Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.026627

ARTICLE

Vulnerability Detection of Ethereum Smart Contract Based on SolBERT-BiGRU-Attention Hybrid Neural Model

Guangxia Xu[1,*], Lei Liu[2] and Jingnan Dong[3]

[1]Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, 510006, China

[2]School of Software Engineering, Chongqing University of Posts and Telecommunications, Chongqing, 400065, China

[3]School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, 400065, China

*Corresponding Author: Guangxia Xu. Email: xugx@gzhu.edu.cn

Received: 16 September 2022 Accepted: 21 December 2022

ABSTRACT

In recent years, with the great success of pre-trained language models, the pre-trained BERT model has been gradually applied to the field of source code understanding. However, the time cost of training a language model from zero is very high, and how to transfer the pre-trained language model to the field of smart contract vulnerability detection is a hot research direction at present. In this paper, we propose a hybrid model to detect common vulnerabilities in smart contracts based on a lightweight pre-trained language model BERT and connected to a bidirectional gate recurrent unit model. The downstream neural network adopts the bidirectional gate recurrent unit neural network model with a hierarchical attention mechanism to mine more semantic features contained in the source code of smart contracts by using their characteristics. Our experiments show that our proposed hybrid neural network model SolBERT-BiGRU-Attention is fitted by a large number of data samples with smart contract vulnerabilities, and it is found that compared with the existing methods, the accuracy of our model can reach 93.85%, and the Micro-F1 Score is 94.02%.

KEYWORDS

Smart contract; pre-trained language model; deep learning; recurrent neural network; blockchain security

1 Introduction

As an emerging service architecture, blockchain has made great progress in recent years under the extensive promotion of academia and industry, and has been applied in various fields, such as product traceability, the Internet of things and logistics, and digital rights management [1]. The innovation and use of blockchain technology are inseparable from finance, and the realization of complex logic on the chain is executed by the smart contract behind it, so the security of smart contract is an important guarantee for the normal operation of blockchain [2]. The innovation of blockchain benefits from its distributed and decentralized architecture [3]. Because blockchain is distributed and decentralized, data on the chain cannot be tampered with, which is the advantage of blockchain. On the contrary, there are some security challenges. If the smart contract is deployed in the blockchain consensus protocol network for operation, it cannot be modified and will always be executed, which brings

great challenges to the developers of smart contracts. If there is a fatal flaw in the smart contract, the economic loss may be huge and irreversible. There have been many huge losses since smart contract was applied to blockchain. In 2016, The Dao attack, a well-known security incident, was attacked by hackers and stole a large number of Ether coins, with a loss of around 2 million Ether (50 million USD at the time) [4]. In "The Dao" event, a function containing vulnerabilities, exists in the smart contract, and there is a recursive repeated call vulnerability, which leads to the attacker can continuously recursively and unrestricted transfer, and steal the balance of the attacker's account. In 2017, Ethereum Wallet lost more than $154 million in a major security breach caused by multi-signature contracts at Wallet.sol [5]. This breach cost three times as much as The Dao incident. In 2018, the famous BEC overflow attack caused the price of a token to shrink to zero [6]. Hackers took advantage of a smart contract vulnerability to transfer sky-high contract tokens to outside accounts for a short period of time. Using an early Solidity language vulnerability, overflow attacks on variables enable bulk transfers to steal the balance of the target contract account. After several well-known security incidents, more and more security vulnerabilities are exploited by illegal users, resulting in a large number of property losses, and smart contract vulnerability detection gradually attracts extensive attention from the academic community.

In order to efficiently and accurately detect the vulnerabilities contained in smart contracts, this study proposes a bidirectional GRU network model based on SolBERT and an attention mechanism for a large number of smart contracts [7]. Firstly, a pre-trained model called SolBERT is used to represent the source code of the smart contract to solve the problem that the Word2Vec static word embedding model is difficult to represent the source code model. Secondly, the word vector represented in the previous step is input into the downstream sequential network BiGRU, and the hierarchical attention mechanism is integrated to capture the words or sentences with greater weight in the smart contract [8]. The significance of this is that the source code of the smart contract has context semantic dependencies, and the weights of words and sentences are different. Finally, the results are classified through the softmax layer to determine whether there is a certain type of vulnerability in the smart contract. The main contributions of our work are:

- We migrated the pre-trained language model BERT into the smart contract programming language Solidity scene and compressed BERT [9]. We cut the original BERT-Base model from 12 layers to 5 layers, and used weight sharing and knowledge distillation techniques to compress the BERT model with more parameters. It proves that the pre-trained language model can be well used in the smart contract scenario.
- Our proposed downstream smart contract vulnerability detection model is a bidirectional gated neural unit with hierarchical attention mechanism, which is improved on the existing bidirectional LSTM temporal neural network model [10]. It captures more semantic information of different sentences in the source code and provides more semantic features for the neural network model. Our proposed model improves the detection accuracy to 93.85%, the false positive rate is 6.21%.
- Compared with traditional vulnerability detection systems based on software testing methods, our proposed deep learning method can detect more smart contract vulnerabilities, and the detection efficiency is faster, the average detection time for a single smart contract is around 4.5 s.

2 Related Work

According to the statistics of existing research literature in this field, it can be roughly divided into two categories: one is vulnerability detection based on static analysis of programs, and the other is vulnerability detection based on dynamic analysis of programs [11]. Static Analysis includes Formal Verification method, Static Error Point Analysis method, Machine Learning method. Dynamic Analysis includes Symbolic Execution method, Dynamic Error Point Analysis method and Fuzz Testing method [12]. Industry recognition of higher automation vulnerabilities detection tools is often used to dynamic analysis technology, using dynamic analysis technology to judge whether there are vulnerabilities in the smart contract by analyzing the execution data status of the contract and other information according to the execution of the contract operation [13].

2.1 Smart Contract Vulnerability Detection Based on Dynamic Analysis

Smart contract vulnerability detection technology based on the dynamic analysis by constructing a particular input data, the program debugging and tags, tags of interest through the execution of a program process critical data, tracking program flow, according to the result of the program information and operation of runtime intelligent judge the difference of the contract if there is a vulnerability. The main methods are dynamic symbol execution, dynamic taint analysis and fuzzy test. The main idea of dynamic symbolic execution is to verify path reachability through symbolic execution and constraint solution. It aims to form Control Flow Graph (CFG) by simulating execution contract. In the analysis process, symbol value is used to replace any uncertain variable in source code, and the reachability is verified after all paths are collected. Chen et al. proposed DefectChecker contract vulnerability security analysis tool based on symbolic execution; it can reach 88.3% on the precision indicator [14]. By analyzing Ethereum bytecode, it uses three features, such as control flow chart and program stack space event, to detect vulnerabilities. Symbolic execution technology not only obtains more accurate execution paths, but also exposes fatal shortcomings: state space explosion and solving complex and difficult constrained problems [15]. Program dynamic analysis technology usually combines a variety of technologies to improve program coverage and vulnerability detection accuracy. For example, Xie et al. proposed a hybrid testing method based on symbolic execution and fuzzy testing to determine whether a program has vulnerabilities by analyzing the program execution state through the generation of use cases [16]. This method can provide many ideas for smart contract vulnerabilities, and can improve code branch coverage by generating various complex test cases to achieve the purpose of detecting vulnerabilities. And its shortcomings are also obvious; it needs to execute the source code, and the generation of test cases is also relatively difficult. At present, smart contract vulnerability detection automation platform with high recognition in the industry, such as Oyente, Mythril, Securify, Mantico and other smart contract vulnerability detection tools [17–20]. These kinds of automated vulnerability detection tools are well recognized by the industry, and can detect various smart contract vulnerabilities through dynamic analysis and other technologies. However, their disadvantages are that the detection time is very long, they depend on the smart contract solidity language version, and the detection accuracy is low. And the disadvantages such as high false positive rate. Ivanov et al. proposed a contract vulnerability detection tool based on symbolic execution to detect the bytecode at the Ethereum virtual machine level, analyze the bytecode and the global state of Ethereum, generate the contract control flow chart, and output the symbol path with vulnerabilities [21]. This method can analyze the bytecode of the smart contract after execution, but sometimes the execution will cause problems such as memory overflow because the control flow chart is too large. Jiang et al. proposed a framework called ContractFuzzer for smart contract vulnerability detection based on fuzzy testing [22]. By analyzing ABI interface of smart contract, input use cases in accordance with smart contract

syntax were generated, and a series of new test rules were defined to monitor the execution of smart contract by operating EVM. It built a detection model for seven common vulnerabilities, and detected lower false positive rate, more types of vulnerabilities, and lower false positive rate than the current highly recognized automated detection tool Oyente.

2.2 Smart Contract Vulnerability Detection Based on Deep Learning

With the rapid development of deep learning, NLP technology is widely used in image recognition, text processing and other fields. Therefore, a large number of researchers use deep learning method to detect the vulnerability of smart contract [23]. According to a large number of experimental results, the method using deep learning can support more smart contract vulnerabilities than the traditional software-based defect detection method, and the detection time can be greatly reduced, as shown in Table 1 below.

Table 1: Existing vulnerability detection methods

Tools	Types of vulnerabilities supported							
	Integer overflow	Permission verification disappears	Exception handling error	Transaction order dependence	Lack of randomness	Reentrancy vulnerability	Frozen assets	Average times (Sec)
Oyente	×	×	√	√	√	√	×	30
Securify	×	√	√	√	×	×	×	217
Mythril	√	√	√	×	√	√	√	34
Manticore	×	√	×	×	√	×	×	1468
Machine learning	√	√	√	√	√	√	√	4

Gao et al. proposed SmartEmbed, a vulnerability detection method combining machine learning and feature matching [24]. It learned the features in Solidity source code, compared them based on word embedding and vector space, and parsed the growing text sequence of smart contracts and kept the structured information of the code. Then the sequence is transformed into a numerical vector, which contains semantic information. Using the transformed vector sum and the vector transformed from the source code with vulnerabilities to perform similarity matching, the smart contract is judged to have security vulnerabilities. This approach focuses more on the similarity between vectors, while ignoring the vulnerabilities caused by the relationship between the semantics of the smart contract context. Wang et al. proposed a method to detect the vulnerabilities of smart contracts using machine learning technology. By extracting the opcode features of smart contracts, the machine learning algorithm model was established, which solved the time-consuming problem of traditional symbol execution methods, and improved the detection accuracy and efficiency [25]. The model can extract bytecode features using machine learning algorithms, with an average detection accuracy of 95%. The application of machine learning in this field also lays the foundation of exploiting deep learning for smart contract vulnerability detection. Scarselli et al. proposed the temporal Message Propagation network (TMP) and the undulated graph convolution neural network (DR-GCN) with the help of the graph neural network model, and constructed the contract graph to represent the syntactic and semantic structure of the smart contract function [26]. The authors construct an ablation

process to standardize the graph, highlight the nodes in the contract graph, and normalize the vulnerability detection of graph learning. Experiments show that DR-GCN has obvious advantages in detecting three different types of vulnerabilities, but its disadvantage is that it detects too few types of vulnerabilities [27]. Church proposed a kind of reentrant vulnerability detection based on temporal neural network model. Word2Vec is used to embed words in smart contract fragments, and then the sequential neural network model with attention mechanism is input, which can achieve 90% accuracy [28]. Using temporal memory neural network instead of traditional recurrent neural network can avoid the problems of gradient dispersion and gradient explosion of neural network model, and can capture more context [29]. The disadvantage of using static word embedding is that it cannot better represent the source code of the smart contract, and the context semantics will be lost when the word is represented, which will reduce the detection accuracy of downstream tasks. Jeon et al. used pre-trained BERT model for contract vulnerability detection, and proposed a SmartConDetect to detect security vulnerabilities. The principle of SmartConDetect is to extract code fragments through preprocessing, and use the pre-trained BERT language model to further detect code patterns containing vulnerabilities [30]. Experimental results show that using NLP technology for vulnerability detection can show high performance, and the detection efficiency and accuracy are better than the current methods, but there is still a lot of room for improvement [31]. To sum up, the source code level contains a large amount of semantic, lexical and grammatical information, which can be lost by bytecode, but some useless information can also be filtered out [32]. In order to improve accuracy and reduce the rate of false positives, and can improve the detection efficiency under the condition, we focus more on the source code level of smart contracts, from establishing an expression between the source model. Using the word embedding method combined with the attention mechanism such as deep learning technology of intelligent flaw detection contract, get a more excellent than the existing methods of detection method [33].

3 SolBERT-BiGRU-Attention Hybrid Neural Network Model

In this section, we will introduce the architecture of our neural network model and how to design it in detail. In this study, we innovatively introduced the pre-trained BERT model into the smart contract source code scene, and the downstream task used the timing network combined with the hierarchical attention mechanism to mine more vulnerability characteristic information in the smart contract source code.

3.1 Model Framework

According to the research status of traditional software defect detection methods and the application of machine learning to smart contract vulnerability detection scenario, it is difficult to support the detection of multiple smart contract vulnerabilities, and it is difficult to solve the problems of low detection efficiency and high false positive rate and underreport rate. Fig. 1 is the smart contract vulnerability detection model we designed. The vulnerability detection model of smart contract proposed in this paper is mainly designed from two aspects: word embedding characterization of source code and vulnerability information feature capture. Firstly, we replace the traditional Word2Vec static word embedding model, because in the smart contract source code each word may contain contextual semantic relations, and the word vector should be represented according to the contextual semantics. Since the source code has contextual semantic relations, it is natural for our downstream tasks to use the temporal network model to capture more contextual semantic information, and integrate the hierarchical attention mechanism to mine some important feature information. The last layer of full connection layer maps the feature space calculated by the network of the previous layer into the sample

space, integrates the features into a value to obtain a 7-dimensional vector, and then classifies through the softmax layer to obtain the probability of each vulnerability classification.

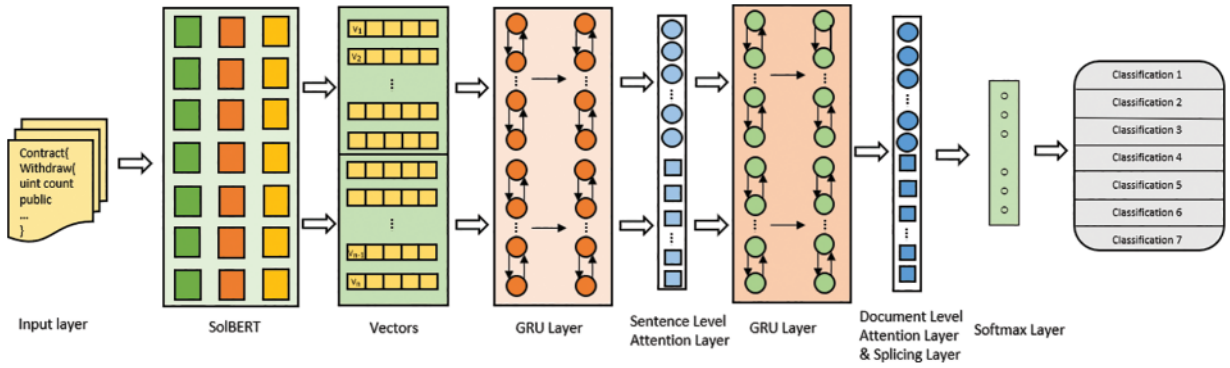

Figure 1: SolBERT-BiGRU-attention model framework

3.2 SolBERT

The pre-trained BERT language model can dynamically embed the word representation of the source code, and can greatly solve the problem of information loss, and better represent the mathematical feature relationship between words and expressions. However, the pre-trained model has not been applied in Solidity source code of smart contracts yet. Its neural network model has complex hierarchical structure, contains 340 million parameters, and has low efficiency of word embedding, so it is difficult to put into real smart contract vulnerability batch detection. Therefore, efficient and accurate word presentation for Solidity source code scenarios of smart contracts needs to be further explored. According to the research experience of TinyBERT, DistillBERT and other models, the proposed SolBERT model is based on the Bert-base model (https://huggingface.co/bert-base-uncased) to do Fine-Tune, in which the Transformer block cutting, knowledge distillation, sharing parameters and other methods to compress our model. The compression method of the model is shown in Fig. 2. A 12-layer BERT-base was used as the Teacher model for training, and our Student model was designed as a 5-layer model, and the vectors of the last two layers were used as the representation of the final smart contract source code, the compressed model parameters are shown in Table 2. The smart contract source code data set we crawled was prepared in advance, and the BERT-Base model and our SolBERT were trained. The model could pay more attention to the context, enhance the characterization of the words in the source code, and provide better characteristic information for the following vulnerability mining and classification tasks. Through the final experiment, the model we proposed has fewer parameters to be trained and fitted, smaller model and lower calculation cost, which solves the problems existing in BERT-base. The current model is about 1/2 of the number of layers of the original model, while ensuring the corresponding accuracy.

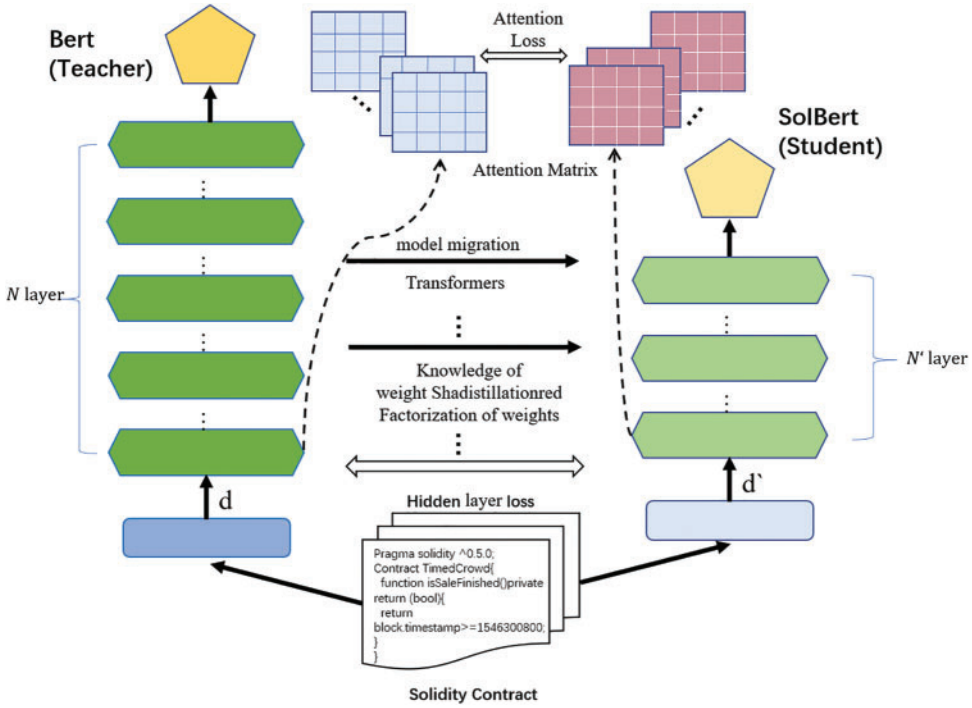

Figure 2: SolBERT model framework

Table 2: BERT-base model and SolBERT parameters

Model	Parameters number	Parameters-sharing	Hidden-layers	Vocab_size
SolBERT	45.8 M	True	5	27402
BERT-base	110 M	False	12	30523

3.3 Bidirectional Gate Recurrent Unit Recurrent Neural Network

There are some problems such as gradient dispersion and gradient explosion when recurrent sequential neural networks are used to learn text information. In the smart contract vulnerability, it is necessary to pay attention to the context information in the source code to determine whether there is a vulnerability. The neural network architecture of the sequential neural network model is studied, and the hierarchical attention mechanism is introduced to focus on the words of the source code, so that the neural network can pay more attention to the information with higher weight. A bidirectional gated recurrent unit neural network model is proposed, as shown in Fig. 3, so that the model can better extract features from the source code sample context of smart contracts, optimize model parameters, and realize efficient and accurate vulnerability detection of smart contracts. With the gradual application of deep learning in smart contract source code vulnerability detection, its problems become more and more prominent. At present, existing research literatures use deep learning methods for smart contract vulnerability detection, which are mainly based on convolutional neural networks, recurrent neural networks and their variants. However, the traditional neural network model has some problems such as gradient disappearance and gradient explosion when training long

sequence data, and the fitting effect of the model is poor. It is difficult to capture the word order, syntactic and semantic rules which are important to the classification results. Recurrent neural network to a long sequence of text such as source code is very weak, because the recurrent time steps of neural network is the last step input to a large extent depends on the time step output information, for the logical sequence far apart in the text of the data information transmission loss is very serious, lead to the loss of semantic code, and can also cause neural network fitting degree is low.

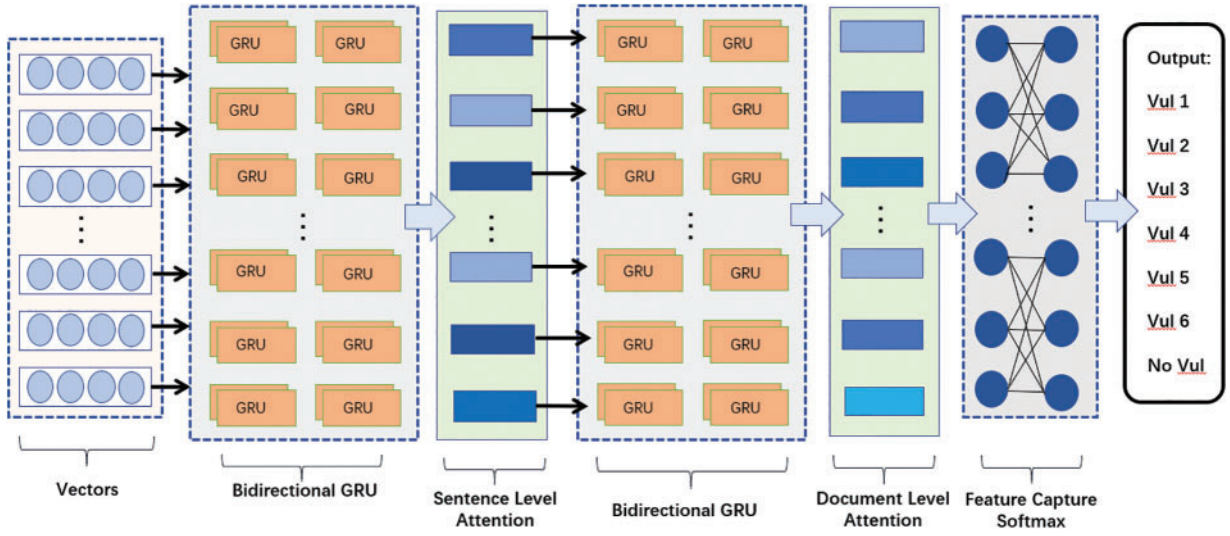

Figure 3: Bidirectional GRU vulnerability detection classification model

Therefore, in our downstream task, we designed a bidirectional GRU gated recurrent unit with hierarchical attention mechanism. The hierarchical attention mechanism has achieved good results in text classification, so we introduced it into our model innovatively. The purpose of introducing hierarchical attention mechanism is that we can find in the source code text of a smart contract with integer overflow vulnerability that the sentences with vulnerability have different weights for text detection to contain the vulnerability. In "*require(totalSupply+_value<=tokenLimit)*", the weights of "*totalSupply+_value<=tokenLimit*" are different. So, in our downstream network we think of text as both text level and sentence level. A two-tier attention mechanism that focuses more on code statements or logical relational operations that are vulnerable. In Figs. 4a and 4b, we can see that each word has a different weight in a sentence of source code, and each sentence of code has a different weight in the entire smart contract file. The higher the importance, the darker the color. Context and semantic relationship will also appear, which is the characteristic information that the hierarchical attention mechanism can highlight.

In addition, there is also semantic dependence between the upper and lower parts of the statement. When a variable is declared above, the following operation is performed, and some logical judgment is not made, the contract has the vulnerability of integer overflow. Therefore, we introduced a bidirectional GRU-gated recurrent neural network model to solve the relationship between RNN recurrent neural network and the large dependence of time step distance in time series. It controls the flow of information by learning a gating unit. The concepts of reset gate and update gate are included in GRU. The input of reset gate and update gate are both the current time step input and the hidden state of the last time step, and the output is calculated by the fully connected layer whose activation function is sigmoid function.

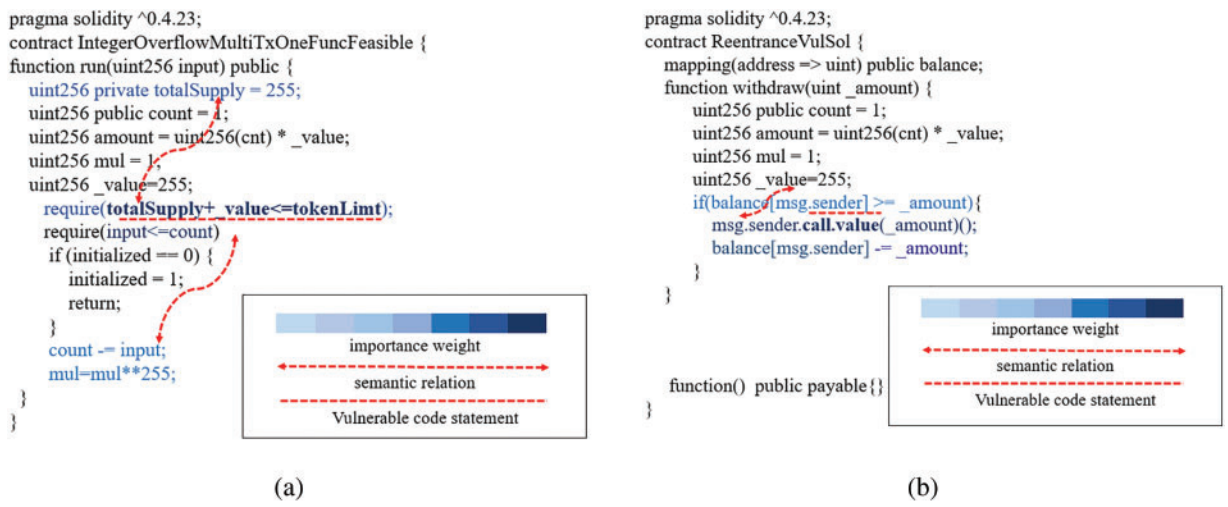

Figure 4: The application of hierarchical attention mechanisms. (a) Semantic relation and weight map of overflow and overflow vulnerability (b) Reentrant vulnerability semantic relation

We compared GRU and LSTM, as shown in Figs. 5 and 6. GRU has one less gating unit and one less parameter than LSTM, and its accuracy is not significantly lower. Therefore, we chose GRU. In the GRU gated neural unit, x_t represents the input information of the smart contract source code word vector at the current time t, and h_{t-1} the hidden state at the last time. The hidden state acts as a neural network memory, which contains the data information previously seen by the node. h_t hidden information passed to the next moment. $\widetilde{h}_t$ candidate hidden state. r_t resets the gate and z_t updates the gate. Sigmoid function, through which data can be changed into values in the range of [0-1]. *tanh*: tanh function, through which data can be changed into data in the range of $[-1, 1]$. The r_t reset gate determines how to combine the new input information with the previous memory. The formula is: W_r is not a value, but a weight matrix that is used to linearly transform the concatenated matrix of x_t and h_{t-1}. Then you multiply the values of the two matrices into the sigmoid function, and you get the value of r_t. The obtained reset gate value will be brought into the candidate hidden state. When the value of r_t is smaller, the value of the matrix produced by the product of r_t and h_{t-1} Hadamard will be smaller, and then the value obtained by the multiplication of the weight matrix will be smaller, indicating that more information input at the last moment needs to be forgotten, and more information is discarded. The larger the value of r_t, the more information needs to be remembered at the last moment, and the newer input information is combined with the previous memory. When the value of r_t is closer to 0, it indicates that only the input at the current time needs to be discarded. Therefore, all historical information unrelated to prediction can be discarded. When the value of r_t is close to 1, it indicates that the hidden state of the previous time is retained. This is where the reset gate plays an important role, helping to capture short-term dependencies in the time series.

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \tag{1}$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \tag{2}$$

$$\widetilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \tag{3}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t * \widetilde{h}_t \tag{4}$$

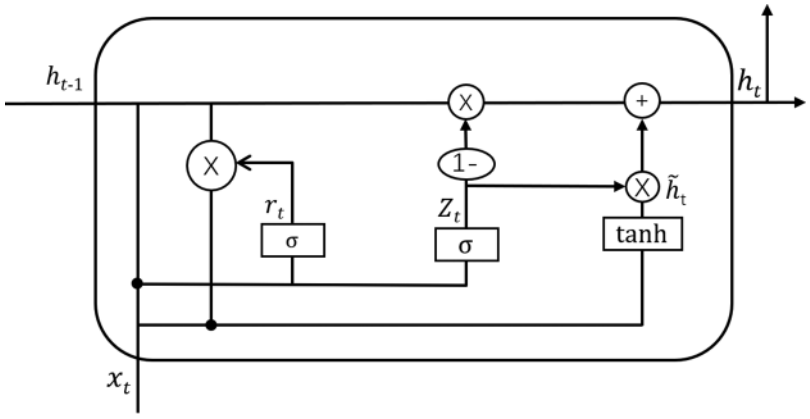

Figure 5: Gate recurrent unit structure

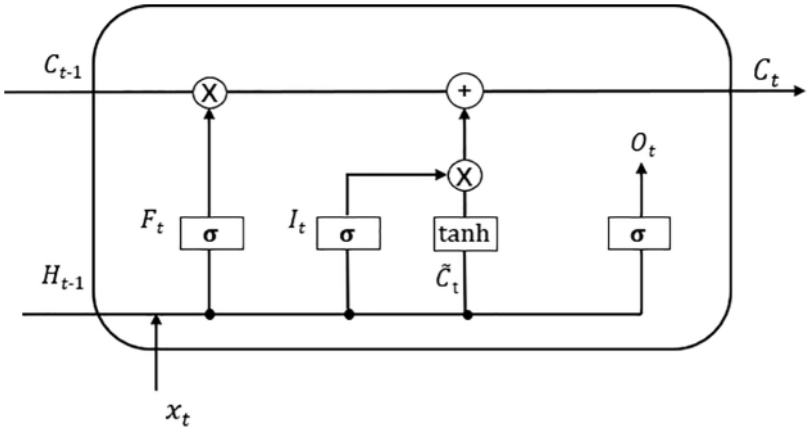

Figure 6: LSTM structure

The update gate is used to control the degree to which the state information of the previous moment is brought into the current state, that is, the update gate helps the model decide how much information of the past should be transferred to the future, which is simply used to update the memory. The closer z_t is to 1, the more data is "memorized", while the closer it is to 0, the more it is "inherited". $(1 - z_t) \cdot h_{t-1}$ indicates selective "forgetting" of the hidden state at the last moment. Forget the unimportant information in h_{t-1} and discard the irrelevant information. $z_t * h_t$ indicates a further selective "memory" of candidate hidden states. The update gate will forget some unimportant information in h_t, and will further select certain information in $\tilde{h}_t$. The vector h_t forgets some information of h_{t-1} passed down, and adds some information of current node input, which is the final memory. The gated recurrent unit GRU will not clear the previous information over time, and will retain the relevant information to be transferred to the next unit. The hierarchical attention mechanism is based on the hierarchical nature of documents. Sentences are composed of words. In the source code of smart contracts, the code is also composed of words and symbols, paragraphs are composed of sentences, and program source files in the source code are composed of code statements. Therefore, program source files are structured hierarchically just like documents. Therefore, the hierarchical attention mechanism can stratify words and sentences, and the word encoder can summarize the information at the word level and input it into the sentence encoder. The sentence encoder can

summarize the information at the sentence level and finally output the classification probability of the document.

The hierarchical attention mechanism consists of four parts: Word Encoder, Word Attention, Sentence Encoder, and Sentence Attention, as shown in Fig. 7.

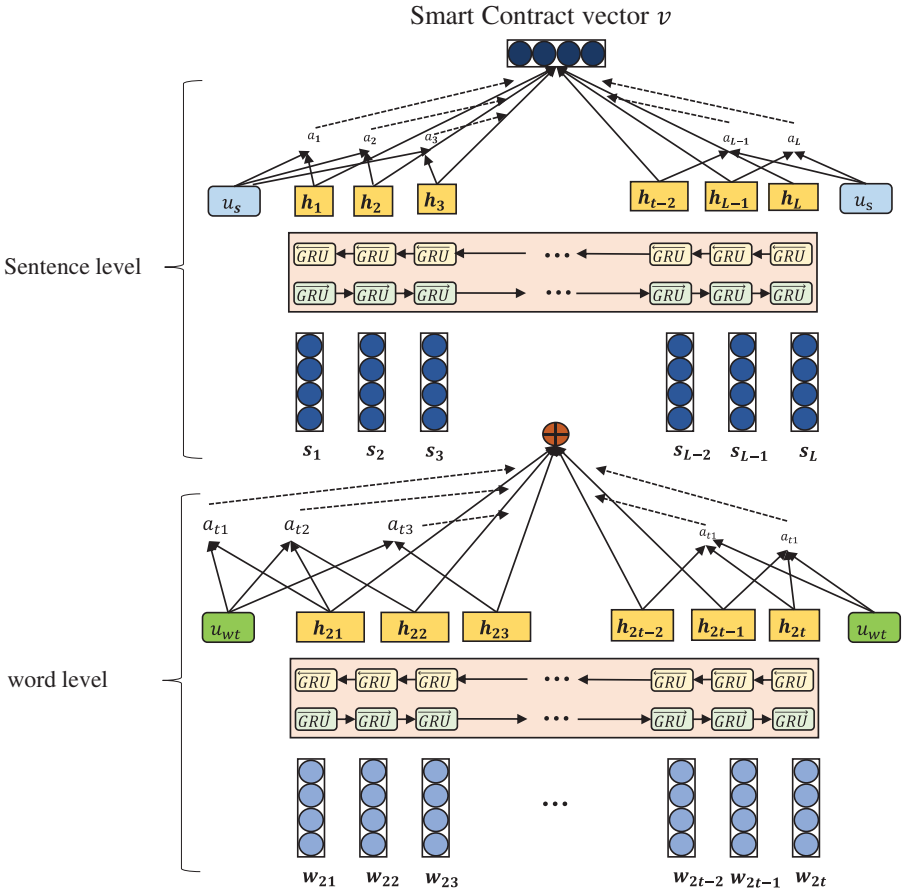

Figure 7: Hierarchical attention mechanism

Word encoder: A piece of smart contract source code contains several words w_{it},t $\in [1, T]$. Firstly, the word vector x_{ij} obtained by the upstream task is used. The bidirectional GRU is used to obtain the information in both directions of the source word to capture the context of the word relation. The bidirectional GRU includes forward $\vec{h}$ and $\overleftarrow{h}$.

$$\vec{h}_{it} = \overrightarrow{GRU}\left(x_{it}\right), t \in [1, T] \tag{5}$$

$$\overleftarrow{h}_{it} = \overleftarrow{GRU}\left(x_{it}\right), t \in [T, 1] \tag{6}$$

$$h_{it} = \left[\vec{h}_{it}, \overleftarrow{h}_{it}\right] \tag{7}$$

Word attention mechanism: In the smart contract source code, each code statement may not all have the same effect on semantic expression, so it is necessary to use the soft attention mechanism to extract important words in the source code statement to form a sentence vector. W_w, u_w, b_w are all

parameters that need to be fitted by training. First, input the word vector h_{it} through the MLP layer to obtain the hidden layer, then use the word-level context u_w to measure the importance of the word, and then obtain a normalized importance weight through the softmax function. Then we calculate the sentence vector S_i as the weighted sum of the word weights.

$$u_{it} = \tanh(W_w h_{it} + b_w) \tag{8}$$

$$a_{it} = \frac{\exp(u_{it}^T u_w)}{\sum_t \exp(u_{it}^T u_w)} \tag{9}$$

$$S_i = \sum_t a_{it} h_{it} \tag{10}$$

Sentence-level encoder: Given source code sentence vector S_i, the document vector is obtained using encoding and attention mechanisms similar to word vectors. Bidirectional GRU is also used for encoding. Aggregate the vectors from front to back and back to front of the sentence, but focus most on the *i* vector.

$$\vec{h}_i = \overrightarrow{GRU}(S_i), t \in [1, T] \tag{11}$$

$$\vec{h}_i = \overrightarrow{GRU}(S_t), t \in [L, 1] \tag{12}$$

$$h_i = \left[\vec{h}_i, \overleftarrow{h}_i\right] \tag{13}$$

Sentence attention mechanism: In the same way as word-level attention mechanism, a sentence-level contextual parameter u_s is introduced, which can be learned to measure the importance of sentences to vulnerability classification. The V vector is the summary feature of the entire input smart contract source code sentence.

$$u_i = \tanh(W_s h_i + b_s) \tag{14}$$

$$a_i = \frac{\exp(u_i^T u_w)}{\sum_i \exp(u_i^T u_s)} \tag{15}$$

$$v = \sum_i a_i h_i \tag{16}$$

The hierarchical attention mechanism process includes Word encoder, Word attention, Sentence attention, softmax to calculate the attention weight by word encoder, and then the important words in the sentence forming sentence can be obtained as input. And the attention weight is calculated to get the important sentences in the text content.

The document vector V is the high-level representation of the entire smart contract, which is fed into the softmax function, using the negative log-likelihood of the correct label as the training loss, and *j* is the classification of the document.

$$p = \text{softmax}(\mathrm{W_c v} + \mathrm{b_c}) \tag{17}$$

$$L = -\sum_d \log p_{dj} \tag{18}$$

4 Experiment Results and Analysis

Through comparative experiments on upstream and downstream tasks, multiple indicators of the dataset are evaluated to find out whether our proposed model is superior to the existing models.

4.1 Experimental Environment

This paper proposes to use our proposed SolBERT-BiGRU-Attention hybrid neural network model to detect reentrant vulnerabilities in smart contracts. The data source is the latest real smart contract in the Ethereum network crawled by the block and smart contract browser provided by Ethereum official. Our experimental environment is all from Intel Xeon E5 quad-core processor, CPU frequency 3.6 GHz, memory 128 G, graphics card RTX 1080 Ti. In the experiment, Python 3.7 is used to write scripts to crawl data sets and preprocess smart contracts. Tensorflow-gpu, Keras 2.2.4 and PANDAS are used as deep learning framework. Sklearn library is used as the final experimental model performance evaluation. The specific experimental environment is shown in Table 3.

Table 3: Experimental environment details

Environment	Detail
Operating system	Windows 10 & Ubuntu
Memory	128 G
CPU	Intel Xeon E5 3.6 GHz
GPU	NVIDIA GeForce GTX 1080 Ti
Language	Python 3.7

4.2 Comparative Experiment Settings

In order to verify the effectiveness and superiority of the trained model, 80% of the crawling smart contract data set is divided into the training set and 20% is divided into the test set. The training set is used to fit our detection model, and the test set evaluates the performance of the model. The dataset contains a total of 37232 smart contracts. Our experiment examines six kinds of smart contract vulnerabilities. In the data samples, our smart contracts may contain multiple smart contract vulnerabilities.

In the comparison experiment, we set up eight groups of comparison experiments, among which three groups are traditional static analysis method tools and one group is the experiment solved by deep learning method.

(1) Oyente: Static analysis of the bytecode of smart contracts compiled in EVM based on symbolic execution. Vulnerability detection requires compilation of contracts rather than direct access to the source code.

(2) Mythril: A static analysis tool based on symbolic execution and taint analysis that verifies the existence of security vulnerabilities by using the transactions required to compute the contract bytecode.

(3) Osiris: A static analysis smart contract vulnerability tool, sensitive to the detection of integer overflow vulnerabilities, but not limited to the detection of integer overflow vulnerabilities.

(4) Word2Vec-RNN: The static word embedding model combined with RNN timing cycle neural network is used to compare our model and reflect the improvement of our model on this basis.

(5) BERT-GRU: The dynamic word embedding model is replaced by the traditional static word embedding model, and the word embedding model has better representation than the traditional model. The GRU gated neural network is added to verify and prevent gradient dispersion.

(6) SolBERT-GRU: Using our compressed neural network model combined with the downstream gated recurrent unit neural network, it is proved that the word embedding of our compressed model is more efficient than the uncompressed word embedding model BERT.

(7) SolBERT-BiGRU: The downstream BiGRU can capture more contextual semantics in smart contracts, and improve more detection accuracy.

(8) SolBERT-BiGRU-Attention: The compressed Bert model is used for word embedding of the smart contract source code, and the bidirectional GRU model with attention mechanism is used downstream. It is proved that hierarchical attention mechanism can make our model more sensitive to the characteristics of smart contract vulnerabilities.

4.3 Evaluation Metrics

The evaluation indexes of several groups of comparative experiments and the neural network model proposed in our experiment are similar to the text classification task, so the evaluation indexes we adopted are divided into five evaluation indexes: accuracy, true positive rate, false positive rate, precision rate and precision rate precision. Index calculation formula is as follows: True Positive (TP): The predicted result indicates that the vulnerability exists. The actual tag also has the vulnerability. False Positive (FP): The prediction result indicates that the vulnerability exists. The actual label indicates that the vulnerability does not exist. False Negative (FN): The predicted result is that no vulnerability exists. The actual label indicates that the vulnerability exists. True Negative (TN): The predicted result is that no vulnerability exists. The actual label indicates that no vulnerability exists.

The accuracy rate is the number of correctly classified samples in the test set, which is defined as follows:

$$Accuracy = \frac{(TP + TN)}{(TP + FP + FN + TN)} \tag{19}$$

Among all positive classes, how many true positive rates are predicted to be positive classes, also known as Recall, is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{20}$$

Precision refers to the proportion of predicted positive samples that are actually positive. The formula is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{21}$$

F1-score (F1 value), also known as F1-measure, is an index that comprehensively considers Precision and Recall and is defined as follows:

$$F1 - Score = \frac{2 * (Precision * Recall)}{(Precision + Recall)} \tag{22}$$

4.4 DateSet

According to literature records, there is currently no standard open source vulnerability dataset in the field of smart contract vulnerability detection. However, some existing vulnerability smart contract data can be collected from some papers and vulnerability library websites. But such datasets are not reliable, and the proportion of positive and negative samples varies greatly. Contracts with vulnerabilities in most papers are detected and determined by traditional smart contract detection tools and then marked. These traditional tools are not authoritative, so we need to add manual judgment on the basis of the original tool detection to improve their label calibration. Correctness and improve the quality of our training set. Therefore, our data set sources include: data sets published in the paper, smart contracts existing in vulnerability libraries, and crawling related smart contracts using crawlers from the Ethereum platform. The final total number of smart contracts includes 35,232. There are 11,012 smart contract samples with integer overflow vulnerabilities, 11,012 contract samples with underflow vulnerabilities, 2,421 contract samples with transaction dependency vulnerabilities, 205 contracts with unchecked return values, and 3,578 contract samples with timestamp dependencies. And there are 4,262 contracts with reentrancy vulnerabilities and unchecked return value vulnerabilities. Each smart contract sample may contain multiple vulnerabilities. Therefore, as shown in Table 4, we show that the table contains a certain vulnerability.

Table 4: Experimental data set ratio

Type	Number		
	Vulnerable	Invulnerable	Total
OverFlow	11,012	24,220	35,232
UnderFlow	10,023	25,209	35,232
TimeStamp dependency	3,578	31,654	35,232
Reentrancy vulnerability	4,262	30,970	35,232
Unchecked return value	4,262	30,970	35,232
Transaction order dependence	2,421	32,881	35,232

4.5 Analysis of Results

By comparing two sets of comparison experiments, compared with traditional detection tools, and compared with other four kinds of neural network models, the effect of introducing hierarchical attention mechanism is verified experimentally. We also carried out comparative experiments on the Bert-base model and the traditional Word2Vec static word embedding model with our model, and proved that our optimized BERT model could represent the vulnerability of the smart contract better, making the final detection result of the smart contract more accurate. In terms of detection time, It also takes less time than traditional smart contract vulnerability detection tools.

4.5.1 Overall Comparison

Table 5 shows the experimental results after preprocessing according to the data set we crawled. The comparison experiment was mainly divided into two groups. The first group compared the hybrid model SolBERT-BiGRU-Attention model we proposed with several good smart contract vulnerability detection models in the industry. Comparison indexes were Accuracy, Precision, Recall and F1-score, and detection efficiency was introduced. The other group is mainly compared with the machine

learning model, comparing the improvement of the word embedding model in the representation of the smart contract source code, which can then affect the vulnerability detection of the downstream neural network model. It can be seen from the figure that the proposed hybrid model has much higher accuracy than the traditional smart contract vulnerability detection model, with the average detection accuracy of 90% or above and the detection efficiency about 10 times higher. In terms of the detection time of the model, it can also be found that the compressed BERT model we proposed has better detection efficiency than BERT-base. In the evaluation of the entire model, we used the Micro-F1 Score to evaluate the detection accuracy and recall rate of the model more fairly. As shown in Table 4, the model we proposed can have higher accuracy and lower false alarm rate than some of the latest detection methods in terms of Accuracy, Precision, Recall, and Micro-F1. Finally, while ensuring the accuracy, it also improves the training efficiency of the model and reduces the training cost of the model.

Table 5: The results of eight comparative experiments

	Accuracy (%)	Precision (%)	Recall (%)	Micro-F1 (%)	Average times (Sec)
Oyente	46.32	54.23	60.24	59.92	30
Mythril	56.59	61.54	59.21	61.23	84
Osiris	69.76	62.42	62.12	62.56	34
Word2Vec-RNN	83.18	83.13	80.15	81.61	1.3
BERT-GRU	86.22	85.24	84.67	84.96	6.5
SolBERT-GRU	89.55	89.64	87.34	88.47	4.2
SolBERT-BiGRU	92.01	91.97	92.10	92.03	4.3
SolBERT-BiGRU-attention	93.85	93.92	94.11	94.02	4.5

In the test data set of the data set, we first compare our proposed model with the traditional smart contract vulnerability detection model. The corresponding ROC image is shown in Fig. 8. It can be compared that the true positive rate of our model is higher than that of the static word, The embedding model combines the RNN sequence neural network, and the false positive rate is lower than other models. The area of the ROC curve in SolBERT-BiGRU-Attention is about 0.94, and the detection effect can be performed very well. As shown in the histogram in Fig. 9, our model can reach 93.92% on TPR, which is 13.55% higher than the traditional model, and the false positive rate is much lower than the static word embedding model. In addition, adding the hierarchical attention mechanism and not adding the hierarchical attention mechanism is 1.82%, which can capture more contextual semantics in the source code of the smart contract. Several experimental indicators show that we have successfully integrated the Attention mechanism and the bidirectional sequential neural network. It also shows that in the smart contract source code scenario, the dynamic word embedding model will perform better than the static word embedding model.

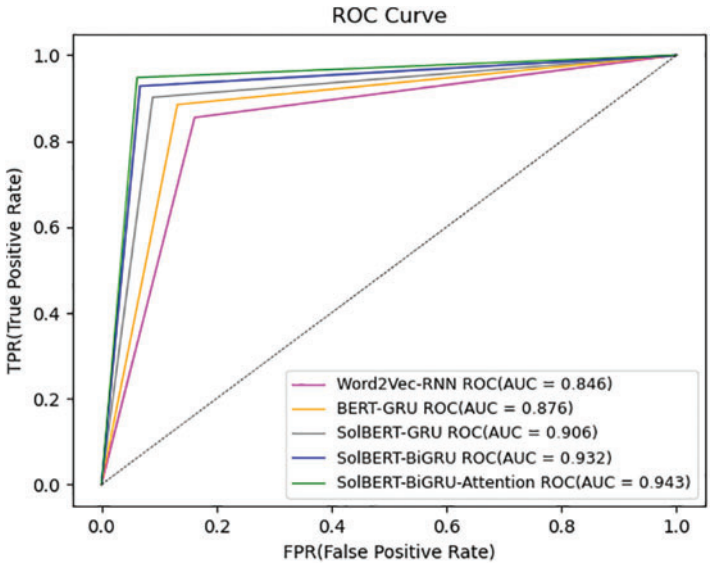

Figure 8: Experimental ROC curve

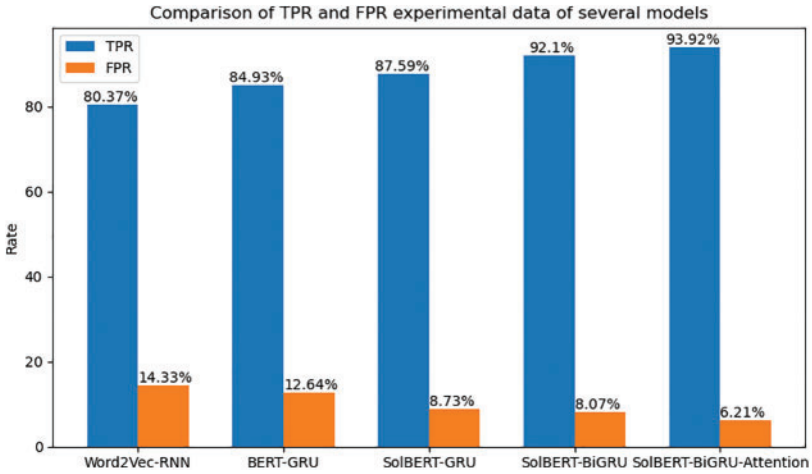

Figure 9: Comparison of TPR and FPR experimental data of several models

Our data set has a total of 35,232 samples. When we choose Batch Size during model training, we choose 256 pairs of models for training. The number of smart contract samples to be trained for each Epoch is 35,232 complete data sets, a total of 138 iterations. Each epoch iterates the model parameters 138 times, and the experiment is set to 100 epochs. During the experiment, at the 80th and 90th epochs, as shown in Fig. 10, the Accuracy of each model is already in a convergent state. In the two-way GRU model that does not use the attention mechanism, it can be found that the Accuracy has a declining process, and it is very likely that the model has an overfitting problem. However, we did not have this problem when we used the hierarchical attention mechanism. It can be explained to a certain extent that the hierarchical attention mechanism can improve the robustness of our model and have stronger generalization ability.

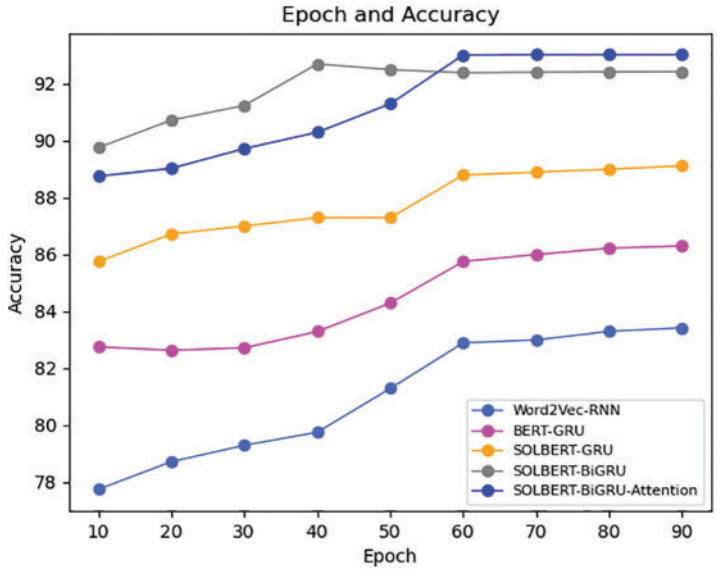

Figure 10: Epoch and accuracy of model training

5 Conclusion and Future Work

In this research, we propose a hybrid neural network model based on SolBERT-BiGRU-Attention for smart contract vulnerability detection. The vulnerabilities that can be detected include 6 classifications, including Integer overflow, Integer underflow, reentrant vulnerability, timestamp dependency vulnerability, transaction order dependency vulnerability, unchecked return value vulnerability. Compared with the existing smart contract vulnerability detection methods, our proposed model can achieve a detection accuracy of 94% probability, and the false positive rate can be guaranteed to be below 10%, which can have a better detection effect than the current model. The detection efficiency can also be guaranteed to be around 4 s. If the traditional method of smart contract vulnerability detection is dynamic detection, it will rely on the current solidity version, while the method of smart contract vulnerability detection using the machine learning method does not rely on the solidity execution process and results, which will greatly improve its running efficiency. At present, many pre-trained language models are gradually applied to the field of code analysis, and the performance is also quite remarkable. In the future, the research may be extended to more smart contracts to improve the robustness of the model. There will certainly be many smart contract vulnerabilities that we have not discovered and classified, and our model also needs to learn more data to improve the sensitivity of the model. The second plan is to use a more lightweight dynamic pre-trained language model for word meaning transformation to reduce the training cost of the model and improve its detection accuracy.

Funding Statement: This work is supported by the National Natural Science Foundation of China (Grant Nos. 62272120, 62106030, U20B2046, 62272119, 61972105); the Technology Innovation and Application Development Projects of Chongqing (Grant Nos. cstc2021jscx-gksbX0032, cstc2021jscx-gksbX0029).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Xu, G., Dong, J., Ma, C., Liu, J., Cliff, U. G. O. (2022). A certificateless signcryption mechanism based on blockchain for edge computing. *IEEE Internet of Things Journal*. https://doi.org/10.1109/JIOT.2022.3151359
2. Zheng, Z., Xie, S., Dai, H. N. (2020). An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems, 105,* 475–491. https://doi.org/10.1016/j.future.2019.12.019
3. Xie, M., Liu, J. (2022). A survey on blockchain consensus mechanism: Research overview, current advances, and future directions. *International Journal of Intelligent Computing and Cybernetics*. https://doi.org/10.1108/IJICC-05-2022-0126
4. Khan, S. N., Loukil, F., Ghedira, C. (2021). Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Networking and Applications, 14(5),* 2901–2925. https://doi.org/10.1007/s12083-021-01127-0
5. Raj, A., Maji, K., Shetty, S. D. (2021). Ethereum for Internet of Things security. *Multimedia Tools and Applications, 80(12),* 18901–18915. https://doi.org/10.1007/s11042-021-10715-4
6. Gao, J., Liu, H., Liu, C. (2019). Easyflow: Keep ethereum away from overflow. *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, pp. 23–26. Montréal.
7. Li, J., Yang, Y., Lv, S. (2019). Attention-based BiGRU-CNN for Chinese question classification. *Journal of Ambient Intelligence and Humanized Computing*. https://doi.org/10.1007/s12652-019-01344-9
8. Bokan, M. E. (2022). Negative-sampling word-embedding method. *Scientific Journal of Astana IT University, 2022,* 35–42. https://doi.org/10.37943/ELGD6408
9. Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *International Conference on Empirical Methods in Natural Language Processing*, pp. 4171–4186. Minneapolis.
10. Yamak, P. T., Yujian, L., Gadosey, P. K. (2019). A comparison between ARIMA, LSTM, and GRU for time series forecasting. *Proceedings of the 2th International Conference on Algorithms, Computing and Artificial Intelligence*, pp. 49–55. Sanya.
11. Wang, Z., Jin, H., Dai, W. (2021). Ethereum smart contract security research: Survey and future research opportunities. *Frontiers of Computer Science, 15(2),* 1–18. https://doi.org/10.1007/s11704-020-9284-9
12. Di, A. M., Salzer, G. (2019). A survey of tools for analyzing ethereum smart contracts. *Proceedings of the 1st International Conference on Decentralized Applications and Infrastructures*, pp. 69–78. Fremont.
13. Almakhour, M., Sliman, L., Samhat, A. E. (2020). Verification of smart contracts: A survey. *Pervasive and Mobile Computing, 67,* 101227. https://doi.org/10.1016/j.pmcj.2020.101227
14. Chen, J., Xia, X., Lo, D. (2021). DefectChecker: Automated smart contract defect detection by analyzing evm bytecode. *IEEE Transactions on Software Engineering, 48,* 2189–2207. https://doi.org/10.1109/TSE.2021.3054928
15. Tang, X., Zhou, K., Cheng, J. (2021). The vulnerabilities in smart contracts: A survey. *Proceedings of the 7th International Conference on Artificial Intelligence and Security*, pp. 177–190. Dublin.
16. Xie, X., Li, X., Chen, X., Meng, G., Liu, Y. (2019). Hybrid testing based on symbolic execution and fuzzing. *Journal of Software, 30(10),* 3071–3089.
17. Luu, L., Chu, D. H., Olickel, H. (2016). Making smart contracts smarter. *2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 254–269. Vienna.
18. Tsankov, P., Dan, A., Drachsler, D. (2018). Securify: Practical security analysis of smart contracts. *2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 67–82. Toronto.
19. Singh, A., Parizi, R. M., Zhang, Q. (2020). Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. *Computers & Security, 88,* 101654. https://doi.org/10.1016/j.cose.2019.101654

20. Mossberg, M., Manzano, F., Hennenfent, E. (2019). Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1186–1189. San Diego.
21. Ivanov, N., Lou, J. Z., Chen, T., Li, J., Yan, Q. B. (2021). Targeting the weakest link: Social engineering attacks in Ethereum smart contracts. *2021 ACM Asia Conference on Computer and Communications Security*, pp. 787–801. Hong Kong, China.
22. Jiang, B., Liu, Y., Chan, W. K. (2018). ContractFuzzer: Fuzzing smart contracts for vulnerability detection. *2018 33rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 259–269. Montpellier.
23. Demertzis, K., Iliadis, L., Tziritas, N. (2020). Anomaly detection via blockchained deep learning smart contracts in Industry 4.0. *Neural Computing and Applications, 32(23),* 17361–17378. https://doi.org/10.1007/s00521-020-05189-8
24. Gao, Z., Jayasundara, V., Jiang, L. (2019). Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding. *Proceedings of the 6th IEEE International Conference on Software Maintenance and Evolution*, pp. 394–397. Shanghai, China.
25. Wang, W., Song, J., Xu, G. (2020). Contractward: Automated vulnerability detection models for ethereum smart contracts. *IEEE Transactions on Network Science and Engineering, 8(2),* 1133–1144. https://doi.org/10.1109/TNSE.2020.2968505
26. Scarselli, F., Gori, M., Tsoi, A. C. (2008). The graph neural network model. *IEEE Transactions on Neural Networks, 20(1),* 61–80. https://doi.org/10.1109/TNN.2008.2005605
27. Zhuang, Y., Liu, Z., Qian, P. Liu, Q, Wang, X. et al. (2020). Smart contract vulnerability detection using graph neural network. *International Joint Conferences on Artificial Intelligence*, pp. 3283–3290. Yokohama.
28. Church, K. W. (2017). Word2Vec. *Natural Language Engineering, 23(1),* 155–162. https://doi.org/10.1017/S1351324916000334
29. De Mulder, W., Bethard, S., Moens, M. F. (2015). A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language, 30(1),* 61–98. https://doi.org/10.1016/j.csl.2014.09.005
30. Jeon, S., Lee, G., Kim, H. (2021). *SmartConDetect:* Highly accurate smart contract code vulnerability detection mechanism using BERT. *2021 KDD Workshop on Programming Language Processing*, 237102485. Virtual Event.
31. Gao, Z. (2020). When deep learning meets smart contracts. *2020 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1400–1402. Virtual Event.
32. Amani, S., Begel, M., Bortin, M., Staples, M. (2018). Towards verifying ethereum smart contract bytecode in Isabelle/HOL. *2018 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 66–77. Los Angeles.
33. Vaswani, A., Shazeer, N., Parmar, N. (2017). Attention is all you need. *International Conference on Neural Information Processing Systems*, pp. 6000–6010. California, USA.