already similar remain similar when SCVSN model performs feature extraction, while sample pairs that are not similar remain dissimilar after feature extraction. When the sample pair label $y = 1$, the loss function is shown in Equation (7).

$$loss = \frac{1}{2N} \sum_{n=1}^N y d^2 \quad (7)$$

The sample pair label $y = 1$ indicates that both smart contracts contain vulnerabilities, and if the Euclidean distance between the two smart sample features is too large, it indicates that SCVSN model is not good at extracting critical information and needs to increase the loss. When the sample pair label $y = 0$, the loss function is shown in Equation (8).

$$loss = \sum (1 - y) \max(\text{margin} - d, 0)^2 \quad (8)$$

The sample pair label $y = 0$ indicates that one of the two smart contract samples is not containing the smart contract vulnerability. If the Euclidean distance between the two smart contract sample pairs is too small, the same indicates that SCVSN model is not good at extracting critical information and needs to increase the loss. If the Euclidean distance between sample pairs is too large over the threshold margin, it is a normal sample pair, but it does not help the model's ability to extract features, so we choose to ignore these sample pairs, which can greatly reduce the amount of computation and increase the training speed. As shown in Figure 6, one subnetwork LSTM1 in our proposed SCVSN model extracts only the feature values of smart contracts containing vulnerabilities in the source bug dataset, and the other subnetwork LSTM2 extracts the feature values of smart contracts in the source code dataset, and the weights are shared between the two subnetworks, the parameters of LSTM1 are retained when the training is completed. Since smart contract vulnerability can appear anywhere in a smart contract, it leads to a high rate of false positives in smart contract vulnerability detection. In this paper, the proposed SCVSN model is trained with one of the subnetworks learning the features of the smart contract containing the vulnerability and the other subnetwork learning the information of the location where the smart contract vulnerability appears in the training sample. Since the weights are shared, it can be understood that LSTM2 helps LSTM1 to optimize the parameters during training.

As the case in Figure 7, the red colored font is the part where the smart contract vulnerability occurred. When training, if SCVSN model predicts the wrong result which different from the corresponding result, The SCVSN model will update the model parameters according to the Contrastive Loss for learning. The parameter update of SCVSN reduces the influence of the features of false positive samples similar to Contract2 extracted by LSTM2 on the model parameters. We trained SCVSN by constructing a large number of positive and negative sample pairs, by training sample pairs of different cases, SCVSN becomes increasingly accurate in extracting vulnerability features and reduces the impact of false positive cases on the model update parameters.

Algorithm 2. training process of the SCVSN model**Input:**

Pairs: The positive and negative sample pairs constructed by Algorithm 1;

Pairs_label: The labels corresponding to the positive and negative sample pairs;

1: for $(C_1, C_2, label)$ in $(Paris, Paris_label)$ **do**2: $C_1^w = \text{word2vec}(C_1)$ 3: $C_2^w = \text{word2vec}(C_2)$ 4: $X_1 = \text{LSTM}_1(C_1^w)$ 5: $X_2 = \text{LSTM}_2(C_2^w)$ 6: $d = |\text{Distance}(X_1, X_2)|$

7: if label = 0 then

8: if $d > m$ then

9: continue

10: else

11: $\text{loss}(\theta) = \text{Equation (7)}$

12: end if

13: else

14: $\text{loss}(\theta) = \text{Equation (8)}$

15: end if

16: Compute gradient: $\nabla(\theta)$ 17: Update model: $\theta = \theta - \omega(\theta)$

18: end for

Contract1

```

1 contract EtherStore {
2   uint256 public withdrawalLimit = 1 ether;
3   mapping(address => uint256) public lastWithdrawTime;
4   mapping(address => uint256) public balances;
5   function depositFunds() public payable {
6     balances[msg.sender] += msg.value;
7   }
8   function withdrawFunds (uint256 _weiToWithdraw) public {
9     require(balances[msg.sender] >= _weiToWithdraw);
10    require(_weiToWithdraw <= withdrawalLimit);
11    require(now >= lastWithdrawTime[msg.sender] + 1 weeks);
12    require(msg.sender.call.value(_weiToWithdraw));
13    balances[msg.sender] -= _weiToWithdraw;
14    lastWithdrawTime[msg.sender] = now;
15  }
16 }

```

Contract2

```

1 contract EtherStore {
2   uint256 public withdrawalLimit = 1 ether;
3   mapping(address => uint256) public lastWithdrawTime;
4   mapping(address => uint256) public balances;
5   function depositFunds() public payable {
6     balances[msg.sender] += msg.value;
7   }
8   function withdrawFunds (uint256 _weiToWithdraw) public {
9     require(balances[msg.sender] >= _weiToWithdraw);
10    require(_weiToWithdraw <= withdrawalLimit);
11    require(now >= lastWithdrawTime[msg.sender] + 1 weeks);
12    balances[msg.sender] -= _weiToWithdraw;
13    lastWithdrawTime[msg.sender] = now;
14    msg.sender.transfer(_weiToWithdraw);
15  }
16 }

```

Figure 7. Similar smart contracts.

4. Experiment

4.1. Dataset Processing

In the research, SCVSN model uses two datasets, the first one is a recently released large-scale dataset named Smart Bugs Dataset-wild [12]. It contains 47,398 real and unique sol files, which together have about 203,716 smart contracts in total, this is because a sol file may contain more than one smart contract. The Source Code Dataset used for training is extracted from this dataset. In addition, the SCVSN model used another small dataset SolidiFI Benchmark [25], which contains 7 different vulnerabilities, each vulnerability type contains 50-sol files, and this dataset serves as Source Bug Dataset.

We detect reentrancy vulnerabilities in the types of smart contract vulnerabilities, so we need to classify SmartBugs Dataset-wild, from which we find sol files containing reentrancy vulnerabilities in smart contracts, although each sol file contains multiple smart contracts, but smart contracts can call each other, and reentrancy vulnerability are caused by calling external contracts, so as long as there is a reentrancy vulnerability in a sol file, then the entire sol file is marked as containing reentrancy vulnerability. With such a large dataset, not only is the time spent if we just rely on manual labelling huge, but the error rate also increases with time, so we refer to Mi F et al. Multiple tools are used for labelling in the research, mainly because the results labelled by a single tool are not completely correct. Through the research of Durieux T et al. [26], they analyzed the effectiveness of nine smart contract analysis tools in smart contract vulnerability detection, and finally chose three tools, Oyente [27], Mythril [28] and SmartCheck [29]. We have selected three vulnerability detection tools to label smart contracts. SmartCheck is a static analysis tool. SmartCheck runs lexical and syntactical analysis on Solidity source code. However, Oyente cannot provide full coverage of known vulnerabilities, and Oyente can only detect for a few vulnerabilities. Mythril is also a symbolic execution tool. The main idea of Mythril is to use symbolic execution to search for all possible unsafe paths. Mythril can efficiently find vulnerabilities, but it cannot find business logic problems in a contract, and takes a long time to detect. In our research, in order to quickly filter out the sol files that contain the reentrancy vulnerability, we use three tools we chosen to detect smart contract sol files in SmartBugs Dataset-wild simultaneously, and when the detection time of one tool exceeds 120 s, we abandon the current sol file and proceed to the next one. We analyzed the SmartBugs dataset-wild and finally filtered out 1459-sol files containing smart contract vulnerabilities. When classifying, a whole sol file is considered as a smart contract. The rule used for the classification of sol files is shown in Figure 8. After the data has been classified, the data still needs to be processed before it can be formed into the Source Code Dataset that SCVSN model needs.

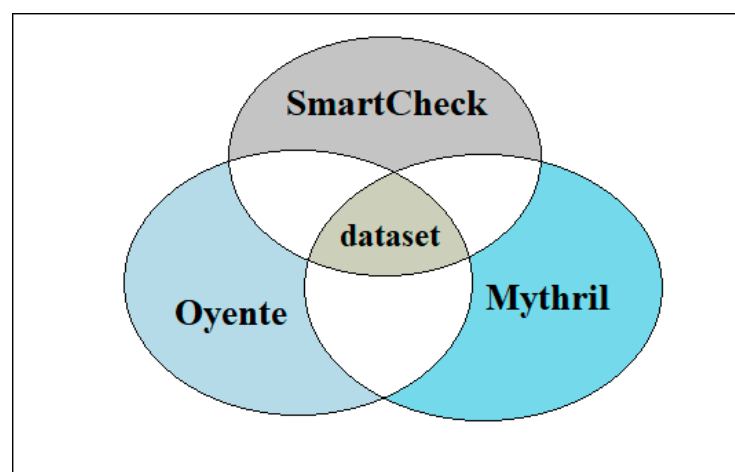


Figure 8. Dataset contains 2918 smart contract.

Smart contracts are written in a solidity language and consist of multiple lines of code but not all of them are related to reentrancy vulnerability detection. Firstly, removes the comments from the smart contract, which do not have any relation to the analysis of the smart contract content, then we collected the list of solidity keywords from the official Ethereum website, following by replacing the names of user-defined variables in the smart contract with symbolic names and replacing the names of user-defined functions in the smart contract with symbolic names. The purpose of this is to reduce the impact of user-defined variable names and function names on smart contract vulnerability detection, and finally Each line of the processed smart contract is treated as a fragment. The specific operation steps are shown in Figure 9.

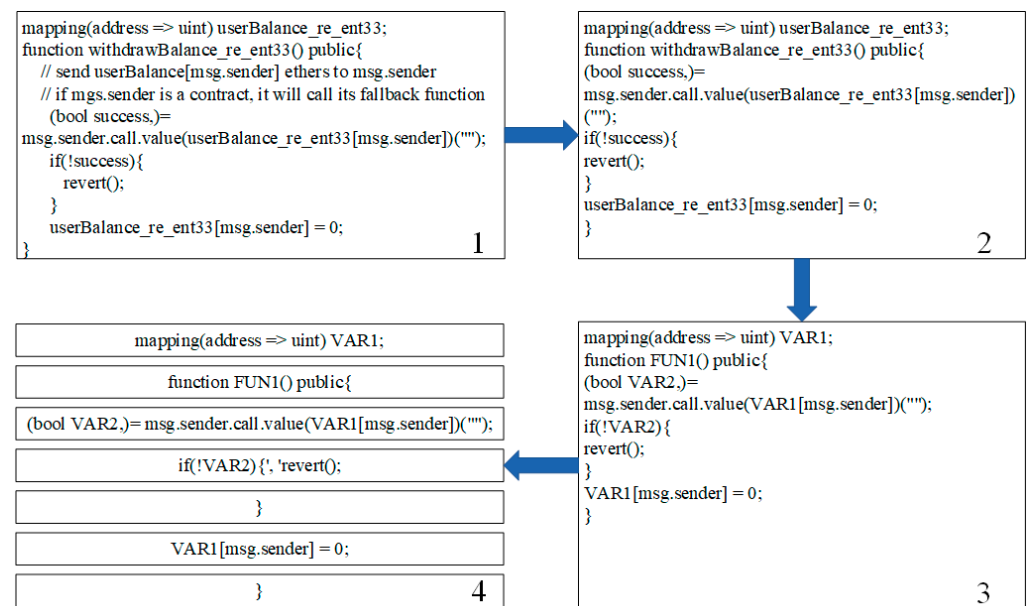


Figure 9. Smart Contract Processing One.

In summary, there are three steps to collect smart contract fragments. (1) Removing non-ASCII characters, comments and blank lines from smart contracts. (2) Replace custom variable names in smart contracts with VAR and custom function names in smart contracts with FUN. (3) Represent each line of the processed smart contract as a smart contract fragment. After collecting the smart contract fragments, each of them is subdivided into a series of tokens, as shown in Figure 10.

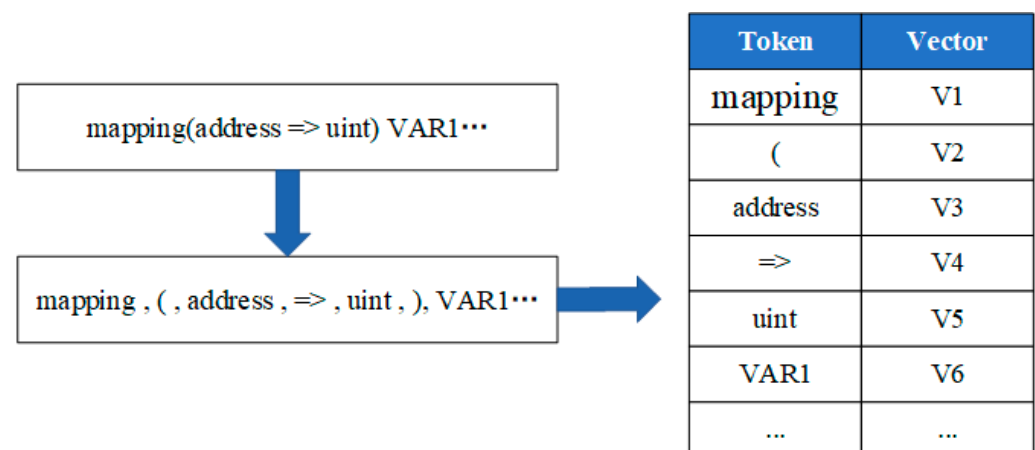


Figure 10. Smart Contract Processing Two.

Each line in the vector will consist of a series of tokens, which contain keywords, operation symbols, built in normalization variables and symbols. The code token's state also contains information from other code tokens that preceded it, which capture semantic and control flow dependencies. Then input the collected large number of tokens into the word2vec [30] model for feature extraction to get the vector and complete the construction of the dataset.

Word2vec captures word-to-word relationships between contexts in a multidimensional space and has been widely used as a preliminary step in predictive modeling and information retrieval tasks. The above is the process of processing the dataset in the research, firstly converting the code file into code fragments, then converting the code fragments into a collection of tokens, then performing feature extraction by word2vec, and finally forming the dataset used in training. The correct handling of datasets can be of great help in vulnerability detection of smart contracts.

4.2. Experimental Procedure

During the experiments, the evaluation criteria for model performance is the accuracy of the model on the test set. We recorded the training data for each epoch and calculated the average of the accuracy and loss values based on the recorded data, as shown in Figure 11.

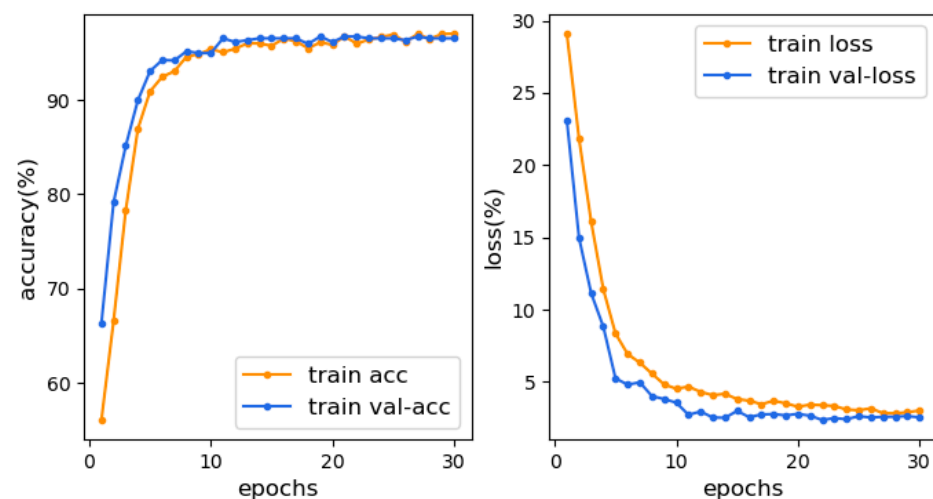


Figure 11. Training results.

The plot on the right shows the relationship between epoch and loss function, and the plot on the left shows the relationship between epoch and accuracy. In this paper, the data of each epoch is recorded during the training to facilitate the observation and optimization of the model. From the change of the curve, it can be seen that the loss value and the correct rate of the model on both the training and test sets are changing rapidly in the first 10 epochs. After 10 epochs, the loss values and accuracy rates of the model on the training and test sets fluctuate up and down regularly, and the overall trend tends to level off, indicating that the model starts to converge gradually at this point. In the right plot, it can be seen that the curve of loss declines in the test set and training set is gradually easing and there is no violent jitter, indicating that the value of batch size in the experiment is correct. The loss curves of the training and test sets crossed during the training process, and the distance between the two curves was moderate, without overlap or excessive distance, indicating that there was no overfitting or under-fitting in the training process.

In order to correctly represent the data of the model, we generated a graph of the data from multiple training of the model, and the specific results are shown in Figure 12. The straight line in the figure indicates the average accuracy calculated from the test results after several training times. The results of 40 times of the model on the test set are recorded in Figure 12, with the worst accuracy of 94.35% and the best accuracy of 97.14%.

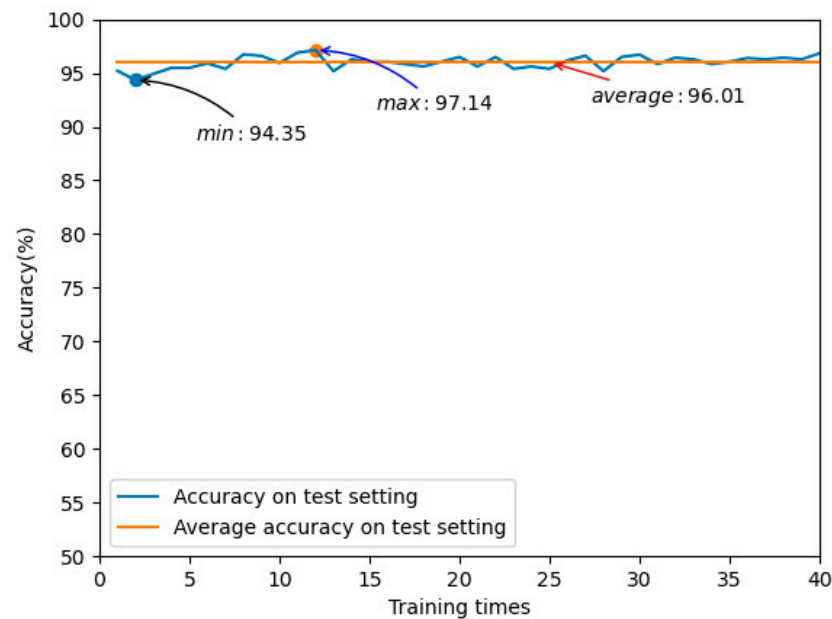


Figure 12. Multi-training results.

As can be seen in Figure 12, the results of the first 30 tests of the SCVSN model fluctuate around the average value, and the results of the last 10 tests of the SCVSN model were almost the same as the average value. From the test results, we can see that the difference between the results of each test of the SCVSN model is very small, which can illustrate the stability of the SCVSN model structure. By averaging the 40 results, the average correct rate was calculated to be 96.01%, and it can also be seen that the results of all 40 tests fluctuate around 96.01%. So, we concluded that our proposed SCVSN model has a correct detection rate of 96.01% for smart contract reentrancy vulnerability. To demonstrate the superiority of the Siamese network model we used, we chose the LSTM model, which has the same structure as the subnetwork model in the SCVSN model, for self-comparison experiments. The average results of LSTM are shown in Figure 13.

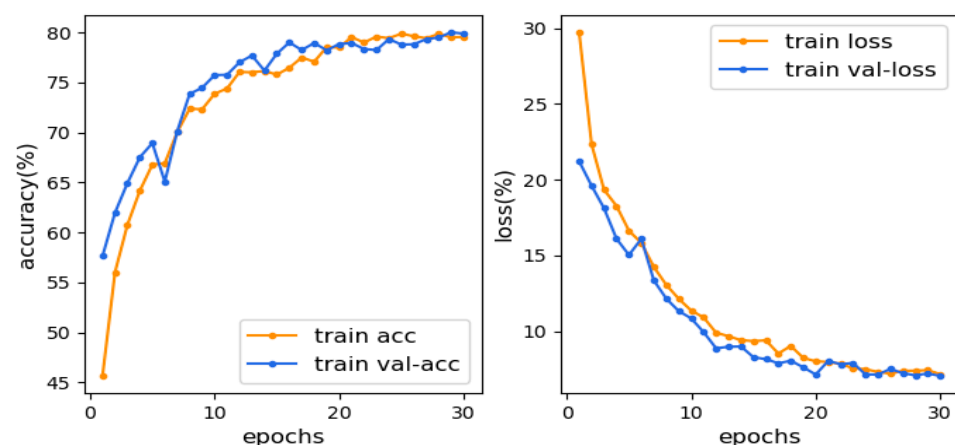


Figure 13. LSTM training results.

It can be seen from the Figure 13 that the traditional LSTM model starts to converge when the epoch is 20 during the training period. In contrast, the SCVSN model combined with the Siamese network learning model starts to converge at epoch 10, and the curve is relatively smooth. From the comparison, it can be seen that SCVSN has good performance in terms of accuracy and convergence speed of the model. We also tested the main hyperparameters of the model, and the test results are shown in Table 2.

Table 2. Model parameter testing.

Units	Lr	ACC (%)	T(s)
250	0.001	94.51	70.28
300	0.001	96.01	75.70
250	0.001	96.00	83.94
300	0.002	95.99	75.44
300	0.003	95.89	74.54

Where ACC is the accuracy on the test set, T is the training time, units is the hidden layer of the model, and Lr is the learning rate of the model. We tested the model with units of 250,300,350 respectively. From the test results, we can see that ACC does not change when the units exceed 300, but T becomes longer, and the ACC decreases slightly when the units are below 300. The learning rate of the model was tested with Lr of 0.001, 0.002, 0.003 respectively. From the test results, it can be seen that the model has the highest ACC when Lr is 0.001, and from the comparison results, it can also be seen that the change of Lr has little effect on the overall model. It shows that the SCVSN model proposed in this paper has parameter stability.

4.3. Experimental Performance Comparison

The performance metrics are those introduced above, including ACC, REC, FPR and F1. In the same experimental setting, the DeeSCV Hunter, the Peculiar, the VSCL, the BLSTM-ATT, and the TMP were selected for comparison. SCVSN data is the average of multiple test results. The results of the comparison are shown in the Table 2. The measurements we chose were accuracy (ACC), precision (PRE), F1-score (F1), and recall rate (REC). They can be calculated by the following formulas:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

$$TPR = \frac{TP}{TP + FP} \quad (10)$$

$$Recall = \frac{TP}{TP + FN} \quad (11)$$

$$FPR = \frac{FP}{FP + TN} \quad (12)$$

$$F1 = \frac{2 * PRE * TPR}{PRE + TPR} \quad (13)$$

TP denotes positive samples predicted by the model as positive class, TN denotes negative samples predicted by the model as negative class, FP denotes negative samples predicted by the model as positive class, and FN denotes positive samples predicted by the model as negative class.

Firstly, it can be seen from Table 3 that our proposed SCVSN model has excellent performance in smart contract vulnerability detection. The SCVSN model has relatively stable values in all four-performance metrics, whereas the SCVSN model has a correct rate of 96.01%, which is better than the models involved in the comparison. The recall of the SCVSN model proposed in this paper is 96.64%, which exceeds the rest of the models, this shows that the SCVSN approach to detecting smart contract vulnerabilities can reduce the false positive rate in vulnerability detection.

Table 3. Comparison of experimental results.

Model	ACC (%)	REC (%)	PRE (%)	F1 (%)
SCVSN	96.01	96.04	94.25	94.78
DeeSCVHunter [14]	93.02	83.46	90.70	86.87
VSCL [13]	95	95	99	95
Peculiar [12]	92.37	92.4	91.80	92.10
BLSTM-ATT [15]	88.47	88.48	88.5	88.26
TMP [16]	84.48	82.63	74.06	83.82
DR-GCN [16]	81.47	80.89	72.36	76.39
LSTM	81.91	91.43	76.80	83.48

Secondly, among the other models involved in the comparison, the VSCL model has the best vulnerability detection capability, followed by the DeeSCVHunter model. We discuss the results in terms of both accuracy and F1-score. The SCVSN model is higher than other models in terms of accuracy, and since the number of positive and negative samples in the dataset used in this paper is the same, the higher the correctness indicates the superior performance of the model. The F1-score of VSCL is slightly higher than that of SCVSN, and when the value of F1 is higher, it indicates that the experimental method is effective. The VSCL model is processing the smart contract byte code during pre-processing, and the SCVSN model processes the smart contract code. VSCL preprocesses the smart contract into a new sequence of operations by applying a control flow graph (CFG) and a depth-first search approach. The new sequence of operations is able to better reflect the execution speech in the smart contract code. However, when building CFG, VSCL will encounter the problem of difficulty in identifying JUMPDEST, and some of the semantic information will be lost when solving this problem. The SCVSN removes parts of the smart contract that are not relevant to vulnerability detection in order to improve the accuracy rate. Both methods aim to remove parts of the smart contract that are not relevant to vulnerability detection. The processing based on control flow graph and depth-first search is able to remove information from smart contracts that is not relevant to vulnerability extraction, but this has the disadvantage that the pre-processing time increases and the complexity of the processing is too high. After pre-processing is complete, VSCL uses a single network for feature extraction and classification, and the SCVSN model uses two branch subnetworks for feature extraction and classification by calculating the similarity. The SCVSN models can learn for vulnerabilities that appear in different locations during training. So, in the test results, it can be seen that the SCVSN model has a lower false positive rate than the VSCL model and a higher correct rate than the VSCL model, and SCVSN is able to detect for a variety of smart contract vulnerabilities. The DeeSCVHunter model uses vulnerability candidate slicing (VSC) in the pre-processing phase of smart contracts to extract VCS through control dependencies and data dependencies, making VSC contain more vulnerability syntax and semantic features. The semantic information can be better extracted by constructing the VSC method, but by this way the constant calls related to the transaction amount need to be defined first, and VCS can currently only detect smart contract timestamp vulnerabilities and smart contract reentry vulnerabilities. The pre-processing method used in this paper is able to fully preserve the semantic information of the smart contract, but the semantic information will contain information related to multiple vulnerabilities. The DeeSCVHunter model also uses a single deep learning model when performing feature extraction. In the discussion of the VSCL model and the DeeSCVHunter model it can be seen that the SCVSN model is able to maintain a high correct rate and low false positive rate in smart contract vulnerability detection tasks compared to the model with a single network structure, the pre-processing method used in this paper can guarantee the full semantic information of the smart contract, but the pre-processed smart contract will contain information about multiple smart contract vulnerabilities.

5. Conclusions

We first proposed the idea of combining smart contract vulnerability detection with Siamese networks and performed smart contract vulnerability detection by comparing similarities. Our proposed SCVSN model has an excellent performance in smart contract vulnerability detection due to the simple structure of Siamese networks, powerful similarity computation capability and feature extraction ability. In the work of this paper, we apply a natural language processing approach to the processing of smart contracts for two realistic datasets and apply the processed dataset to the training and testing of the model. We demonstrate, through extensive experiments, that our proposed SCVSN model can achieve good results in smart contract reentrancy vulnerability detection tasks. Our research also provides a new way of thinking about vulnerability detection, specifically by calculating the similarity to detect code vulnerabilities, and our code will be open-sourced afterwards.

Our proposed SCVSN model still has shortcomings in the types of vulnerability detection. The DeeSCV Hunter model can detect reentrancy vulnerability and timestamp dependence vulnerability. Our subsequent work is to detect more smart contract vulnerabilities, such as integer overflow vulnerability, timestamp vulnerability. Our future idea is to expand the Smart Bug Dataset to classify smart contract files containing vulnerabilities, extract features for multiple classification, and to consider the case of multiple vulnerabilities in one smart contract.

Author Contributions: Conceptualization, W.C. and L.Z.; methodology, W.C.; software, W.C.; validation, W.C., L.Z. and H.C.; formal analysis, R.G.; investigation, Guopeng Wang; resources, L.Z.; data curation, G.W.; writing original draft preparation, W.C.; writing—review and editing, W.C.; visualization, W.C.; supervision, R.G. and G.W.; project administration, L.Z.; funding acquisition, L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China No. 62172353 and No. U20B2046. Future Network Scientific Research Fund Project No. FNSRFP-2021-YB-48. Innovation Fund Program of the Engineering Research Center for Integration and Application of Digital Learning Technology of Ministry of Education No.1221045.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The experimental data and associated code used in this study have been deposited in the GitHub repository (<https://github.com/xiaoaochen/SCVSN>) accessed on 15 December 2022.

Acknowledgments: The authors would like to thank the reviewers for their detailed reviews and constructive comments, which have helped improve the quality of this paper.

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

References

1. Giancaspro, M. Is a ‘smart contract’ really a smart idea? Insights from a legal perspective. *Comput. Law Secur. Rev.* **2017**, *33*, 825–835. [CrossRef]
2. Khan, S.N.; Loukil, F.; Ghedira-Guegan, C.; Benkhelifa, E.; Bani-Hani, A. Blockchain smart contracts: Applications, challenges, and future trends. *Peer Peer Netw. Appl.* **2021**, *14*, 2901–2925. [CrossRef] [PubMed]
3. Ghiasi, M.; Wang, Z.; Mehrandezh, M.; Jalilian, S.; Ghadimi, N. Evolution of smart grids towards the Internet of energy: Concept and essential components for deep decarbonisation. *IET Smart Grid* **2022**, 1–17. [CrossRef]
4. Oliva, G.A.; Hassan, A.E.; Jiang, Z.M. An exploratory study of smart contracts in the Ethereum blockchain platform. *Empir. Softw. Eng.* **2020**, *25*, 1864–1904. [CrossRef]
5. Samreen, N.F.; Alalfi, M.H. Reentrancy vulnerability identification in ethereum smart contracts. In Proceedings of the 2020 IEEE 3rd International Workshop on Blockchain Oriented Software Engineering, London, ON, Canada, 18 February 2020; IEEE: Piscataway, NJ, USA, 2020.
6. Ghiasi, M.; Niknam, T.; Wang, Z.; Mehrandezh, M.; Dehghani, M.; Ghadimi, N. A comprehensive review of cyber-attacks and defense mechanisms for improving security in smart grid energy systems: Past, present and future. *Electr. Power Syst. Res.* **2023**, *215*, 108975. [CrossRef]

7. Ferrag, M.A.; Maglaras, L. DeepCoin: A Novel Deep Learning and Blockchain-Based Energy Exchange Framework for Smart Grids. *IEEE Trans. Eng. Manag.* **2020**, *67*, 1285–1297. [\[CrossRef\]](#)
8. Dehghani, M.; Niknam, T.; Ghiasi, M.; Bayati, N.; Savaghebi, M. Cyber-Attack Detection in DC Microgrids Based on Deep Machine Learning and Wavelet Singular Values Approach. *Electronics* **2021**, *10*, 1914. [\[CrossRef\]](#)
9. Ghiasi, M.; Dehghani, M.; Niknam, T.; Kavousi-Fard, A.; Siano, P.; Alhelou, H.H. Cyber-Attack Detection and Cyber-Security Enhancement in Smart DC-Microgrid Based on Blockchain Technology and Hilbert Huang Transform. *IEEE Access* **2021**, *9*, 29429–29440. [\[CrossRef\]](#)
10. Liang, G.; Weller, S.R.; Luo, F.; Zhao, J.; Dong, Z.Y. Distributed Blockchain-Based Data Protection Framework for Modern Power Systems Against Cyber Attacks. *IEEE Trans. Smart Grid* **2019**, *10*, 3162–3173. [\[CrossRef\]](#)
11. Yu, X.; Zhao, H.; Hou, B.; Ying, Z.; Wu, B. DeeSCVHunter A Deep Learning-Based Framework for Smart Contract Vulnerability Detection. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; IEEE: Piscataway, NJ, USA, 2021.
12. Wu, H.; Zhang, Z.; Wang, S.; Lei, Y.; Lin, B.; Qin, Y.; Zhang, H.; Mao, X. Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques. In Proceedings of the 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), Wuhan, China, 25–28 October 2021; IEEE: Piscataway, NJ, USA, 2021.
13. Mi, F.; Wang, Z.; Zhao, C.; Guo, J.; Ahmed, F.; Khan, L. VSCL: Automating Vulnerability Detection in Smart Contracts with Deep Learning. In Proceedings of the 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Sydney, Australia, 3–6 May 2021; IEEE: Piscataway, NJ, USA.
14. Qian, P.; Liu, Z.; He, Q.; Zimmermann, R.; Wang, X. Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models. *IEEE Access* **2020**, *8*, 19685–19695. [\[CrossRef\]](#)
15. Xing, C.; Chen, Z.; Chen, L.; Guo, X.; Zheng, Z.; Li, J. A new scheme of vulnerability analysis in smart contract with machine learning. *Wirel. Netw.* **2020**, 1–10. [\[CrossRef\]](#)
16. Zhuang, Y.; Liu, Z.; Qian, P.; Liu, Q.; Wang, X.; He, Q. Smart Contract Vulnerability Detection using Graph Neural Network. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20), Yokohama, Japan, 11–17 July 2020; pp. 3283–3290.
17. Bromley, J.; Bentz, J.W.; Bottou, L.; Guyon, I.; Lecun, Y.; Moore, C.; Säckinger, E.; Shah, R. Signature verification using a “Siamese” time delay neural network. In *Advances in Neural Information Processing Systems*; Series in Machine Perception and Artificial Intelligence; World Scientific: Singapore, 1994; pp. 25–44. ISBN 1793-0839.
18. Amin-Naji, M.; Mahdavinataj, H.; Aghagholzadeh, A. Alzheimer’s disease diagnosis from structural MRI using Siamese convolutional neural network. In Proceedings of the 2019 4th International Conference on Pattern Recognition and Image Analysis (IPRIA), Tehran, Iran, 6–7 March 2019; IEEE: Piscataway, NJ, USA, 2019.
19. Li, Z.; Song, P. Audio similarity detection algorithm based on Siamese LSTM network. In Proceedings of the 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi’an, China, 9–11 April 2021; IEEE: Piscataway, NJ, USA, 2021.
20. Wu, Y.; Wang, W. Code Similarity Detection Based on Siamese Network. In Proceedings of the 2021 IEEE International Conference on Information Communication and Software Engineering (ICICSE), Chengdu, China, 19–21 March 2021; IEEE: Piscataway, NJ, USA, 2021.
21. Wang, Q.; Wang, Z.; Sun, Y.; Zhang, X.; Li, W.; Ge, Y.; Huang, X.; Liu, Y.; Chen, Y. SCCNN: A Diagnosis Method for Hepatocellular Carcinoma and Intrahepatic Cholangiocarcinoma Based on Siamese Cross Contrast Neural Network. *IEEE Access* **2020**, *8*, 85271–85283. [\[CrossRef\]](#)
22. Tang, E.; Wang, Y.; Liu, Y. Siamese Network with Channel-wise Attention and Multi-scale Fusion for Robust Object Tracking. In Proceedings of the 2020 39th Chinese Control Conference (CCC), Shenyang, China, 27–29 July 2020; IEEE: Piscataway, NJ, USA, 2020.
23. Shen, J.; Tang, X.; Dong, X.; Shao, L. Visual Object Tracking by Hierarchical Attention Siamese Network. *IEEE Trans. Cybern.* **2020**, *50*, 3068–3080. [\[CrossRef\]](#) [\[PubMed\]](#)
24. Karevan, Z.; Suykens, J.A.K. Transductive LSTM for time-series prediction: An application to weather forecasting. *Neural Netw.* **2020**, *125*, 1–9. [\[CrossRef\]](#) [\[PubMed\]](#)
25. Ghaleb, A.; Pattabiraman, K. How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA ’20), Online, 18–22 July 2020; Khurshid, S., Păsăreanu, C.S., Eds.; ACM: New York, NY, USA, 2020; pp. 415–427, ISBN 9781450380089.
26. Durieux, T.; Ferreira, J.F.; Abreu, R.; Cruz, P. Empirical review of automated analysis tools on 47,587 Ethereum smart contracts. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; ACM: New York, NY, USA, 2020.
27. Luu, L.; Chu, D.-H.; Olickel, H.; Saxena, P.; Hobor, A. Making Smart Contracts Smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS’16), Vienna Austria, 24–28 October 2016; Weippl, E., Katzenbeisser, S., Kruegel, C., Myers, A., Halevi, S., Eds.; ACM: New York, NY, USA, 2016; pp. 254–269, ISBN 9781450341394.

28. Prechtel, D.; Gros, T.; Muller, T. Evaluating Spread of ‘Gasless Send’ in Ethereum Smart Contracts. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Canary Islands, Spain, 24–26 June 2019; IEEE: Piscataway, NJ, USA, 2019.
29. Tikhomirov, S.; Voskresenskaya, E.; Ivanitskiy, I.; Takhaviev, R.; Marchenko, E.; Alexandrov, Y. SmartCheck. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, Gothenburg Sweden, 27 May 2018; Tonelli, R., Destefanis, G., Counsell, S., Marchesi, M., Eds.; ACM: New York, NY, USA, 2018; pp. 9–16, ISBN 9781450357265.
30. Jatnika, D.; Bijaksana, M.A.; Suryani, A.A. Word2Vec Model Analysis for Semantic Similarities in English Words. *Procedia Comput. Sci.* **2019**, *157*, 160–167. [[CrossRef](#)]