A High-Performance Smart Contract Vulnerability Detection Scheme Based on BERT

Shengqiang Zeng
Tianjin University of Technology
Tianjin, China
poilzero@stud.tjut.edu.cn

Ruhuang Chen
Tianjin University of Technology
Tianjin, China
chenruhuang@stud.tjut.edu.cn

Hongwei Zhang*
Tianjin University of Technology
Tianjin, China
hwzhang@email.tjut.edu.cn

Jinsong Wang
Tianjin University of Technology
Tianjin, China
jswang@tjut.edu.cn

Abstract*—With the emergence of technologies like web3.0, smart contracts have witnessed a flourishing development trend. However, the threat posed by contract vulnerabilities hinders the progress in this field. Traditional vulnerability detection tools have lost their effectiveness due to the unique code and function characteristics of smart contracts. Consequently, a novel approach utilizing deep learning for intelligent contract vulnerability detection has emerged. Nevertheless, the current solutions still face bottlenecks in terms of accuracy and efficiency, primarily due to the scarcity of labeled vulnerability samples. To address these challenges, this paper proposes an efficient intelligent contract vulnerability detection approach called SCVulBERT, based on Bidirectional Encoder Representation from Transformers (BERT). The proposed approach leverages transfer learning and utilizes rich prior knowledge for training to ensure the model's effectiveness in a scarce supervised sample environment. Furthermore, to enhance tokenization efficiency, a specialized tokenizer called SCVulTokenizer is designed to transform contract code into parameters recognizable by neural networks. The proposed approach utilizes the BERT network architecture to extract more precise and efficient features from the context, thereby achieving accurate and efficient vulnerability detection. Experimental comparisons demonstrate that the proposed approach outperforms existing solutions in the context of scarce supervised samples, exhibiting significant improvements in accuracy, precision, recall, and F1-score metrics. Specifically, regarding vulnerability detection for reentrancy, timestamp, and delegate call, the F1-scores achieved by the proposed approach show respective improvements of 13.71%, 13.14%, and 7.7% compared to the state-of-the-art solutions.

***Index Terms*—smart contract, BERT, vulnerability detection, transfer learning, blockchain**

I. Overview

Smart contracts, as a critical core component of blockchain technology [1], have experienced rapid development in recent years [2]. However, this growth has been accompanied by a multitude of security risks [3] [4]. In 2018, the BEC (Beauty Chain) platform fell victim to a third-party integer overflow vulnerability attack, resulting in a staggering loss of 6 billion US dollars. In 2020, criminals exploited a delegate call vulnerability to infiltrate the Lendf.Me platform, leading to the hijacking of virtual currency worth 25 million US dollars. These incidents have accelerated the research efforts of security professionals towards developing more precise, efficient, and universally applicable techniques for detecting smart contract vulnerabilities.

Presently, the related research can be categorized into two main classes: traditional approaches and neural network-based methods. Traditional approaches heavily rely on expert experience, exhibit limited automation, and demonstrate lower generality. Neural networks, on the other hand, can mitigate these drawbacks of traditional approaches. They do not require predefined patterns of vulnerabilities and can achieve efficient detection with a sufficient number of labeled samples. Nevertheless, due to the scarcity of labeled samples, there remains significant room for improvement. Furthermore, the recurrent neural networks (RNNs) or graph convolutional neural networks (GCNs) utilized in relevant approaches are constrained by the size of their receptive fields, leading to the issue of memory degradation as text length increases [5].BERT (Bidirectional Encoder Representation from Transformers) [6], as a novel natural language processing model, effectively mitigates the memory degradation issue by employing attention mechanisms, ensuring that the accuracy of text encoding does not diminish with increasing text length [7]. Research revolving around BERT has progressively expanded into other domains, yielding promising outcomes [8].This approach is founded on the BERT model, leveraging the BERT network to automatically capture contract features for vulnerability detection. Additionally, it employs transfer learning to address the scarcity of labeled samples.

This paper primarily makes the following contributions:

1) Addressing the inadequacy of feature processing performance in deep learning-based approaches, this work significantly enhances feature extraction capabilities through the utilization of BERT;
2) In pursuit of enhancing the overall solution performance,

This research was supported by the National Natural Science Foundation of China (Project No. 62072336) and the Tianjin Major Special Project on Next-Generation Artificial Intelligence Technology (Project No. 19ZXZNGX00080).

a tokenizer model is proposed to account for the tokenization characteristics of smart contracts;

3) Under the constraint of a lack of labeled samples for contract vulnerabilities, the accuracy of identification is improved through the application of transfer learning.

II. Related Work

The detection of smart contract vulnerabilities can primarily be categorized into two approaches: traditional methods and neural network-based methods. Traditional approaches are rule-based, relying on predefined rules that require in-depth analysis of various vulnerabilities to derive specific strategies. These methods exhibit artificial uncertainties in terms of generality and performance. Formal verification approaches employ mathematical logic to conduct empirical completeness checks on contracts [9] [10]. While these methods are theoretically derivable, they necessitate specialized contract models and domain expertise, resulting in a high entry barrier; Symbolic execution methods, such as Oyente [11], Mythril [12], simulate the execution of smart contracts at the symbolic level to detect vulnerabilities. While these methods can perform detection without actually executing contracts, they suffer from the problem of path ex plosion, leading to infinite solving times [13];Fuzzy detection methods such as Regurad [14], and ILF [15], generate a substantial amount of fuzzy input data to test smart contracts for vulnerabilities. While these methods are simple and user-friendly, they are prone to false negatives due to their inability to cover all possible inputs. Additionally, there is a method known as intermediate representation, which detects vulnerabilities by examining intermediate languages in the compilation process, such as Low-Level Virtual Machine (LLVM) [16]. In contrast to the aforementioned approaches, neural network-based detection methods train on labeled samples of smart contract vulnerabilities and continually learn from obtained weights to uncover potential contract vulnerabilities. Tann et al. [17] were among the first to apply neural networks to the field of smart contract vulnerability detection. Their approach enhanced the Maian tool by introducing Long Short Term Memory (LSTM) [18] networks, resulting in a more effective semi-automated detection model named SaferSC. Compared to traditional methods, SaferSC achieved higher accuracy; Qian et al. [19] introduced a smart contract vulnerability detection approach utilizing the Word2Vec [20] model, recurrent neural networks, and attention mechanisms [21]. The method involved tokenizing and vectorizing contracts using Word2Vec, extracting features through Bidirectional LSTM (BLSTM) and attention layers, and ultimately conducting detection based on the extracted contract features. Zhuang et al. in 2021 [31] proposed a novel approach that employs a degree-free graph convolutional neural network (DR-GCN) and a temporal message propagation network (TMP) for smart contract vulnerability detection. This method significantly outperforms traditional rule-based and neural network-based approaches in detecting various types of vulnerabilities. The above-mentioned approaches still fall short in addressing the pressing need for accuracy and efficiency brought about by the sheer scale and variety of smart contracts. Notably, these approaches face the challenge of scarce labeled vulnerability samples. Furthermore, as most of these methods are based on supervised learning, they heavily rely on limited training data, posing significant performance bottlenecks

III. The SCVulBERT Approach

Addressing the current limitations of low detection accuracy and restricted detection performance due to the scarcity of labeled samples, this paper introduces a novel smart contract vulnerability detection approach called SCVulBERT (Smart Contract Vulnerability BERT). SCVulBERT preprocesses smart contract code to extract essential text, reducing the impact of irrelevant information on vulnerability detection results. To enhance tokenization performance and thereby further improve the model's feature extraction capabilities, this paper introduces the SCVulTokenizer tokenizer, which segments smart contract code. Furthermore, SCVulBERT inputs the extracted code text into the BERT feature extraction network to extract feature vectors representing high-level semantics. Finally, these feature vectors are fed into a classifier, ultimately achieving more efficient and accurate vulnerability detection. The overall logic of SCVulBERT is illustrated in Figure 1, and the subsequent sections will provide detailed explanations of each step.

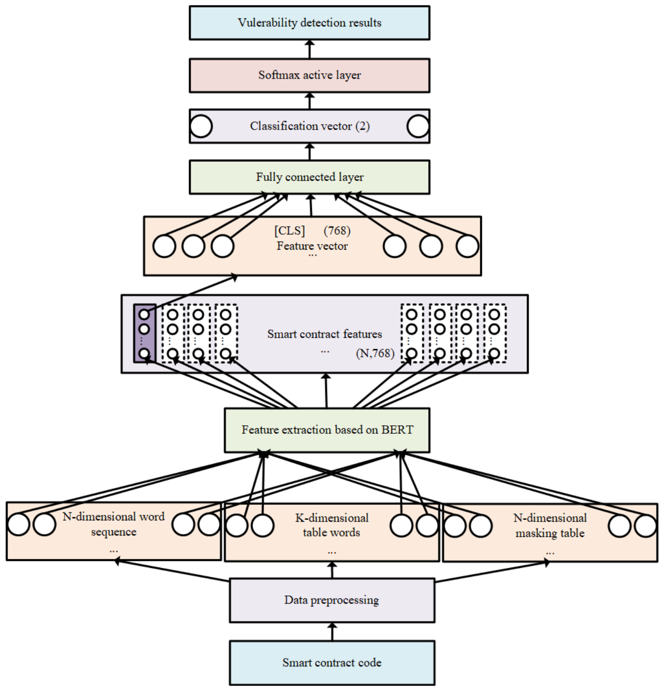

Vulerability detection results
Softmax active layer
Classification vector (2)
Fully connected layer
[CLS] (768)
Feature vector
...
Smart contract features
... (N,768)
Feature extraction based on BERT
N-dimensional word sequence
...
K-dimensional table words
...
N-dimensional masking table
...
Data preprocessing
Smart contract code

Fig. 1. SCVulBERT structure

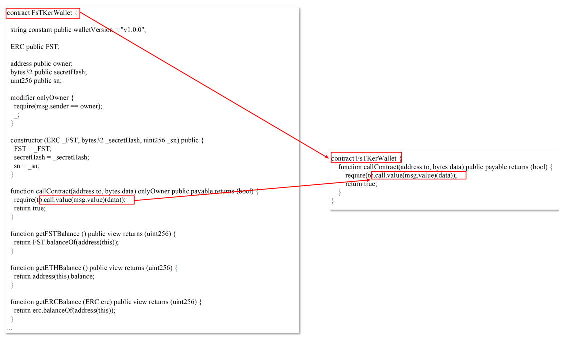
A. Contract Preprocessing

The proposed approach preprocesses smart contract code through several steps as follows:

Step 1: Redundancy Removal - In the initial stage of data preprocessing, it is essential to remove code comments and compress spaces, as vulnerabilities are not directly related to comments and certain whitespace characters. This ensures that the subsequent feature extraction process focuses on the code logic itself.

Step 2: Code Segment Extraction and Type Matching - Based on the characteristics of corresponding vulnerabilities, function segments that could trigger these vulnerabilities are automatically extracted as model input samples through control flow analysis. Taking reentrancy vulnerabilities as an example, since reentrancy vulnerabilities involve call.value calls that can recursively invoke themselves through a call chain, they are typically found within functions containing call.value calls. In this paper, when detecting reentrancy vulnerabilities, we extract functions containing call.value calls, combine them with relevant definitions, and generate code segments as depicted in Figure 2. Subsequently, the original text is matched and labeled with corresponding categories: labeled as 0 if it corresponds to a vulnerability, and labeled as 1 if it does not.

Step 3: WordPiece Tokenization -To ensure the consistency of input for subsequent feature extraction, this approach introduces a tokenizer called SCVulTokenizer to encode and transform code segments into parameters recognizable by neural networks. These parameters are then passed to BERT for feature extraction.



```
contract FsTKerWallet {

 string constant public walletVersion = "v1.0.0";

 ERC public FST;

 address public owner;
 bytes32 public secretHash;
 uint256 public sn;

 modifier onlyOwner {
   require(msg.sender == owner);
   _;
 }

 constructor (ERC _FST, bytes32 _secretHash, uint256 _sn) public {
   FST = _FST;
   secretHash = _secretHash;
   sn = _sn;
 }

 function callContract(address to, bytes data) onlyOwner public payable returns (bool) {
   require(to.call.value(msg.value)(data));
   return true;
 }

 function getFSTBalance () public view returns (uint256) {
   return FST.balanceOf(address(this));
 }

 function getETHBalance () public view returns (uint256) {
   return address(this).balance;
 }

 function getERCBalance (ERC erc) public view returns (uint256) {
   return erc.balanceOf(address(this));
 }
...
```

```
contract FsTKerWallet {
   function callContract(address to, bytes data) public payable returns (bool) {
      require(to.call.value(msg.value)(data));
      return true;
   }
}
```

Fig. 2. Code extraction sample

SCVulTokenizer is designed to address the intricacies and uniqueness of smart contract code. It combines the WordPiece [22] tokenization algorithm with keyword tokenization restrictions to meet the requirements of BERT for accurate tokenization and semantic consistency, ultimately delivering more precise and fine-grained tokenization results.

Specifically, SCVulTokenizer employs the WordPiece tokenization algorithm to segment smart contract code at the character level, further refining it into smaller subword sequences. In comparison to traditional character-level tokenization, SCVulTokenizer can offer more accurate and comprehensive tokenization results. Additionally, this paper conducted tests on the built-in keywords of the Solidity language, identifying 54 keywords that the original tokenization algorithm would subdivide. SCVulTokenizer was employed to enforce keyword tokenization restrictions, ensuring that these keywords remain as single-word units without further subdivision. This safeguard ensures the integrity and accuracy of keywords, allowing the BERT network to intuitively understand and recognize key terms within smart contract code without the need for additional semantic inference.

B. BERT-Based Contract Feature Extraction

Since the preprocessed segments of smart contract code cannot be understood by computers, it is necessary to perform feature extraction to convert the semantic information contained in the code into corresponding feature vectors, which will serve as inputs for the classification model. Typically, smart contract feature extraction mostly utilizes the Word2Vec model as the feature extraction network. However, since the Word2Vec model cannot differentiate feature words based on context semantics when representing word vectors, it is common in practice to concatenate the Word2Vec model with RNN (Recurrent Neural Network), LSTM, BiLSTM (Bi-directional Long Short-Term Memory), or similar models. While the above-mentioned approach can address the application of deep learning in contract detection, it still leaves significant room for improvement in feature extraction performance. This is because recurrent neural networks and similar techniques inherently suffer from memory loss issues, and Word2Vec is limited in addressing polysemy effectively.

To address these challenges, this paper proposes a contract feature extraction strategy based on the analysis of the inherent associations between the BERT network and smart contract code. This strategy consists of two main components: embedding feature extraction and multi-head self-attention feature extraction.

a) Embedding Feature Extraction: The purpose of this step is to extract the intrinsic information of each word in the tokenized contract, along with contextual information between words, and convert them into corresponding vector representations. Specifically, the preprocessed word sequence is transformed into three sets of embedding vectors through an embedding layer:$(E_{[cls]}, E_{contract}, E_{C1}, E_{\{,...,E\}}, E_{[SEP]})$ $(\mathrm{E}_A, E_A, E_A, ..., E_A, E_A)$ $(\mathrm{E}_0, E_1, E_2, ..., E_k, E_{k+1},)$. These three sets of word vectors correspond to the word features, segment features, and positional features in the word sequence, respectively. These features encompass essential information about the smart contract code, such as semantics (word features), code structure (segment features), and the positional information of words within the contract (position features). As illustrated in Figure 3, these features are combined through addition to form a comprehensive composite embedding feature vector,$(E_{[cls]}, E_{contract}, E_{C1}, E_{\{,...,E\}}, E_{[SEP]})$. This feature vector contains the static representation of each word in the smart contract code, independent of contextual information at specific positions, laying the foundation for subsequent context feature extraction at the code segment level.

b) Multi-Head Self-Attention Feature Extraction: Through the multi-head self-attention mechanism, the model can consider contextual information from all other words

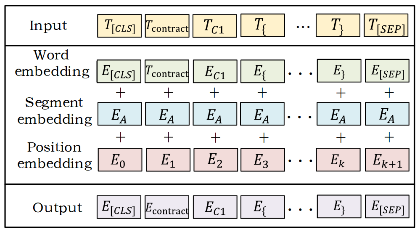

Input
$T_{[CLS]}$ $T_{contract}$ T_{C1} $T_{\{}$ ... $T_{\}}$ $T_{[SEP]}$
Word embedding
$E_{[CLS]}$ $T_{contract}$ E_{C1} $E_{\{}$ ... $E_{\}}$ $E_{[SEP]}$
Segment embedding
E_A E_A E_A E_A ... E_A E_A
Position embedding
E_0 E_1 E_2 E_3 ... E_k E_{k+1}
Output
$E_{[CLS]}$ $E_{contract}$ E_{C1} $E_{\{}$ ... $E_{\}}$ $E_{[SEP]}$

Fig. 3. BERT embedding layer

when comprehending the meaning of a particular word. This mechanism enables the model to capture contextual relationships within smart contract code and the significance of each word in its context.

Specifically, it utilizes model weights obtained through training, denoted as $W^Q W^K$, and W^V, to compute $Q\ K\ V$, where E represents the composite feature vector obtained from the previous embedding feature extraction step:

$$\begin{cases} Q = EW^Q \\ K = EW^K \\ V = EW^V \end{cases} \tag{1}$$

Plug the $Q\ K\ V$ into the following formula to calculate the self-attention vectors $head$, where d_k represents the number of columns in the vector $Q\ K\ V$, and Softmax is the activation function:

$$Attention(Q,K,V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2}$$

The calculation process is illustrated in Figure 4.

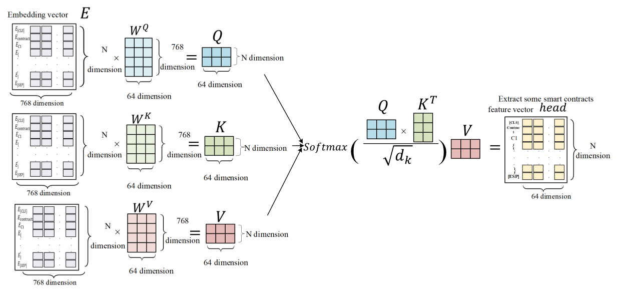


Fig. 4. Example of self-attention computation

Next, parallelize the self-attention calculations 12 times, obtaining the corresponding attention scores . Subsequently, perform concatenation and projection calculations using the following formula to derive the result of one multi-head self-attention calculation. Here, $W_i^Q\ K_i^K\ W_i^V$ represents the weights of the fully connected neural network trained in the i-th self-attention operation:

$$MultiHead\left(Q,K,V\right) = Concat\left(head_1, head_2, \cdots, head_h\right)W^o \tag{3}$$

$$head_i = Attention\left(E{W_i}^Q, E{K_i}^K, E{W_i}^V\right) \tag{4}$$

Finally, sequentially repeat the multi-head self-attention calculation 12 times (corresponding to 12 serially connected Transformer encoding blocks [24]), with the result of the last multi-head self-attention calculation serving as the feature vector for the smart contract. By performing the multi-head self-attention mechanism multiple times as described above, the BERT network conducts multidimensional dimensionality reduction and augmentation operations on the contract code at the token level to achieve a more precise contextual understanding. Ultimately, it extracts high-level semantics from the entire contract segment and represents it as a feature vector for output, serving its purpose in subsequent vulnerability detection.

C. Smart Contract Vulnerability Detection

After the feature extraction process in Step 2.2, the smart contract word sequence is transformed into character-level feature vectors. Each character in the sequence is associated with a corresponding feature vector of dimension 768. Since this approach utilizes the overall semantic features of smart contracts, the extracted feature vectors correspond to the first character, i.e., the $[CLS]$ character's feature vector.

Next, the $[CLS]$ character's feature vector is fed into a fully connected layer with a dimension of 768 and an output dimension of 2. The resulting feature vector of dimension 2 is then directed into a Softmax regression model for normalization. The Softmax calculation is performed as follows, where z represents the feature vector:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{2} e^{z_j}}\ for\ i = 1,2\ and\ z = (z_{\mathbf{1}}, z_{\mathbf{2}}) \in R^2 \tag{5}$$

The model's output values $\sigma(z)_i$ obtained from the above equation represent the corresponding probabilities that the model predicts for positive and negative samples. For example, if the output result is $[a,b]$, the probability of being predicted as a negative sample is denoted as a, while the probability of being predicted as a positive sample is denoted as b. Ultimately, the model provides a numerical label as the output, corresponding to the label with the higher predicted probability. This label serves as the result of SCVulBERT's detection for that particular sample.

D. Contract Transfer Learning

In the context of neural network-based approaches, the training process typically involves the use of labeled samples to train the neural network. During this process, the neural network compares its detection results with the actual labels, computes gradients with respect to the loss function, and then utilizes these gradients for backpropagation to adjust the neuron parameters. This iterative process ultimately results in a neural network capable of performing the detection task. In this scheme, considering the scarcity of labeled samples in the field of smart contract vulnerability detection, we introduce transfer learning alongside the conventional training process.

Transfer learning is a learning strategy that leverages pre-training on unlabeled datasets from other domains, enabling the model to acquire a wealth of prior knowledge. This accumulated knowledge can then be transferred and applied to new tasks, enhancing the model's learning efficiency and performance in these novel tasks [25]. In the context of smart contract vulnerability detection, the limited availability of labeled samples can often lead to underperforming models when trained directly. However, transfer learning offers an efficient solution to this challenge, as it provides a way to leverage knowledge gained from other domains.

Specifically, before training our model for smart contract detection, we initialize SCVulBERT using the pre-trained BERT-base model by Google on large-scale corpora, including the "BooksCorpus" (comprising around 800 million words) and "English Wikipedia" (around 2.5 billion words). This base model has undergone unsupervised pre-training on these extensive text corpora, enabling it to acquire rich language representations and comprehend contextual structures and semantic information in language. By initializing SCVulBERT with these pre-trained model weights, our model inherits this valuable prior knowledge.

Subsequently, we conduct supervised training for smart contract vulnerability detection on top of this base model. During this process, the model needs to gain a deep understanding of contract characteristics and vulnerability features. Despite the scarcity of labeled samples in the smart contract domain, the pre-trained model weights serve as prior knowledge that aids the model in better comprehending and learning the features of smart contract vulnerabilities. Therefore, even in the presence of limited labeled samples, our model can efficiently perform smart contract vulnerability detection.

Through this supervised transfer learning strategy, we further enhance the accuracy of smart contract vulnerability detection in scenarios where labeled samples are scarce.

Experimental and Analysis

E. Experimental Environment

In terms of the experimental environment, the operating conditions for SCVulBERT-related experiments are detailed in TABLE I.

TABLE I
SCVulBERT runtime environment

Category	SCVulBERT
Neural Network Framework	Pytorch
System	Ubuntu22.04 LTS
Processor	Intel Core i5-12400 CPU
Acceleration unit	NVIDIA Tesla P40 24G
Memory capacity	16 GB RAM

In terms of the dataset, for the convenience of future comparisons and replications of the experiments conducted in this study, we utilized a recently released public dataset known as Smart-Contract-Dataset. Specifically, we selected subsets from the Resource2 dataset within this dataset, focusing on reentrancy vulnerabilities, timestamp vulnerabilities, and delegate call vulnerabilities. The reentrancy vulnerability subset comprises 273 samples, with 73 (26.74%) labeled as positive samples and 200 (73.26%) as negative samples. The timestamp vulnerability subset contains 349 samples, with 179 (51.29%) positive samples and 170 (48.71%) negative samples. Additionally, the delegate call vulnerability subset includes 196 samples, with 62 (31.63%) positive samples and 134 (68.37%) negative samples. It's important to note that the reason for selecting these specific subsets is their prevalence and representativeness among common smart contract vulnerabilities. Furthermore, the use of a limited number of labeled samples allows for the evaluation of the model's effectiveness in scenarios where labeled samples are scarce. Additionally, this dataset has already undergone preprocessing, with vulnerability-related code fragments extracted through automated control flow analysis and labeled with their respective vulnerability types. For readers interested in conducting similar preprocessing for other datasets, reference [26] can provide further guidance, though we won't delve into extensive details here.

Prior to conducting experiments, we divided each subset of the dataset into training and validation sets using an 8:2 ratio. We ensured that the balance between positive and negative samples was maintained in both sets, mitigating the issue of sample proportion imbalance that could arise from random partitioning.

In terms of evaluation metrics, the field of vulnerability detection often employs widely recognized performance metrics such as Accuracy, Precision, Recall, and F1-score [28] for assessing model performance. In this study, we also utilized these four-evaluation metrics to assess the model's performance.

In terms of experimental parameters, this approach employed a cross-validation methodology to assess the impact of various hyperparameters on the results. By comparing the outcomes associated with different hyperparameters, we aimed to identify the optimal configuration. The ultimate experimental parameters are detailed in TABLE III.

TABLE II
Selection of experimental parameters

Parameter	Meaning	Value	
Learning Rate	Initial learning rate	0.00001	
Batch Size	Sample training volume for each batch	Timestamp vulnerability	16
		Three other vulnerabilities	4
Epoch	Training rounds	100	
Optimizer	Learning rate optimization algorithm	AdamW	
Dropout	Random inactivation rate	0.1	
Embed Size	Word vector dimension	768	

F. Performance Comparison and Analysis

To further validate the performance of our model, we conducted comparative experiments with two representative neural network approaches: ReChecker (a solution based on Word2Vec and LSTM, BiLSTM-ATT [29]) and DR-GCN(TMP). The results are presented in TABLE IV. Analyzing TABLE IV, it is evident that SCVulBERT outperforms other models significantly in terms of accuracy for all vulnerability types. Particularly in the detection of reentrant vulnerabilities, SCVulBERT achieves an accuracy of 98.18%, far surpassing other models, demonstrating its superior ability to recognize and classify vulnerability types more accurately, especially reentrant vulnerabilities. Furthermore, in terms of precision, SCVulBERT exhibits precision exceeding 90% for all vulnerability types, indicating superior performance. Notably, in the detection of delegate call vulnerabilities, SCVulBERT achieves 100% precision, showcasing its outstanding capability in reducing false positives. Lastly, SCVulBERT also demonstrates a notable advantage in recall and F1 scores. Across all vulnerability types, SCVulBERT achieves higher recall and F1 scores, indicating excellent real vulnerability detection capability and the ability to balance precision and recall effectively.

Additionally, we observed a significant improvement in the recall value of SCVulBERT in the detection of reentrant vulnerabilities, with an increase of up to 40.67% compared to other comparative solutions. Through analysis, we attribute this improvement to the scarcity of positive samples for reentrant vulnerabilities in the dataset and the absence of prior knowledge transfer in other deep learning approaches. As a result, these approaches fail to generalize and recognize other vulnerability expressions effectively, leading to issues of underreporting. In contrast, SCVulBERT, with the benefit of transfer learning and prior knowledge, efficiently achieves stronger generalization and recognition with very few supervised samples, thus effectively identifying reentrant vulnerabilities. In summary, SCVulBERT outperforms other neural network models in various evaluation metrics, highlighting its outstanding performance in vulnerability detection tasks.

Furthermore, we compared our proposed solution with traditional vulnerability detection tools. The traditional tools used for comparison include Slither, Mythril, and Oyente, and the comparative results are shown in TABLE V. Notably, Slither does not support integer overflow vulnerabilities, and Oyente does not support delegate call vulnerabilities. From TABLE V, it is evident that compared to traditional detection solutions, the SCVulBERT model exhibits superior performance in terms of accuracy, precision, recall, and F1 scores. In the detection of reentrant vulnerabilities, SCVulBERT achieves an accuracy of up to 98.18%, significantly surpassing Slither, Mythril, and Oyente. Furthermore, SCVulBERT's performance in terms of precision and recall is also notably superior to the other three tools, especially in precision, where SCVulBERT reaches 93.33%, far exceeding Slither's 72.91%. In the detection of timestamp vulnerabilities and delegate call vulnerabilities, SCVulBERT's accuracy and precision also significantly outperform the other three tools. Notably, SCVulBERT achieves a precision of 97.22% in timestamp vulnerability detection and a perfect 100% precision in delegate call vulnerability detection. Additionally, in terms of F1 scores, SCVulBERT maintains scores above 90% for all vulnerability types, which is significantly superior to traditional tools. This indicates that SCVulBERT not only accurately detects vulnerabilities but also strikes a good balance between reducing false positives and false negatives.

Moreover, in the case of reentrant vulnerabilities, SCVulBERT exhibits a significant improvement in recall compared to other deep learning solutions, with an increase of up to 40.67% compared to the best-performing alternative solutions. Through analysis, it is concluded that the scarcity of positive samples for reentrant vulnerabilities in the subset of the dataset and the absence of prior knowledge transfer in other deep learning approaches contribute to their inability to generalize and effectively recognize other vulnerability expressions, leading to underreporting issues. Conversely, SCVulBERT leverages transfer learning to acquire substantial prior knowledge, enabling efficient generalization and recognition even with minimal positive samples, effectively identifying reentrant vulnerabilities.

In summary, the superiority of SCVulBERT in vulnerability detection tasks compared to traditional tools can be attributed to its utilization of the BERT network, which captures deep semantic and contextual information, thereby enhancing the model's recognition capabilities. Additionally, the model's adoption of a transfer learning strategy, leveraging initialization training on large-scale pre-trained datasets, further enhances its generalization capabilities. Through these comparative experiments, it is evident that the SCVulBERT model excels in improving the accuracy of smart contract vulnerability detection, reducing false positives and false negatives, and opens up new possibilities for efficient and precise vulnerability detection in the future.

G. Ablation Experiments

In order to further investigate the effectiveness of transfer learning and the SCVulTokenizer in enhancing SCVulBERT, this section conducts ablation experiments to validate the effectiveness of transfer learning and the SCVulTokenizer.

In the experimental design phase, we kept the model parameters and dataset consistent with Section B. In this study, the final SCVulBERT model is compared with SCVulBERT-GPT2Tokenizer and SCVulBERT-scratch. SCVulBERT-GPT2Tokenizer represents a version of SCVulBERT in which SCVulTokenizer is replaced with GPT2Tokenizer. On the other hand, SCVulBERT-scratch is a version that starts training from random weights as the initialization point, without pretraining on a large-scale English corpus, following the training approach consistent with other deep learning solutions. The experimental results are shown in TABLE VI. Firstly, by comparing the performance of SCVulBERT with SCVulBERT-scratch, the effectiveness of transfer learning

TABLE III
DETECTION RESULTS OF VULNERABILITIES IN DIFFERENT DEEP LEARNING SCHEMES

loophole Type	Evaluate Index	LSTM	BiLSTM	BiLSTM-ATT	DR-GCN (TMP)	SCVulBERT (Model in this article)
Refers to a reentrant Vulnerability	Accuracy	84.72%	82.00%	82.36%	80.00%	98.18%
	Precision	81.39%	76.11%	80.13%	82.00%	93.33%
	Recall	59.33%	50.66%	48.00%	34.66%	100.00%
	F1 value	66.97%	60.00%	56.98%	48.45%	96.55%
Timestamp Vulnerability	Accuracy	71.57%	77.00%	71.14%	77.28%	90.00%
	Precision	74.51%	78.11%	73.59%	78.40%	97.22%
	Recall	68.61%	77.50%	68.61%	77.22%	85.37%
	F1 value	70.76%	77.59%	70.88%	77.76%	90.91%
Delegate Call Vulnerability	Accuracy	85.25%	90.75%	84.75%	77.75%	95.00%
	Precision	88.52%	88.63%	92.83%	68.74%	100.00%
	Recall	64.61%	82.30%	58.46%	59.23%	86.67%
	F1value	73.52%	85.16%	70.17%	62.86%	92.86%

TABLE IV
COMPARISON OF VULNERABILITY DETECTION RESULTS OF TRADITIONAL TOOLS

loophole Type	Evaluate Index	Slither	Mythril	Oyente	SCVulBERT (Model in this article)
Refers to a reentrant Vulnerability	Accuracy	89.37%	62.63%	66.30%	98.18%
	Precision	72.91%	38.21%	42.85%	93.33%
	Recall	95.89%	64.38%	78.08%	100.00%
	F1value	82.84%	47.95%	55.33%	96.55%
Timestamp Vulnerability	Accuracy	46.99%	47.85%	44.41%	90.00%
	Precision	48.68%	48.57%	5.88%	97.22%
	Recall	62.01%	28.49%	0.55%	85.37%
	F1value	54.54%	35.91%	1.02%	90.91%
Delegate call Vulnerability	Accuracy	41.32%	65.81%	-	95.00%
	Precision	31.46%	33.33%	-	100.00%
	Recall	72.58%	8.06%	-	86.67%
	F1value	43.90%	12.98%	-	92.86%

TABLE V
COMPARISON RESULTS OF ABLATION EXPERIMENTS

loophole Type	Evaluate Index	SCVul BERT	SCVulBERT-GPT2Tokenizer	SCVulBERT (Model in this article)
Refers to a reentrant Vulnerability	Accuracy	74.55%	96.36%	98.18%
	Precision	6.67%	93.33%	93.33%
	Recall	100.00%	93.33%	100.00%
	F1value	12.50%	93.33%	96.55%
Timestamp Vulnerability	Accuracy	84.29%	88.57%	90.00%
	Precision	83.33%	83.33%	97.22%
	Recall	85.71%	93.75%	85.37%
	F1value	84.51%	88.24%	90.91%
Delegate call Vulnerability	Accuracy	87.50%	92.50%	95.00%
	Precision	84.62%	100.00%	100.00%
	Recall	78.57%	81.25%	86.67%
	F1value	81.48%	89.65%	92.86%

in this study is emphasized. SCVulBERT-scratch exhibits significantly lower accuracy, precision, recall, and F1 scores across all vulnerability types compared to SCVulBERT. This suggests that initializing the model with weights pretrained on a large-scale English corpus is more effective in enhancing model performance than starting with random weights as the initialization point.

Secondly, the importance of SCVulTokenizer is evident in the comparison between SCVulBERT and SCVulBERT-GPT2Tokenizer. It can be observed that the SCVulBERT model using GPT2Tokenizer performs worse in all evaluation metrics compared to the model using SCVulTokenizer. This indicates that SCVulTokenizer is more effective in handling the SCVulBERT model, outperforming GPT2Tokenizer. This is because SCVulTokenizer is better suited for specific tasks, enabling a better understanding and parsing of domain-specific data. This finding further underscores the importance of custom tokenizers for specific tasks, as they can provide finer vocabulary segmentation and better handle domain-specific syntax and vocabulary, thereby improving model performance.

In conclusion, the results of the ablation experiments validate the crucial roles of transfer learning and SCVulTokenizer in enhancing the SCVulBERT model. Additionally, these findings provide insights into how to better utilize deep learning models in specific domains, which will contribute to more effective improvements of the proposed approach in the future.

Conclusion and Future Outlook

This paper addresses the shortcomings of current smart contract vulnerability detection solutions and proposes an efficient detection scheme called SCVulBERT based on BERT. This scheme involves preprocessing smart contract code and utilizes the proposed SCVulTokenizer for code tokenization. By leveraging the BERT feature extraction network, SCVulBERT achieves more efficient and accurate vulnerability detection. The proposed approach can accurately identify the four most prominent contract vulnerabilities and outperforms current state-of-the-art solutions. Specifically, it improves F1 scores for reentrancy, timestamp, delegate call, and integer overflow vulnerabilities by 11.38%, 12.48%, 4.84%, and 7.11%, respectively. The model exhibits significant performance advantages across all metrics. However, due to computational resource limitations and model complexity, code snippet extraction is required before feature extraction. Additionally, while employing transfer learning to mitigate the challenge of limited labeled samples, performance remains constrained. Therefore, future work will focus on further enhancing vulnerability detection performance with limited labeled samples and improving the generalizability of the solution. While SCVulBERT demonstrates superior performance in smart contract vulnerability detection, it is important to note that the model's complexity and use of pre-trained weights necessitate higher computational resources and memory. Therefore, a trade-off between model performance and computational efficiency should be considered when opting for this approach. Future work will also focus on optimizing the model for better computational efficiency without compromising its performance.

References

[1] Szabo N. Smart contracts: building blocks for digital markets[J]. EXTROPY: The Journal of Transhumanist Thought,(16), 1996, 18(2): 28.

[2] Santoso N, Javaid H. Improving Energy Efficiency of Permissioned Blockchains Using FPGAs[C]//2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2023: 177-184.

[3] Hu Tianyuan, Li Zecheng, Li Bixin, et al. A Survey of Contract Security and Privacy Security for Smart Contracts[J]. Journal of Computer Science, 2021, 44(12): 2485-2514.

[4] Bai Xiang, Xu Congfang, Liu Xing, et al. A Review of Security Technologies and Key Technologies Analysis for Blockchain IoT[J]. Information Technology, 2022, No.371(10): 24-30+40. DOI:10.13274/j.cnki.hdzj.2022.10.005.

[5] Yang Zhongju, Zhu Weixing, Shi Yaqing, et al. A Survey of Smart Contract Vulnerabilities and Detection Technologies[J]. Network Security Technology and Applications, 2022, No.263(11): 6-9..

[6] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv, 2018, 1810-04805.

[7] Tao M, Feng Y, Zhao D. Can BERT Refrain from Forgetting on Sequential Tasks? A Probing Study[J]. arXiv preprint arXiv:2303.01081, 2023.

[8] Dan Dan, Duan, Jiashan Tang, Yong Wen, et al. Chinese Short Text Classification Algorithm Based on BERT Model[J]. Computer Engineering, 2021, 47(01): 79-86. DOI: 10.19678/j.issn.1000-3428

[9] Amani S, Bégel M, Bortin M, et al. Towards verifying ethereum smart contract bytecode in Isabelle/HOL[C]//Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2018: 66-77.

[10] Kalra S, Goel S, Dhawan M, et al. Zeus: analyzing safety of smart contracts[C]//Ndss. 2018: 1-12.

[11] Luu L, Chu D H, Olickel H, et al. Making smart contracts smarter[C]//Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016: 254-269.

[12] Mythril. A framework for bug hunting on the Ethereum blockchain. [EB/OL].[2017-07-11].https://mythx.io/

[13] Yao P, Zhou J, Xiao X, et al. Efficient path-sensitive data-dependence analysis[J]. arXiv preprint arXiv:2109.07923, 2021.

[14] Liu C, Liu H, Cao Z, et al. Reguard: finding reentrancy bugs in smart contracts[C]//Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. 2018: 65-68.

[15] He J, Balunović M, Ambroladze N, et al. Learning to fuzz from symbolic execution with application to smart contracts[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 531-548.

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

[17] Tann W J W, Han X J, Gupta S S, et al. Towards safer smart contracts: A sequence learning approach to detecting security threats[J]. arXiv preprint arXiv, 2018, 1811-06632.

[18] Wang Junnian, Zhu Bin, Yu Wenxin, et al. Side-Channel Analysis Based on Deep Learning LSTM[J]. Computer Engineering, 2021, 47(10): 140-146. DOI: 10.19678/j.issn.1000-3428.

[19] Qian P, Liu Z, He Q, et al. Towards automated reentrancy detection for smart contracts based on sequential models[J]. IEEE Access, 2020, 8: 19685-19695.

[20] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv, 2013, 1301-3781.

[21] Wang Z, Guo S, Liu G. SA-DDQN: Self-Attention Mechanism Based DDQN for SFC Deployment in NFV/MEC-Enabled Networks[C]//2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2023: 720-727.

[22] Wu Y, Schuster M, Chen Z, et al. Google's neural machine translation system: Bridging the gap between human and machine translation[J]. arXiv preprint arXiv, 2016, 1609-08144.

[23] Zaremba W, Sutskever I, Vinyals O. Recurrent neural network regularization[J]. arXiv preprint arXiv, 2014, 1409-2329.

[24] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[25] Wang Tianran, Wang Qi, Wang Qingshan. Cross-Object Sign Language Gesture Recognition Method Based on Transfer Learning[J]. Computer Science, 2023, 50(S1): 129-133.

[26] Google Research: bert-base-cased model weights. https://huggingface.co/bert-base-cased, date of publication not available, Accessed: 2023-1-2

[27] Qian, P.: Smart contract dataset (resource2). https://github.com/Messi-Q/SmartContract-Dataset (2022), (Accessed: 2022-12-28)

[28] Zhang Xiaosong, Niu Weina, Huang Shiping, et al. Overview of Smart Contract Vulnerability Detection Methods Based on Deep Learning[J]. Journal of Sichuan University (Natural Science Edition), 2023, 60(02): 7-18. DOI: 10.19907/j.0490-6756.2023.020001.10.19907/j.

[29] Zhou P, Shi W, Tian J, et al. Attention-based bidirectional long short-term memory networks for relation classification[C]//Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers). 2016: 207-212.

[30] Tompkins D, Kumar K, Wu J. Maximizing audio event detection model performance on small datasets through knowledge transfer, data augmentation, and pretraining: an ablation study[C]//ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2022: 1016-1020.

[31] Zhuang Y, Liu Z, Qian P, et al. Smart contract vulnerability detection using graph neural networks[C]//Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. 2021: 3283-3290.