Computer Networks 217 (2022) 109289

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Smart contract vulnerability detection combined with multi-objective detection

Lejun Zhang [a,b,c,*], Jinlong Wang [b], Weizheng Wang [d], Zilong Jin [e], Yansen Su [f], Huiling Chen [g,**]

[a] Cyberspace Institute Advanced Technology, Guangzhou University, Guangzhou, 510006, China
[b] College of Information Engineering, Yangzhou University, Yangzhou, 225127, China
[c] Research and Development Center for E-Learning, Ministry of Education, Beijing, 100039, China
[d] Computer Science Department, City University of Hong Kong, Hong Kong
[e] School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, 21004, China
[f] Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, School of Computer Science and Technology, Anhui University, Hefei 230601, China
[g] Department of Computer Science and Artificial Intelligence, Wenzhou University, Wenzhou 325035, China

ARTICLE INFO

Keywords:
Blockchain
Smart contract
Vulnerability detection
Machine learning
Multi-objective

ABSTRACT

Blockchains have been booming in recent years. As a decentralized system architecture, smart contracts give blockchains a user-defined logic. A smart contract is an executable program that can automatically carry out transactions on the Ethereum blockchain. However, some security issues in smart contracts are difficult to fix, and smart contracts also lack quality assessment standards. Therefore, this study proposes a Multiple-Objective Detection Neural Network (MODNN), a more scalable smart contract vulnerability detection tool. MODNN can validate 12 types of vulnerabilities, including 10 recognized threats, and identify more unknown types without the need for specialist or predefined knowledge through implicit features and Multi-Objective detection (MOD) algorithms. It supports the parallel detection of multiple vulnerabilities and has high scalability, eliminating the need to train separate models for each type of vulnerability and reducing significant time and labor costs. This paper also developed a data processing tool called Smart Contract-Crawler (SCC) to address the lack of smart contract vulnerability datasets. MODNN was evaluated using more than 18,000 smart contracts from Ethereum. Experiments showed that MODNN could achieve an average F1 Score of 94.8%, the current highest compared to several standard machine learning (ML) classification models.

1. Introduction

The smart contract is a very early concept first proposed by the American scholar Nick Szabo and is a computer protocol designed to disseminate, validate, or execute contracts in an informational way [1]. Because smart contracts had not yet been practically applied, they had not garnered mainstream attention. It was not until smart contracts were combined with blockchain technology that they became widely used. The Blockchain 1.0 era is represented by Bitcoin, while Blockchain 2.0 is the era of smart contracts represented by Ethereum [2], the most popular platform providing an execution environment for smart contracts today [3], and many decentralized applications are deployed to Ethereum in the form of smart contracts.

Motivation: However, a significant and common issue with smart contracts at this stage is their security, which cannot be guaranteed because of the (lack of) complexity of the programming languages used to create smart contracts [4]. Since every user can join the blockchain, which is not governed by a credible third party, and smart contracts often control vast amounts of financial assets, smart contracts are easy targets of cyberattacks [5]. Unlike traditional programs, no one can modify smart contracts after they are deployed. This offers the advantage of anti-tampering, but it also poses a considerable security risk. Smart contracts are code manually written by nature, which means that they are inevitably subject to errors or loopholes. A large number of real-world events have proven the seriousness of this issue; for example, a recursive vulnerability in the code of the Decentralized Autonomous Organization (DAO) [6] was used to steal 3.6 million Ether. Table 1 summarizes the attack platforms and losses that occurred during the two months when this article was being written. Contracts must be checked rigorously and thoroughly and made robust and leak-free before they are deployed. This paper aims to design a novel and

* Corresponding author at: College of Information Engineering, Yangzhou University, Yangzhou, 225127, China.
** Corresponding author.
E-mail addresses: zhanglejun@yzu.edu.cn (L. Zhang), chenhuiling_jsj@wzu.edu.cn (H. Chen).

https://doi.org/10.1016/j.comnet.2022.109289
Received 7 May 2022; Received in revised form 24 June 2022; Accepted 11 August 2022
Available online 24 August 2022

Table 1
Attack platforms and losses in two months.

Platform	Loss ($)
Grim Finance	30 million
Wault Finance	930,000
Popsicle Finance	21 million
Zabu Finance	600,000
Cream Finance	18 million
MonoX Finance	31 million
Cream Finance	115 million

efficient method for detecting vulnerabilities in smart contracts and predicting unknown types of vulnerabilities.

Many previous works have studied vulnerability detection in smart contracts, such as through symbolic execution [7] and data flow analysis [8]. The symbolic execution-based approach needs to check all execution paths, which are affected by the problem of path explosion. Data flow analysis does not guarantee complete coverage of the program [9]. Moreover, they do not perform well on specific error patterns or rules defined by human experts and construct new checkers for new types of errors and vulnerabilities requiring significant time and labor costs. Therefore, this paper uses ML to automate learning and address the above problems.

Primary objective: ML has been used for cloning and identifying vulnerabilities in traditional software programs, but its application to smart contracts is still limited. This paper proposes MODNN, a MOD network. Not only can it use existing features to identify vulnerabilities in known patterns, but it can also learn vulnerabilities in unknown patterns from extended features, thus increasing the scalability of the model and reducing the time needed to train the model. The MODNN is based on the crucial operation sequence (COS), which can analyze any contracts that can be converted to opcode, so it has a wide range of applications. To facilitate the healthy development of blockchain systems, more MODNN materials will be uploaded to [10].

Contribution: The main contributions of this paper can be summarized as follows:

(1) The notion of the COS was proposed, a tool was built to extract the COS, the input for the neural network (NN) was optimized, and the model's performance was improved.

(2) A novel method of implicit feature extraction of the operation sequence based on the co-occurrence matrix was designed and tested on a MODNN, which proved the method's effectiveness.

(3) A MOD model with backpropagation was designed, and multiple loopholes were output in parallel in the form of probability. Multiple vulnerability objects were reported with one detection.

(4) A vulnerability detection model based on MOD was designed, and the model can be extended at a lower cost when new vulnerabilities are identified.

(5) Many tests and evaluations of MODNN on real-world contracts have been conducted. The effectiveness of this approach was demonstrated through a comparison with state-of-the-art vulnerability detection models.

The rest parts are organized as follows. Section 2 reviews the ML-based approaches for detecting smart contract vulnerabilities and discusses their limitations. Section 3 presents the details of the proposed model and its technical difficulties. Section 4 briefly describes the data collection approach. Section 5 introduces the system implementation in detail, including data processing and MOD. Section 6 analyzes the performance of the MODNN by comparing it to other models. Section 7 concludes the paper and explains future research directions.

2. Related work

Many kinds of research on smart contract security verification have been conducted. This section discusses research on smart contract vulnerability detection.

2.1. ML

With the progress in ML, security researchers have achieved a breakthrough in the detection of smart contract vulnerabilities. The works [11–13] proposed methods for detecting contract vulnerabilities using a convolutional neural network (CNN). The vulnerabilities detected are transformed into the classification of images by constructing semantic graphs or converting bytecodes into digital images according to predefined rules based on contract syntax and semantic information.

As natural language processing (NLP) is becoming more and more sophisticated, many scholars have started to consider program statements as ordinary sentences in human language and to retain as much as possible the original flow of information in the code. The work [14] used the Bert model to process the contract source code, a strategy that is limited by the embedding length and does not consider information such as semantics. The works [15,16] used the improved Bert, GraphCodeBERT, to process code. Compared to Bert, GraphCodeBERT retains more original information about the code and is more suitable for code representation. In addition, works [17–19] used sentence similarity as a starting point for word embedding with the Word2Vec model, and then the classifier was trained on the embedding results to analyze vulnerabilities. Most of these methods used unsupervised or semi-supervised training, however, Hill et al. [20] showed that supervised learning can yield better results in NLP.

The work [21] demonstrated that the percentage of smart contract code clones was significantly higher than that in general software. The works [22,23] used long short-term memory (LSTM) models to deal with contractual vulnerabilities. While both achieved relatively high detection performance, they ignored the impact of local code on the overall code and demonstrated inconsistent detection performance for multiple classes of vulnerabilities. The work [24] extracted graph features from the simplified operational code of a smart contract, and the results were represented by abstract zeros and ones. The work [25] used an unsupervised graph-embedding approach to trace the data and control flow by simulating bytecode execution.

2.2. MOD

In recent years, the Multi-Objective algorithm has developed rapidly and achieved good results in many fields. Multi-Objective-based NN optimization is a classical application [26,27] that can effectively reduce the errors resulting from the dataset. Multi-Objectives are also used in face recognition scenarios due to their good sensitivity and specificity [28,29]. Multi-Objective also plays an important role in software security [30–33]. However, the use of MOD for smart contract loophole detection is still in its infancy. To the best of our knowledge, this study is the first attempt to apply MOD to smart contract loophole detection.

3. The proposed architecture

Previous ML-based vulnerability detection techniques have only investigated simple architectures, such as k-nearest neighbors (K-NNs), support vector machines (SVMs), and decision trees (DTs), to determine whether a program contains a vulnerability. In recent years, vulnerability detection methods based on transformer and ensemble learning, such as [34,35], have achieved great success. Unlike the above methods, MODNN builds a scalable vulnerability recognizer that can

(1) give the probability of specific vulnerabilities in the smart contract rather than using 0 or 1 as an absolute representation;

(2) use a single network to detect multiple vulnerabilities without multiple training for different vulnerabilities; and

(3) learn potential unknown vulnerabilities through implicit features to achieve the purpose of model scalability. To this end, this paper proposes a novel MOD structure that can be used to detect multiple types of vulnerabilities at once.

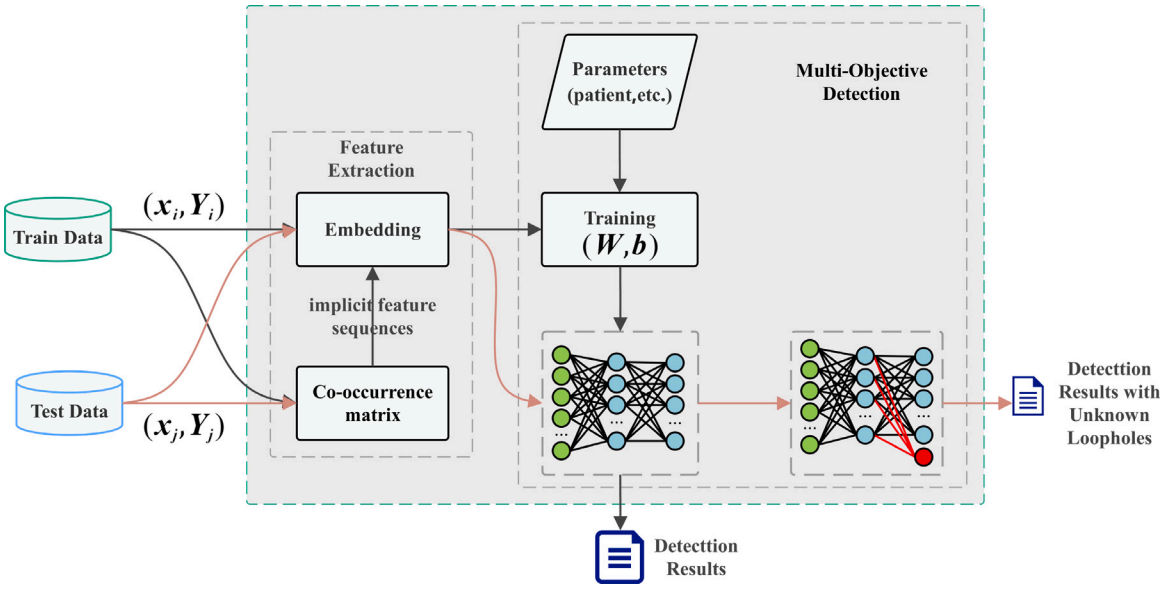


Fig. 1. Overall framework of the system.

3.1. Overview of the system model

Fig. 1 depicts the overall architecture of the proposed method that consists of two parts:

Feature Extraction. This paper is the first to use pre-trained Bert to convert the COS that can significantly improve the performance of smart contract vulnerability detection into explicit features and the first to use opcodes to construct co-occurrence matrices for learning implicit features.

MOD. This paper uses the features obtained in feature extraction to output the detection results, and the model will extend the output layer automatically when new types of vulnerabilities are detected. The two components and technological difficulties will be elaborated on in detail one by one.

3.2. Feature extraction

Previous works [36,37] have shown that the combination of Bert and CNN has demonstrated better classification performance. GraphCodeBert [16] also showed that ordinary programs can be transformed into a series of tokens that can be used to learn vulnerability characteristics. Recently, Bert [14] made considerable strides in the smart contract source code. Given these successes, this paper is the first to use Bert to embed the sequence of operations, which is used to transform a sequence of tokens into a vector to obtain training features. At the same time, these vector representations can be directly associated with vulnerability operations and can offer more meaningful information than the original token. However, the realistic problem is that the maximum length of a sequence embedded using Bert is 512, beyond which the sequence will be truncated. The original operation sequence of most contracts is much longer than this threshold. Suppose that the complete operation sequence is used for embedding. In that case, much of the information related to the vulnerability will be discarded, affecting the model's performance in practice.

However, this paper argues that smart contract vulnerabilities are only caused by part of the sequence of operations. When focusing on the whole sequence of operations of the contract, the extracted features will become inaccurate and will affect model performance. Therefore, for smart contracts in source form, this paper innovatively proposes an automated tool to obtain those sequences of operations (COS) that are most relevant to vulnerabilities, which is the subsequence that contains crucial information that may trigger vulnerabilities, in the original

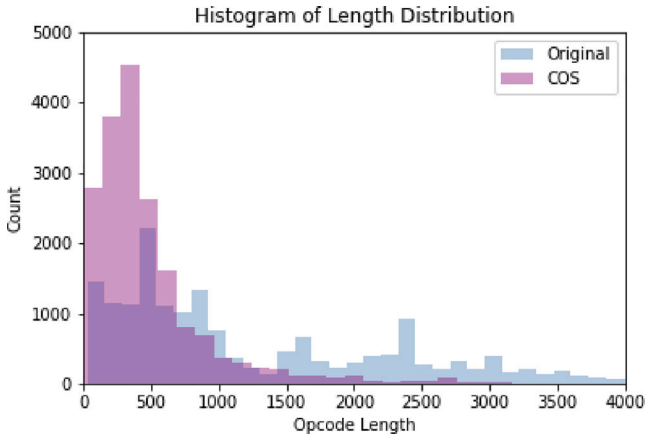


Fig. 2. Sequence length distribution comparison.

sequence and to focus the features on the COS, thus significantly reducing the dimensionality of the embedded data. If a smart contract in bytecode form $B = \{b_1, b_2, \ldots, b_m\}$ is given, this paper improves the location results of the static analysis tool so that it can return bytecode fragments with anomalous behavior directly, which can then be converted into COS. Fig. 2 shows the length distributions of the original sequence and the COS, where the average length of the original sequence is 1693.41, while the average length of the COS is only 505.09, a reduction of more than two-thirds.

Given the source code of a contract with vulnerabilities $SC = \{c_1, c_2, \ldots, c_p\}$, this article uses the smart contract patching tool SCRepair [38] to fix the contract. Because SCRepair [38] officially only supports Slither [9] and Oyente [39] vulnerability identifiers, this paper follows the analytical patterns proposed by Mythril [40], Smartcheck [41], and Osiris [42] to modify SCRepair [38] to facilitate more types of vulnerability repairs. For example, Lines 6 and 7 of the code snippet on the left in Fig. 3(a), which may incur a re-entry vulnerability, are patched and shown in the red rectangle of the code snippet on the right. Then, using the multiple sequence alignment (MSA) [43] algorithm to compare the patched bytecode to the original bytecode and filter out the differences as the COS, Fig. 3(b) shows the original and patched bytecode, respectively (the same parts have

Algorithm 1: COS Extraction Algorithm

Input: FileName $FN\{FN_1, \ldots, FN_\gamma\}$, Label $C\{C_1, \ldots, C_\gamma\}$, ByteCode $BC\{BC_1, \ldots, BC_\gamma\}$, OpCode $OC\{OC_1, \ldots, OC_\gamma\}$, StopWord $SW\{SW_1, \ldots, SW_\gamma\}$
Output: Operation Sequence-Label correspondence $< OS - C >$

```
for BC_i, C_i, i ← 1 to γ do
    if C_i = 1 then
        SC ← SCRePair(FN_i);
        BC_i' ← CompileFiles(SC);
        CS_i ← MSA(BC_i, BC_i');
        OC_i' ← Byte2Op(CS_i);
        for opcode in OC_i' do
            if opcode not in SW then
                add opcode into OS_i
            end
        end
        Generate an operation sequence-label correspondence
         < OS_i − C_i >
    end
    else if C_i = 0 then
        for opcode in OC_i' do
            if opcode not in SW then
                add opcode into OS_i
            end
        end
        Generate an operation sequence-label correspondence
         < OS_i − C_i >
    end
end
return {< OS_1 − C_1 >, ..., < OS_γ − C_γ >}
```

been omitted). The MSA [43] starts with the first discrepancy and ends with the last discrepancy. Algorithm 1 shows the detailed process of obtaining the COS. Where $CompileFiles()$ is responsible for converting the contract source code SC into the corresponding bytecode BC, $Byte2Op()$ converts every two bits of BC into the corresponding opcode OC.

3.3. Implicit features

Existing work [12] has converted contract bytecodes into matrices and then into grayscale images, an approach that has inspired this paper's implicit feature extraction and representation processes. To obtain the implicit features from the COS, this paper transforms them into co-occurrence matrices and identifies the implicit features using confidence and similarity scores.

Explicit features are defined as represented by the matrix after the COS embedding, and implicit features are defined as the embedding matrix representation of operations directly related to the vulnerability. The COS still contains sequences of operations not directly related to the vulnerability, such as "60 600060 25 5050 905 5b50" in Fig. 3(b). It is still part of the embedding in the explicit feature, i.e. "52 60 200190152 60 2 00 1 60 00206000828 25 40392 5050 81 905 550 5b50" as a whole word embedding. However, "52 2001908152 2 1 00206000828 40392 81 550" is often difficult to identify and embed directly as an implicit feature to be embedded that is directly related to the contractual vulnerability. In this paper, we cleverly construct a co-occurrence relationship matrix based on the co-occurrence counts of operands, and ranking based on the co-occurrence counts makes it easier to identify implicit features even if the members are not adjacent to each other. Section 5.1 provides a more specific description and demonstrates the implementation of implicit feature extraction and representation.

3.4. MOD

A smart contract may contain multiple vulnerabilities. An objective of this work is to increase model utilization and reduce the number of model training sessions by using a single model to detect multiple vulnerabilities simultaneously. In ML, a complex learning problem is first decomposed into theoretically independent subproblems, each subproblem is learned separately, and then a mathematical model of the complex problem is built by combining the learning results of the subproblems. Inspired by this idea, this paper divides vulnerability detection into multiple subtasks, each responsible for detecting a specific type of vulnerability. The branch responsible for each subtask is a combination of layers, all of which share the feature extraction results and are trained to learn the explicit or implicit features of the corresponding vulnerability class. Given a smart contract SC_i, MODNN uses explicit features to detect known types of vulnerabilities in parallel and outputs the corresponding probabilities, and predicts new types of vulnerabilities based on the implicit features identified (the branches of MODNN are not directly related to each other), and the final output is independent compared to the general partition. Section 5.2 provides a more specific description and the implementation for this section.

3.5. Technical difficulty

Assuming that an attacker can access the blockchain's contents from any public data structure and can distribute malicious contract code throughout it to ensure that no space is left for any malicious exploit after the smart contract is deployed, users can detect multiple vulnerabilities using MODNN before the contract is deployed. However, the following technical aspects must be overcome.

Data Collection and Preprocessing. Supervised learning of the MODNN requires a large-enough labeled dataset. Using contract source code for analysis is very difficult because open-source smart contracts make up only a tiny fraction of all contracts in the real world. Another impacting factor is that blockchain platforms usually store smart contracts in bytecode. Nevertheless, direct analysis of bytecode sequences is also a considerable challenge because each contract is accompanied by a long bytecode, which may be truncated when embedding is performed and takes up a large amount of memory during model training. In addition, it is a challenging task to mark vulnerabilities effectively, and the use of manual marking has the following limitations:

(1) Developers must have professional and comprehensive expert knowledge.

(2) Contract bytecode is generally too long to be easily interpreted.

(3) Manual marking is time-consuming and is only restrictly effective.

Feature Extraction and Representation. This difficulty lies mainly in accurately representing vulnerability features, especially implicit features, which can infer possible new vulnerabilities.

Ductility. Since new types of attacks on smart contracts are emerging quickly, smart contract vulnerability detection models must hastily learn new vulnerabilities using known knowledge. When new attacks emerge, or contracts are rewritten, models need to adapt to new vulnerabilities, but this new information is not easy to incorporate into existing contract detection tools because new vulnerabilities may be equipped with unknown features compared to known ones. Training a new model from scratch consumes many resources and incurs additional costs.

How the MODNN handles these difficulties will be shown in Sections 4 and 5.

Original:

```
1   contract ADAO {
2     mapping (address => uint) public credit;
3
4     function withdraw(uint amount) public {
5       if (credit[msg.sender]>= amount) {
6         require(msg.sender.call.value(amount)());
7         credit[msg.sender]-=amount;
8       }
9     }
10  }
```

Patched:

```
1   contract ADAO {
2     mapping (address => uint) public credit;
3
4     function withdraw(uint amount) public {
5       if (credit[msg.sender]>= amount) {
6         credit[msg.sender]-=amount;
7         require(msg.sender.call.value(amount)());
8       }
9     }
10  }
```

(a) Example of Reentrancy and Patched by SCRepair

... ffffff1681 52 60 2001908152 60 2 00 1 60 00206000828 25 40392 5050 81 905 550 5b50 ...

... 60 4051 600060 40518083038185875af19 25 0 5050 15156102 905 7600080fd5b 5b50 ...

(b) Original Bytecode and Patched by SCRepair

Fig. 3. An example of COS.

4. Data collection

In this section, the problem of data collection and pre-processing is addressed.

Bytecode is executed in the Ethereum virtual machine (EVM) context and contains critical information for detecting vulnerabilities. However, bytecode-level considerations can break the limitations of the contract platform so that any smart contract that is eventually compiled to run in bytecode can be handled, such as in Wanchain [44]. To avoid the loss of vulnerability information during bytecode decompilation and to handle cross-platform contracts, a smart contract crawler called SCC, which can build a dataset large enough for MODNN training from the source and bytecode levels, was developed. Since Ethereum is currently the most prominent smart contract deployment platform, this paper analyzes it as an example.

Fig. 4 shows the overall SCC process. The specific process is as follows:

i Getting Contract Addresses. The verified contract address from [45] is retrieved manually;

ii Extracting Bytecode or Source Code. Crawlers are used to download web information from contract addresses and regular expressions to match source code or bytecode from web information;

iii Cleaning the data. Comments and blank lines are removed;

iv Labeling the code for supervised learning. SCC uses Mythril [40], Smartcheck [41], and Osiris [42] for first-time tag identification. Specifically, the SCC only uses these three tools for the initial flagging of contracts and does not classify specific vulnerability types; For bytecode-only contracts, this paper modifies the intermediate flow of Mythril [40] so that it accepts the input bytecode directly. This is because Mythril [40] works by running the contract bytecode in a purpose-built EVM to check for security issues. Codes with no vulnerabilities are converted to opcodes and marked as 0 and 1 otherwise;

v Vulnerable contract source code is patched with the improved SCRepair [38];

vi Extracting the COS according to the method described in Section 3.2. For hard-to-fix bytecode, this paper extracts the results based on the vulnerability scanner's location as COS;

vii Tagging label for the converted opcode. Previous studies have shown that supervised learning using NLP models can lead to better experimental results.

In step ii, the SCC may not extract the contract source code due to restricted access to Etherscan [45], so the contract information was also downloaded manually, and EDAUB [46] was used as an alternative data source. In step vi, similar operations are merged according to the main keyword, as specified in the Ethereum Yellow Paper [47]. For example, SWAP1, ..., SWAP16 (bytecodes are expressed as 0x90 0x9f), consolidate into SWAP (0x90), regarding them as stop words and deleting them. This paper uses multiple static identification tools for two purposes: (1) identifying the broader range of vulnerabilities; and (2) using a voting mechanism to validate smart contract labels.

Finally, a total of 20545 smart contracts were collected. After removing redundant and unsuccessfully tagged contracts, 18,796 smart contracts remained and were used as experimental data in this paper.

5. System implementation

5.1. Feature extraction

This subsection describes the detailed implementation of the feature extraction tool.

As mentioned in Section 3.2, this paper uses Bert to embed smart contracts to learn explicit features. However, for the model to better identify potential vulnerabilities, implicit features also need to be extracted. The sequence of smart contract operations contains many attributes that cannot be directly determined like "52 2001908152 2 1 00206000828 40392 81 550" in Section 3.3, which can be considered implicit features. Due to the small vocabulary of smart contract sequences, there is no unified or standard corpus, and it is difficult to find these features through vulnerability identification. Therefore, the co-occurrence matrix was used to extract implicit features.

First, the co-occurrence matrix was constructed. Since opcodes are a sequence of specific symbols, they do not conform to the specifications of natural language syntax; therefore, it is almost impossible to use syntax and semantics in the usual sense to extract implicit features. This paper classifies opcodes into nine categories according to the specification in Ethereum Yellow Paper [47], as shown in Table 2. The Stack class is generally considered to be of low relevance to vulnerabilities, so it was deactivated and removed from the sequence. Then Calldata&Codedata, Jump&Stop, and Memory were defined as EOPERATION, and Compute, Compare, Block, Transaction, and Bitoperation were defined as VALUE. EOPERATION and VALUE were used to build the co-occurrence matrix. Fig. 5 illustrates the co-occurrence matrix of the original Opcode Sequence.

Table 2
OPCODES classifications.

Dimension	Type	Instructions
EOPERATION	Calldata&Codedata	callcode, calldatacopy, callvalue, calldataload, calldatasize, codecopy, codesize, extcodecopy
	Jump&Stop	stop, jump, junpi, pc, returndatacopy, return, returndatasize, revert, invalid, selfdestruct
	Memory	mload, mstore, msize, sstore, call, create, delegatecall, staticcall
VALUE	Compute	add(x, y), addmod(x, y, m), div(x, y), exp(x, y), mod(x, y), mul(x, y), mulmod(x, y), sdiv(x, y), signextend(i, x), smod(x, y, m)
	Compare	eq(x, y), iszero(x), gt(x, y), lt(x, y), sgt(x, y), slt(x, y)
	Block	gasprice, gaslimit, difficulty, number, timestamp, coinbase, blockhash(b), keccak256
	Transaction	caller, gas, origin, address, balance
	Bitoperation	and(x, y), byte(n, x), not(x), or(x, y), shl(x, y), shr(x, y), sar(x, y), xor(x, y)
Discarded	Stack	dup, log, pop, push, swap

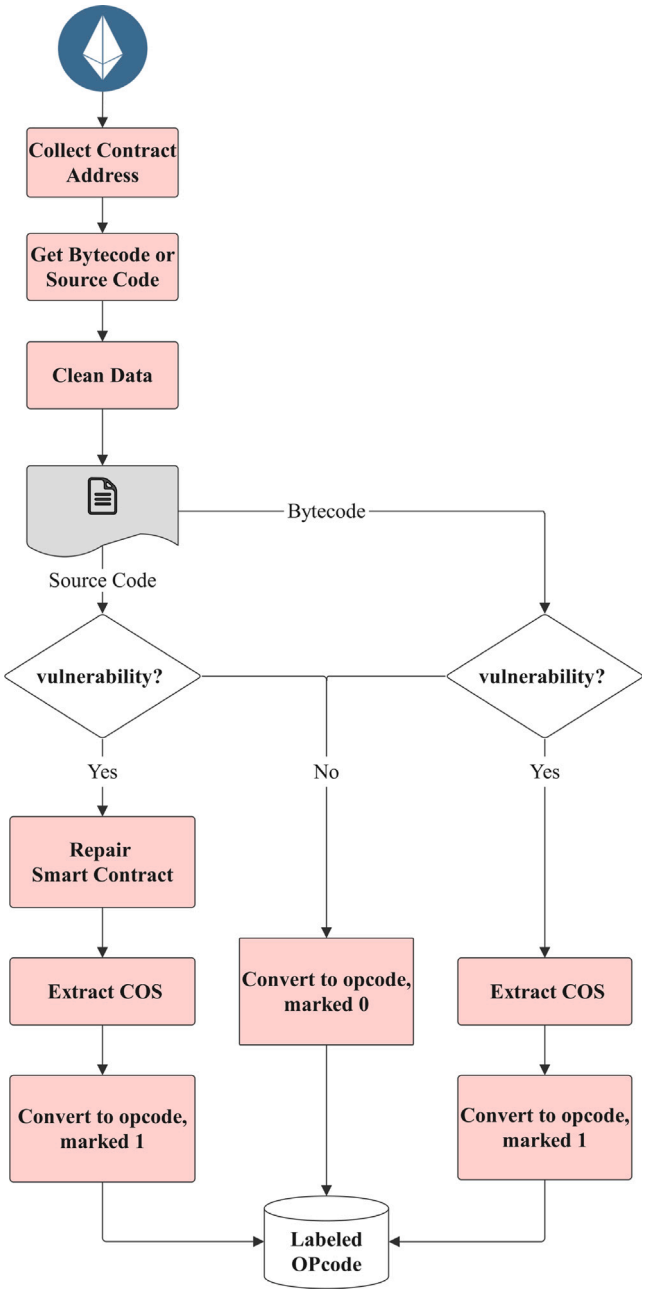


Fig. 4. The overall SCC process.

	EXP	MUL	ISZERO	EXP	SUB	ADD	DIV	EQ
MSTORE	1	10	14	6	2	194	7	1
SLOAD	0	29	47	35	7	21	45	2
SSTORE	0	1	6	10	1	26	6	2
CALLVALUE	0	0	44	1	0	0	1	0
JUMPI	1	8	45	16	8	70	19	49
REVERT	1	2	40	16	3	49	22	6
JUMPDEST	0	5	170	30	11	112	28	8
MLOAD	1	6	63	5	54	73	0	2
CODESIZE	0	0	0	0	1	0	0	0
CODECOPY	0	0	0	0	0	7	0	0
CALLER	0	0	22	3	0	8	0	12
JUMP	0	2	74	16	5	54	16	2

Fig. 5. An example of co-occurrence matrix $\boldsymbol{M}$.

The VALUE with the lower frequency from the co-occurrence matrix, such as EXP in Fig. 5, is selected, and the confidence level CF of the corresponding EOPERATION is calculated using Eq. (1). If CF is also lower, the corresponding VALUE and EOPERATION will be removed from the co-occurrence matrix, and the co-occurrence matrix will be updated.

$$CF(i) = P(i)\sum_{s=i} N \quad i \in E \text{ or } V \tag{1}$$

Where $P(i)$ represents the probability of EOPERATION or VALUE appearing in the sequence, E represents the EOPERATION, V represents the VALUE, and N represents the number of EOPERATIONs or VALUEs with a high frequency.

Second, the relationship between EOPERATION and VALUE is used to extract the implicit features, such as the probability that both are used simultaneously in adjacent or close positions. A candidate's set of implicit features is found based on the probability of the candidate's implicit features corresponding to the EOPERATION, as shown in Eq. (2).

$$F = \{f_i | P(f_i) > p\} \tag{2}$$

Where F represents the candidate set of implicit features, f_i represents the implicit feature, and $P(f_i)$ represents the probability of f_i corresponding to VALUE, which is determined by some trials on all sequences in the training set. p represents the lowest value of the probability of the candidate implicit feature corresponding to EOPERATION, which is taken as 0.5 in this experiment.

This paper uses the co-occurrence matrix for each candidate implicit feature f_i in the F set to calculate the similarity between VALUE and f_i. This paper then filters the implicit features by similarity and confidence scores. The similarity is calculated as shown in Eq. (3), and the final

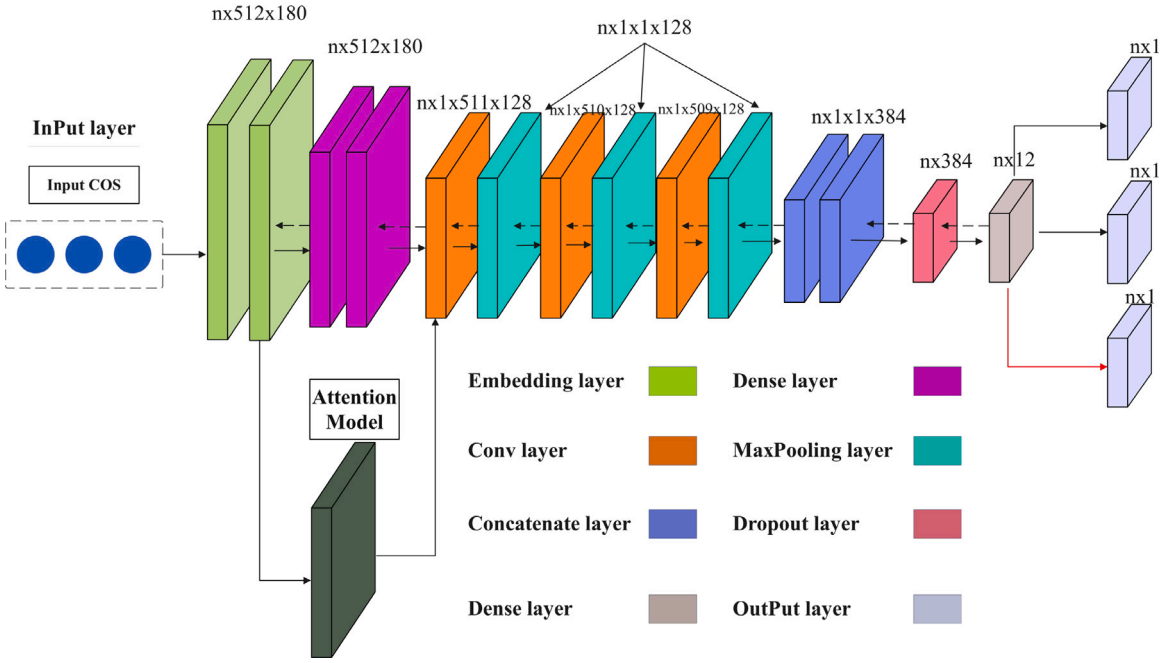


Fig. 6. The entire process of the MODNN.

score is calculated by Eq. (4).

$$sim\left(f_i, f\right) = \frac{\boldsymbol{M}_i * \boldsymbol{M}}{\left\|\boldsymbol{M}_i\right\| * \|\boldsymbol{M}\|} \tag{3}$$

$$score\left(f_i\right) = CF\left(f_i\right)\left[\vartheta * P\left(f_i\right) + \kappa * sim\left(f_i, f\right)\right] \tag{4}$$

In Eq. (3), f_i denotes the ith candidate's implicit feature in the set F, f indicates the actual VALUE in the sequence, $\boldsymbol{M}_i$ denotes the co-occurrence matrix composed of candidate implicit features f_i, and $\boldsymbol{M}$ denotes the co-occurrence matrix consisting of the actual VALUE. In Eq. (4), $CF(f_i)$, $P(f_i)$, and $sim(f_i, f)$ correspond to the values in Eqs. (1), (2), and (3), respectively. For $\vartheta + \kappa = 1$, the weights of ϑ and κ are determined by the data distribution. In the experiments in this paper, $\vartheta = 0.6, \kappa = 0.4$.

In the end, the candidate implicit features with *score* values greater than 0.5 are used as the actual implicit features corresponding to EOPERATION.

5.2. MOD

The COS can be generated using the extraction method described earlier, and the COS is processed through Bert to obtain the eigenvectors of the sequence. For downstream tasks, MOD was used for parallel vulnerability identification and outputted the probability of each vulnerability. Unlike multi-label classification, there is little correlation between the multiple vulnerabilities of a smart contract, so the output layer of the NN was improved to output multiple independent probabilities at once.

First, assume that $X = \mathbb{R}^d$ is the smart contract sample space, $Y = \{y_1, y_2, \ldots, y_q\}$ is the set of loophole names, and the training set $D = \{(\boldsymbol{x}_i, Y_i) \mid 1 \le i \le n\}$ is used to train the multi-objective recognizer $r : X \to 2^L$, where n is the number of smart contracts in the training set. In the training phase, r is trained with the training data $(\boldsymbol{x}_i, Y_i)$ corresponding to the smart contract SC_i. The vulnerability learning results are output, where $\boldsymbol{x}_i \in X$ represents the d dimension feature vector of SC_i after embedding and $Y_i \subseteq Y$ represents all vulnerability classes corresponding to SC_i, with little correlation between them.

Second, the vulnerability of SC_i by function $f^\rho : X \times L \to \mathbb{R}$ is ranked from highest to lowest probability. The function $f^\rho(\boldsymbol{x}_i, \lambda)$ is calculated from r, as shown in Eq. (5).

$$r(x_i) = \left\{\lambda \mid f^\rho(\boldsymbol{x}_i, \lambda) > t(\boldsymbol{x}_i), \lambda \in L\right\} \tag{5}$$

Where $t(\boldsymbol{x}_i)$ is a threshold, and since this paper uses f as the probability of selection rather than a real-valued function, $t(\boldsymbol{x}_i)$ is set to 0.5—in other words, the probability of selecting a vulnerability class with a probability greater than 50%.

Given a test contract SC_j, the multi-objective recognizer r predicts $r\left(\boldsymbol{x}_j\right) \in Y$ as the label set of SC_j and the corresponding probabilities in the testing phase. Since the trained model parameters did not change again during the testing phase, the weighted squared differences were used as weights and biases for the new branches.

To train and test the multi-objective recognizer r, a CNN with the attention mechanism and backpropagation process was used. Unlike normal CNNs, MOD uses a feed-forward NN. There are many problems to be solved when incorporating the backpropagation process. For example, the convolutional layer is the sum of several matrix convolutions used to obtain the output of the current layer. In contrast, the fully connected layer of a normal DNN obtains the output of the current layer by performing matrix multiplication. The whole training process can be divided into four main steps: (i) using the CNN's embedding layer to extract features from $\boldsymbol{x}_i$, (ii) using the attention mechanism to enhance the features of the embedding layer, (iii) using three basic sets of convolution and pooling layers to extract the features of the sequence and using the rectified linear unit (ReLU) as the activation function of the convolution layer, and (iv) using a multiple output structure for the dense layer. As such, each branch outputs a separate probability of vulnerability, and no direct connection exists between each probability. Fig. 6 shows the entire process of the MODNN, and Algorithm 2 describes the training process for the model.

One issue worth noting is that parallel detection was done through hard parameter sharing in the hidden layers, which was done by sharing the hidden layer between all predictive channels while keeping multiple loophole output layers and was not achieved at the cost of reducing detection accuracy. In contrast, thanks to the pre-processing of the input data in Section 3.2, the model could maintain high performance while detecting in parallel and reducing the hardware and

time costs associated with training. Most neural networks essentially make predictions in probabilistic form, but unlike other domains, where expressing vulnerabilities results in absolute binary numbers leading to a less reliable neural network, outputting vulnerability results in a probabilistic form ensures the efficiency of the network quantitatively while effectively increasing the confidence level of the model.

To summarize, the overall process of the MODNN is as follows: it acquires as many COSs as possible, embeds COSs with Bert to obtain explicit features, and introduces the co-occurrence matrix to obtain implicit features to identify potential unknown vulnerabilities, thus supporting the scalability of MODNN. The MOD analyzes learned vulnerabilities from explicit features in parallel, learns potential unknown vulnerabilities from implicit features, and expands the model.

Algorithm 2: MOD with Backpropagation

Input: D, The number of layers of MODNN L, Gradient iteration step size α, Maximum number of iterations MAX, stop iteration threshold Ξ.

Output: W and b of each layer

$W = Random(), b = Random()$;
for $iter \leftarrow 1$ *to* MAX **do**
  for $i \leftarrow 1$ *to* n **do**
    $a^1 = Tensor(\boldsymbol{x}_i)$;
    $a^{i,L} = sigmoid(W^L * a^{i,L-1} + b^L), \delta^{i,L} = BCEWITHLOGITS(a^{i,L}, y_i)$;
    for $l \leftarrow L-1$ *to* 2 **do**
      $z^{i,l} = W^l * a^{i,l-1} + b^l$;
      $\wp \leftarrow \delta^{i,l}$;
      if *Dense* **then**
        $\wp = (W^{l+1})^T \delta^{i,l+1} \odot tanh'\left(z^{i,l}\right)$;
      end
      else if $Conv2D$ **then**
        $\wp = \delta^{i,l+1} * rot180\left(W^{l+1}\right) \odot relu'\left(z^{i,l}\right)$;
      end
      else if $MaxPooling2D$ **then**
        $\wp = upsample\left(\delta^{i,l+1}\right) \odot sigd'\left(z^{i,l}\right)$;
      end
    end
  end
  for $l \leftarrow 2$ *to* L **do**
    if *Dense* **then**
      $W^l = W^l - \alpha \sum_{i=1}^{n} \delta^{i,l}\left(a^{i,l-1}\right)^T$;
      $b^l = b^l - \alpha \sum_{i=1}^{n} \delta^{i,l}$;
    end
    else if $Conv2D$ **then**
      $W^l = W^l - \alpha \sum_{i=1}^{n} rot180\left(\delta^{i,l} * a^{i,l-1}\right)$;
      $b^l = b^l - \alpha \sum_{i=1}^{n} \sum_{u,v} (\delta^{i,l})_{u,v}$;
    end
  end
  if $all(\Delta W, \Delta b) < \Xi$ **then**
    break;
  end
end
return $\{< W_1, b_1 >, \ldots, < W_\gamma, b_\gamma >\}$

6. Performance evaluation

In this section, the experimental setup and evaluation metrics are presented to demonstrate the performance of the MODNN.

Table 3
Performance indicators confusion matrix.

	Positive	Negative
True	TP	TN
False	FP	FN

Table 4
Hardware & software specification.

System	Ubuntu 18.04 LTS	Gensim	3.8.3
CPU	AMD 5600X	Pandas	1.1.3
GPU	NVIDIA 1080ti	max_length	512
RAM	32GB	dropout_rate	0.5
Disk	SSD 512GB	learning_rate	0.0001
NVIDIA Driver	497.29	num_filters	128
Kernel	5.4.72-WSL2	attention_size	180
Tensorflow	2.3.0	patient	10
CUDA	10.1	batch_size	64
Keras	2.3.1	epoch	200
Scikit-learn	0.23.2	embedding_dim	180
BERT	Bert-base-cased	Confidence	0.5

6.1. Label processing

The SCC was used to collect and tag approximately 18,796 smart contracts to build the dataset. Vulnerability scanners can often detect multiple vulnerabilities, and the identification results may intersect. To ensure the reliability of the vulnerability label, the SCC uses a voting mechanism for verification. For example, given a smart contract SC_k, three tools can detect whether it contains reentrant vulnerabilities $SC_{k,rv}$. If $Mythril\left(SC_{k,rv}\right)$ and $Osiris\left(SC_{k,rv}\right)$ are 1, but $Smartcheck\left(SC_{k,rv}\right)$ is 0, then the SCC marks $SC_{k,rv}$ as 1. Note that this is only one example. Other vulnerabilities and tags are treated in the same way.

6.2. Evaluation indicators

MODNN supports the simultaneous detection of multiple vulnerability types. This paper uses the F1 Score, Precision, Accuracy, and Recall to evaluate the performance of the MODNN. In addition, this paper computes the variation in the models' losses during the training process and utilizes the Fowlkes and Mallows Index and Euclidean Distance to validate the classification results of the sample.

Basis criteria. True positive (TP), true negative (TN), false positive (FP), and false negative (FN) are the base values for the calculation of the other indicators. A true value represents a correct prediction, which may be a TP or TN. False values indicate that the neural network has made incorrect predictions. The relationship between these four is often represented by a confusion matrix, as shown in Table 3.

Precision, Accuracy, and Recall. The Accuracy metric characterizes the ratio of TPs to all predicted positives. The Recall metric shows the proportion of correctly classified positive samples to the total positive samples. The measures for calculating these two metrics are communicated as follows:

$$Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN} \tag{6}$$

F1 Score. The F1 Score metric is commonly used for information retrieval, and it uses Accuracy and Recall to quantify the overall prediction Accuracy of the model. The F1 Score is defined as the summed average of Accuracy and Recall.

$$F1_{Score} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{7}$$

The optimal and worst values of the F1 Scores are 1 and 0, respectively. F1 Scores can be calculated for each label class or globally. The current evaluation used weighted F1 Scores, where each class's F1 Score was weighted by the number of samples from that class.

Table 5
Initial training performance & ductility.

Metrics	Category	Count	Loss	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
Known	Underflow	11421	0.186	96.15	95.86	96.41	96.13
	Overflow	12960	0.120	98.57	98.84	99.06	98.95
	CallDepth	2363	0.185	93.83	92.67	95.83	94.22
	TOD	2580	0.180	91.40	98.51	97.23	97.87
	TimeDep	1167	0.191	94.66	94.58	92.74	93.65
	Reentrancy	1528	0.183	98.33	95.86	92.95	94.38
	AssertFail	7467	0.167	91.40	96.36	99.16	97.74
	TxOrigin	135	0.172	96.67	99.89	95.56	97.68
	CheckEffects	6825	0.159	97.41	96.06	95.49	95.77
Unknown	InlineAssembly	1317	0.207	91.23	90.24	92.18	91.20
	BlockTimestamp	5478	0.332	89.81	88.28	90.47	89.36
	LowlevelCalls	5268	0.218	90.92	89.59	91.46	90.52

Table 6
Performance comparison on validation sets.

Dataset	Model	Count	Precision (%)	Recall (%)	F1 Score (%)	Prediction (%)
Own	MODNN	1600	**96.51**	96.05	**96.28**	**90.65**
	DR-GCN	1329	89.68	**100.00**	94.56	–
	Eth2Vec	1600	77.00	50.70	57.50	74.90
	SoliAudit	1600	92.90	88.20	90.40	–
Xblock	MODNN	2000	**95.07**	**94.65**	**94.68**	**85.74**
	DR-GCN	2000	92.21	92.46	92.33	58.93
	Eth2Vec	2000	79.20	65.70	71.82	76.23
	SoliAudit	2000	87.74	83.15	85.38	80.69
SB Wild	MODNN	2000	**92.20**	91.59	**91.89**	**86.53**
	DR-GCN	2000	80.70	81.40	81.05	64.42
	Eth2Vec	2000	77.50	79.82	78.64	76.51
	SoliAudit	2000	86.43	**91.80**	89.03	84.00
SolidiFI	MODNN	350	91.78	**95.12**	93.42	85.04
	DR-GCN	350	**95.89**	93.06	**94.45**	63.17
	Eth2Vec	350	84.83	82.61	83.71	73.27
	SoliAudit	350	89.59	90.76	90.17	**88.96**

Fowlkes and Mallows Index. In multi-label classification, the Fowlkes and Mallows Index is defined as the measure of similarity between two hierarchical clusters or clustering and a benchmark classification. The Fowlkes and Mallows Index is only applicable to a binary variable, for which the Fowlkes and Mallows Index ranges from 0 to +1, where +1 is the highest similarity. This indicates that the closer the predicted value is to the actual value, the better the model effect is and is calculated as follows.

$$FM_{Index} = \sqrt{\frac{TP}{TP+FP} * \frac{TP}{TP+FN}} \tag{8}$$

Euclidean Distance. Euclidean Distance represents the ratio of incorrect labels to the total number of labels. The lower the Euclidean Distance, the better the model's performance. A Euclidean Distance of 0 indicates that the model is already perfect. In multi-label classification, the Euclidean Distance is the distance between the actual value h and the predicted value m, where $Length(h)$ represents the total number of vulnerabilities the total number of types.

$$d(h,m) = \sqrt{\left(\sum \left(h_i - m_i\right)^2\right)} \tag{9}$$

$$\mathcal{L}_{Euclidean} = \frac{1}{1+d(h,m)} \tag{10}$$

Loss function. The loss function is a key part of the model because the training process aims to minimize the loss to obtain high Accuracy for the task. Therefore, the loss value quantifies the classifier's performance on a given dataset. In the multi-label classification task, each result of the output layer is generally considered a binomial distribution; in other words, a multi-label problem is transformed into a binary classification problem on each label. Therefore, MODNN uses sigmoid as the activation function of the output layer and binary_crossentropy_with_logits as the loss function. Given the expected values and predictions, the loss is calculated as follows, where h is the target value and m is the model's actual output.

$$L_l(h,\ m) = -(h\log(m) + (1-h)\log(1-m)) \tag{11}$$

6.3. Experimental setup

Processing Dataset Imbalance. Since the raw contract data collected often suffers from class imbalance [24], it needs to be preprocessed to ensure that each batch of input data has almost an equal number of vulnerable and secure contracts. MODNN uses the focal loss algorithm to deal with the imbalance problem in the dataset.

Overfitting and Underfitting. MODNN uses k-fold cross-validation (KCV) to validate each randomly generated subset. Compared with leave-one-out cross-validation (LOOCV), leave-p-out cross-validation (LPOCV), and repeated random subsampling validation (RRSV), each sample data for KCV can be used for both testing and training to avoid overfitting and underfitting to obtain more convincing results. When the loss of MODNN is within the acceptable range, the stop condition control parameter "early_stop" value is reset to 1.

Optimizer and Learning Rate. Due to the long operation sequence of smart contracts, the model takes up much memory during training. The Adam optimizer is easy to implement, computationally efficient, and can reduce resource requirements, so it was chosen as an optimizer for MODNN. In addition, compared to stochastic gradient descent (SGD), Adam combines the mean and uncentered variance of the gradient and calculates the update step, which also enables a natural annealing process—in other words, the automatic adjustment of the learning rate—and is very suitable for scenarios with large scale data and parameters. However, this paper found that when using the incipient learning rate, the training curve will have intermittent and large oscillations, and some networks will have a cliff-like fall and will not recover after fixing to a value. Therefore, this paper reduces

the initial learning rate to 0.0001, which can solve some unstable situations, although the model training is much slower.

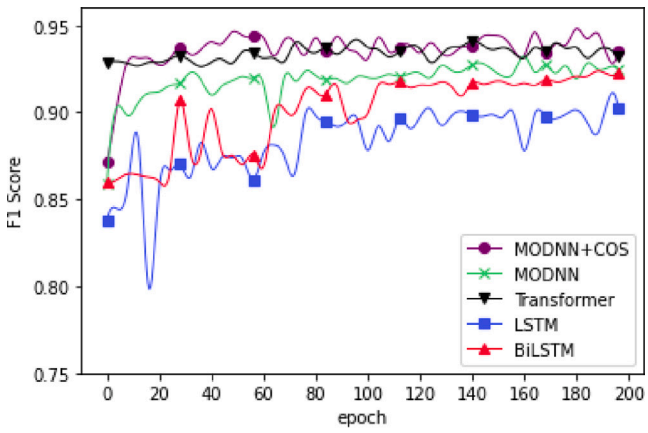
Hardware and Drivers. Our experiments were conducted on a Linux subsystem with Ubuntu 18.04 LTS, a mother computer with an AMD 5600X CPU, NVIDIA RTX 1080ti graphic card, and 32 GB of RAM. The graphic card driver version was 497.29, and the kernel version was 5.4.72-Microsoft-standard-WSL2. Note that to use NVIDIA GPUs and CUDA in the Ubuntu subsystem, the host system needs to be upgraded to Microsoft's June 2020 update or later. Complete information about the hardware and programs used in this paper is shown in Table 4.

6.4. Evaluation results

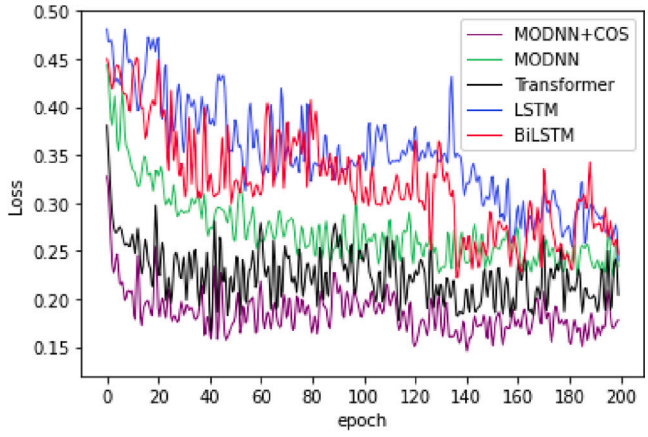
The model was trained with nine types of contracts with different vulnerabilities. Three types of new contracts were added in the testing phase to verify the model's ability to identify unknown vulnerabilities.

Classifier Learning. The MODNN model was trained on the training set using the hardware platforms and programs listed in Table 4 and evaluated by the test set. To confirm the MODNN's effectiveness on the nine known vulnerabilities, the F1 Score and loss curves in the training process are plotted in Fig. 7. The training and validation curves demonstrate the time-evolving performance of MODNN under supervised learning and the generalization ability of the model, respectively. The learning curves show that the MODNN could achieve an average F1 Score higher than 94% for both the training and validation sets. MODNN showed a 2%–3% improvement compared to no COS (92.57%) and an improvement compared to the transformer (93.81%), while for both models in the LSTM family, the MODNN showed a 3%–5% improvement. Another conclusion can also be drawn from the curves in Fig. 7: under the same conditions, the MODNN converged at a smaller number of epochs, a value (about 20 epochs) that is five times higher than that of bi-directional (Bi-) LSTM (about 100 epochs). This suggests that MODNN can be trained in a relatively short time, with considerable reductions in training costs.

Detection Performance. This paper provides fine-grained insight into the capabilities of each vulnerability category. Table 5 shows the specific metric values obtained by our model for each specific vulnerability type. Among all the results, Recall obtained the smallest value of 90.47% on BlockTimestamp. In the case of InlineAssembly, MODNN had lower Precision compared to others. The F1 Score indicated the generation prediction Accuracy of MODNN according to Eq. (7). In particular, MODNN showed an average Precision of 96.51% on the nine learned types of vulnerabilities, a Recall of 96.05%, and an F1 Score of 96.27%. For the other three vulnerabilities, the scalability of the MODNN, transformer, LSTM, and Bi-LSTM at a default confidence level of 0.5 was evaluated. Since it is impossible to give the corresponding labels in advance for unknown types of vulnerabilities, all three remaining vulnerability labels in the test sequence were assumed to be 1, and then vulnerability prediction was performed. To ensure that the prediction results of these three vulnerabilities were reliable, the prediction results of the NNs were compared with the markers of the static analysis tools. The final scalability performance of these four models on 2000 test sequences is shown in Fig. 8.



(a) F1 Score



(b) Loss

Fig. 7. Performance comparison of different.

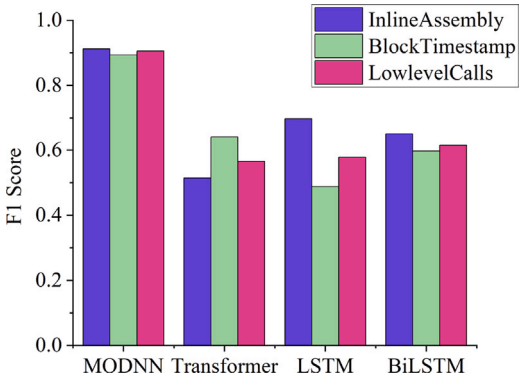


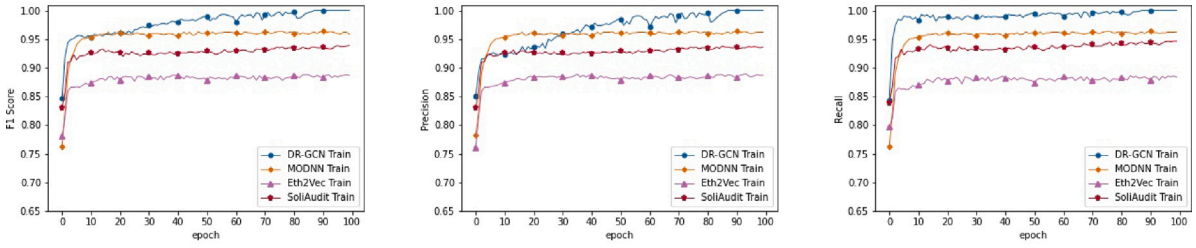
Fig. 8. Ductility comparison.

6.5. Methods comparison

The open-source and currently state-of-the-art DR-GCN [11], SoliAudit capable of identifying unknown vulnerabilities [18], and Eth2Vec capable of learning tacit knowledge [19] was selected as baseline models. To eliminate the contingencies and untrustworthiness of contract data collected once, a comparative experiment using public datasets Xblock [48], SB Wild [49], and SolidiFI Benchmark [50] was also conducted. Note that the parameter configurations of the baseline experiments all used recommended or default values for each model. In Figs. 9–11, and Table 6, the comparison results of the MODNN and baseline models for training and validation sets are shown, respectively. Please note that SolidiFI was only used for validations of models as it was too small.

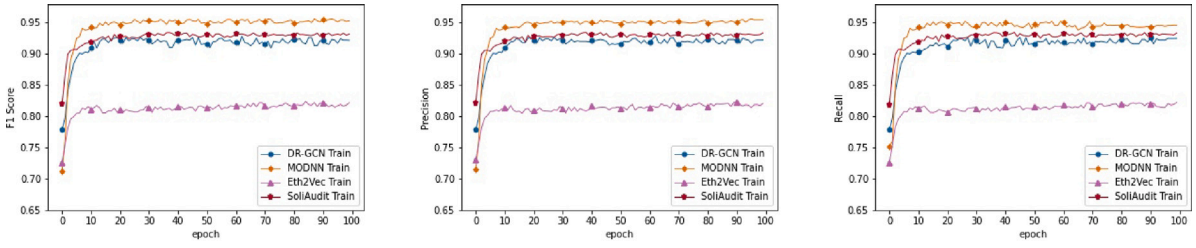
Combining Table 6 with Figs. 9–11, the following conclusions can be drawn in terms of overall performance.

Own Dataset. On their own datasets, the MODNN outperformed the state-of-the-art baseline model on both F1 Score and Precision, with the largest gain of 6.83% for Precision and a smaller gain of 1.72% for F1 Score, which shows the true significance of the MODNN. Although the MODNN failed to achieve breakthroughs in Recall, the MODNN showed notable results that superior to Eth2Vec and SoliAudit in the area as



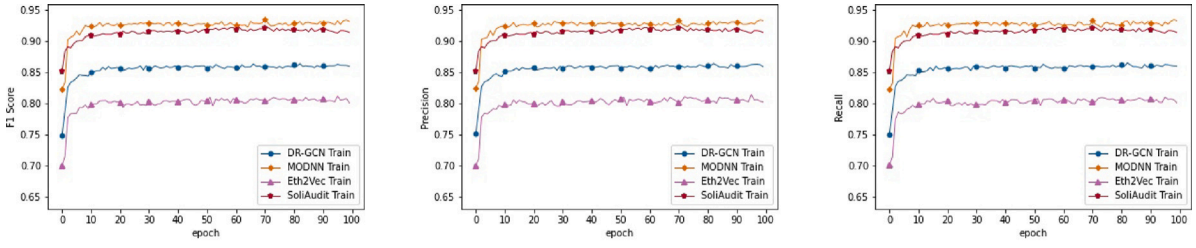
(a) F1 Score (b) Precision (c) Recall

Fig. 9. Comparison of training results of MODNN and baseline models on own datasets.



(a) F1 Score (b) Precision (c) Recall

Fig. 10. Comparison of training results of MODNN and baseline models on Xblock.



(a) F1 Score (b) Precision (c) Recall

Fig. 11. Comparison of training results of MODNN and baseline models on SB wild.

well. MODNN also gained over 15% performance improvement in new vulnerability prediction.

Public Dataset. On public datasets, the performance improvement of MODNN was significant, and it outperformed the baseline model across all four commonly used performance indicators. MODNN demonstrated relatively more consistent recognition and better compatibility with random datasets than the unimpressive F1 Score performance of baseline models. It also further demonstrated the sufficiency of the SCC's use of bytecodes and opcodes to pre-process data, which can go a long way toward making different datasets have more uniform rules for feature extraction.

For DR-GCN, its relatively poor performance on public datasets could be explained by the fact that it is a statement-specific rule-based vulnerability detection model that uses keywords from the contract source code as classification flags, which is particularly suitable for Reentrancy, InfiniteLoop, and TimeDep vulnerabilities; however, it lacks compatibility with the remaining types of vulnerabilities, and when validated using more general datasets, the lack of malleability leads to a large number of classification errors. In contrast, the MODNN is a model that accommodates multiple vulnerability types and can learn new vulnerabilities through parameter and feature sharing, which again demonstrates the relevance of the MODNN. It appears to be better trained than the MODNN on its own dataset, but a deeper analysis of Fig. 9 reveals new findings. As shown in Fig. 9(a), MODNN maintains almost the same training curve as DR-GCN until the epoch reaches 20. During $20 < \text{epoch} < 85$, DR-GCN showed a significant performance improvement while MODNN maintained a stable training curve, but this performance improvement cannot be simply attributed to DR-GCN being better. This is because during $85 <= \text{epoch} <= 100$, the F1 Score (Fig. 9(a)), Precision (Fig. 9(b)), and Recall (Fig. 9(c)) surprisingly all moved toward 1. There was an obvious overfit in DR-GCN, yet the only stable section of the training curve before the overfitting remained almost identical to the MODNN. Figs. 10 and 11 show the difference in performance between the MODNN and the DR-GCN on public datasets. Compared to the training curves for their own datasets, both the MODNN and the DR-GCN show different degrees of performance degradation, the most severe being the 18.60% drop in the Recall of DR-GCN shown in Table 6, while the performance loss of the MODNN was maintained between 0.93% and 4.46%, less a fourth of DR-GCN, thus fully demonstrating MODNN's robustness and generalizability.

For Eth2Vec, although it was able to learn tacit knowledge, it did not show outstanding performance in this experiment, and it could be assumed that Eth2Vec's unsupervised learning limited model performance. For SoliAudit, it was able to quickly adapt to new unknown vulnerabilities without expert knowledge and predefined features. On SolidiFI, SoliAudit did demonstrate optimal unknown vulnerability prediction, however, it is worth noting that this unknown vulnerability prediction was based on retraining the model with new data, which inadvertently increased the model overhead. Furthermore, the MODNN also showed more stable predictions of new vulnerabilities compared to SoliAudit.

In summary, the MODNN showed good recognition results on both its own and public datasets, and the Precision and F1 Score reached optimum values. Thus, the contract data collected in this paper matches

the overall data distribution pattern on the blockchain system without serious bias or prejudice. Admittedly, several performance metrics of the MODNN on public datasets were lower than the performance on the dataset collected in this paper, which is probably related to the size of the smart contracts, as contracts that are too long or too short tend to bring additional noise. When the noisy data reach a certain percentage, this will affect the model. Finally, compared to the performance of the baseline model on the public dataset, MODNN also demonstrated greater robustness and generalizability, further illustrating the importance of the MODNN.

7. Conclusion and future work

This paper proposed the MODNN, a smart contract vulnerability detection tool based on the ML MOD method, to secure cryptocurrency systems. The concept of the COS was also introduced, and it was shown that using the COS improved performance more than ordinary sequences of operations. In addition, a novel approach to extracting the implicit features of operation sequences was designed to support the model's scalability. Finally, the scalability of several models was tested to demonstrate the effectiveness of the proposed approach. However, the current experimental data in this paper used opcodes, which only consider the correlation between operations, and MODNN does not fully consider the syntactic and semantic information of the original program compared to many existing NLP-based program vulnerability detection methods. In particular, the smart contract source code's contextual information and syntactic and semantic information are not sufficiently considered when extracting implicit features. One of our future goals is to combine the contract's raw syntactic and semantic information and the operation. To make our work more beneficial to the sound development of blockchain systems such as Ethereum, more MODNN materials will be released to [10].

CRediT authorship contribution statement

Lejun Zhang: Supervision, Funding acquisition, Data curation, Writing – original draft. **Jinlong Wang:** Conceptualization, Methodology, Software, Investigation, Formal analysis, Writing – review & editing. **Weizheng Wang:** Visualization, Investigation. **Zilong Jin:** Resources, Supervision. **Yansen Su:** Software, Validation. **Huiling Chen:** Conceptualization, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the link to our data/code at the Attach File step.

Acknowledgments

The authors would like to thank the reviewers for their detailed reviews and constructive comments, which have helped improve the quality of this paper. This work is sponsored by the National Key Research and Development Program of China No. 2021YFE0102100. The National Natural Science Foundation of China under grant number No. 62172353. Future Network Scientific Research Fund, China Project No. FNSRFP-2021-YB-48. Science and Technology Program of Yangzhou City, China No. YZU202003 and Six Talent Peaks Project in Jiangsu Province, China No. XYDXX-108.

References

[1] M. Fries, B.P. Paal (Eds.), Smart Contracts, first ed., Mohr Siebeck, 2019, http://dx.doi.org/10.1628/978-3-16-156911-1.
[2] R. Ma, Z. Jian, G. Chen, K. Ma, Y. Chen, ReJection: A AST-based reentrancy vulnerability detection method, in: Chinese Conference on Trusted Computing and Information Security, Springer, 2019, pp. 58–71.
[3] C. Liu, J. Gao, Y. Li, Z. Chen, Understanding out of gas exceptions on ethereum, in: International Conference on Blockchain and Trustworthy Systems, Springer, 2019, pp. 505–519.
[4] W. Zou, D. Lo, P.S. Kochhar, X.-B.D. Le, X. Xia, Y. Feng, Z. Chen, B. Xu, Smart contract development: Challenges and opportunities, IEEE Trans. Softw. Eng. 47 (10) (2019) 2084–2106.
[5] N. Atzei, M. Bartoletti, T. Cimoli, A survey of attacks on ethereum smart contracts (sok), in: International Conference on Principles of Security and Trust, Springer, 2017, pp. 164–186.
[6] M.I. Mehar, C.L. Shier, A. Giambattista, E. Gong, G. Fletcher, R. Sanayhie, H.M. Kim, M. Laskowski, Understanding a revolutionary and flawed grand experiment in blockchain: The DAO attack, J. Cases Inf. Technol. 21 (1) (2019) 19–32.
[7] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, M. Vechev, Securify: Practical security analysis of smart contracts, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 67–82.
[8] J. Choi, D. Kim, S. Kim, G. Grieco, A. Groce, S.K. Cha, SMARTIAN: Enhancing smart contract fuzzing with static and dynamic data-flow analyses, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, 2021, pp. 227–239.
[9] J. Feist, G. Grieco, A. Groce, Slither: A static analysis framework for smart contracts, in: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB, IEEE, 2019, pp. 8–15.
[10] J. Wang, MODNN data, 2022, [EB/OL]. https://github.com/yangzuwjl/MODNN. (Accessed 29 April 2022).
[11] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, X. Wang, Combining graph neural networks with expert knowledge for smart contract vulnerability detection, IEEE Trans. Knowl. Data Eng. (2021).
[12] K. Zhou, J. Cheng, H. Li, Y. Yuan, L. Liu, X. Li, Sc-VDM: A lightweight smart contract vulnerability detection model, in: International Conference on Data Mining and Big Data, Springer, 2021, pp. 138–149.
[13] T.H.-D. Huang, Hunting the ethereum smart contract: Color-inspired inspection of potential attacks, 2018, arXiv preprint arXiv:1807.01868.
[14] S. Jeon, G. Lee, H. Kim, S.S. Woo, SmartConDetect: Highly accurate smart contract code vulnerability detection mechanism using BERT, 2021.
[15] H. Wu, Z. Zhang, S. Wang, Y. Lei, B. Lin, Y. Qin, H. Zhang, X. Mao, Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques, in: 2021 IEEE 32nd International Symposium on Software Reliability Engineering, ISSRE. IEEE, 2021, pp. 378–389.
[16] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, et al., Graphcodebert: Pre-training code representations with data flow, 2020, arXiv preprint arXiv:2009.08366.
[17] X. Yu, H. Zhao, B. Hou, Z. Ying, B. Wu, DeeSCVHunter: A deep learning-based framework for smart contract vulnerability detection, in: 2021 International Joint Conference on Neural Networks, IJCNN, IEEE, 2021, pp. 1–8.
[18] J.-W. Liao, T.-T. Tsai, C.-K. He, C.-W. Tien, Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing, in: 2019 Sixth International Conference on Internet of Things: Systems, Management and Security, IOTSMS, IEEE, 2019, pp. 458–465.
[19] N. Ashizawa, N. Yanai, J.P. Cruz, S. Okamura, Eth2Vec: Learning contract-wide code representations for vulnerability detection on ethereum smart contracts, in: Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure, 2021, pp. 47–59.
[20] F. Hill, K. Cho, A. Korhonen, Learning distributed representations of sentences from unlabelled data, 2016, arXiv preprint arXiv:1602.03483.
[21] Z. Gao, When deep learning meets smart contracts, in: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, 2020, pp. 1400–1402.
[22] W.J.-W. Tann, X.J. Han, S.S. Gupta, Y.-S. Ong, Towards safer smart contracts: A sequence learning approach to detecting security threats, 2018, arXiv preprint arXiv:1811.06632.
[23] A.K. Gogineni, S. Swayamjyoti, D. Sahoo, K.K. Sahu, R. Kishore, Multi-class classification of vulnerabilities in smart contracts using AWD-LSTM, with pre-trained encoder inspired from natural language processing, IOP SciNotes 1 (3) (2020) 035002.
[24] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, C. Su, Contractward: Automated vulnerability detection models for ethereum smart contracts, IEEE Trans. Netw. Sci. Eng. 8 (2) (2020) 1133–1144.
[25] J. Huang, S. Han, W. You, W. Shi, B. Liang, J. Wu, Y. Wu, Hunting vulnerable smart contracts via graph embedding based bytecode matching, IEEE Trans. Inf. Forensics Secur. 16 (2021) 2144–2156.

[26] Z. Hu, K. Shaloudegi, G. Zhang, Y. Yu, Federated learning meets multi-objective optimization, IEEE Trans. Netw. Sci. Eng. (2022) 1, http://dx.doi.org/10.1109/TNSE.2022.3169117.
[27] X. Zhang, H. Liu, L. Tu, A modified particle swarm optimization for multimodal multi-objective optimization, Eng. Appl. Artif. Intell. 95 (2020) 103905, http://dx.doi.org/10.1016/j.engappai.2020.103905, URL https://www.sciencedirect.com/science/article/pii/S0952197620302414.
[28] Y. Bi, B. Xue, M. Zhang, Multi-objective genetic programming for feature learning in face recognition, Appl. Soft Comput. 103 (2021) 107152, http://dx.doi.org/10.1016/j.asoc.2021.107152, URL https://www.sciencedirect.com/science/article/pii/S1568494621000752.
[29] S. Larabi-Marie-Sainte, S. Ghouzali, Multi-objective particle swarm optimization-based feature selection for face recognition, Stud. Inform. Control. J. 29 (2020) 99–109.
[30] A. Ouni, M. Kessentini, H. Sahraoui, M. Boukadoum, Maintainability defects detection and correction: A multi-objective approach, Autom. Softw. Eng. 20 (1) (2013) 47–79.
[31] U. Mansoor, M. Kessentini, B.R. Maxim, K. Deb, Multi-objective code-smells detection using good and bad design examples, Softw. Qual. J. 25 (2) (2017) 529–552.
[32] Z. Cui, L. Du, P. Wang, X. Cai, W. Zhang, Malicious code detection based on CNNs and multi-objective algorithm, J. Parallel Distrib. Comput. 129 (2019) 50–58.
[33] Z. Cui, Y. Zhao, Y. Cao, X. Cai, W. Zhang, J. Chen, Malicious code detection under 5G HetNets based on a multi-objective RBM model, IEEE Netw. 35 (2) (2021) 82–87.
[34] C. Shi, Y. Xiang, R.R.M. Doss, J. Yu, K. Sood, L. Gao, A bytecode-based approach for smart contract classification, 2021, arXiv preprint arXiv:2106.15497.
[35] L. Liu, W.-T. Tsai, M.Z.A. Bhuiyan, H. Peng, M. Liu, Blockchain-enabled fraud discovery through abnormal smart contract detection on Ethereum, Future Gener. Comput. Syst. 128 (2022) 158–166.
[36] Y. Lin, H. Xiaoshuo, W. Jiayang, D. Lingling, L. Zixiao, L. Jiao, Clinical trial disease subtype identification based on BERT-TextCNN, in: Data Analysis and Knowledge Discovery, 2021, p. 1.
[37] Y. Song, MIHNet: Combining N-gram, sequential and global information for text classification, J. Phys. Conf. Ser. 1453 (1) (2020) 012156.
[38] T. Ji, L. Chen, X. Mao, X. Yi, Automated program repair by using similar code containing fix ingredients, in: 2016 IEEE 40th Annual Computer Software and Applications Conference, COMPSAC, Vol. 1, IEEE, 2016, pp. 197–202.
[39] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254–269.
[40] ConsenSys, Mythril-reversing and bug hunting framework for the ethereum blockchain, 2021, URL https://pypi.org/project/mythril/0.22.0. (Accessed 12 August 2021).
[41] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, Y. Alexandrov, Smartcheck: Static analysis of ethereum smart contracts, in: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, 2018, pp. 9–16.
[42] C.F. Torres, J. Schütte, R. State, Osiris: Hunting for integer bugs in ethereum smart contracts, in: Proceedings of the 34th Annual Computer Security Applications Conference, 2018, pp. 664–676.
[43] R.C. Edgar, S. Batzoglou, Multiple sequence alignment, Curr. Opin. Struct. Biol. 16 (3) (2006) 368–373.
[44] jameszhu, Wanchain - Decentralized finance interoperability, 2022, URL https://www.wanchain.org. (Accessed 7 January 2022).
[45] Etherscan, Etherscan China ethereum (ETH) blockchain explorer, 2021, URL https://goto.etherscan.com. (Accessed 15 August 2021).
[46] EDAUB, Ethereum contract library by Dedaub, 2021, URL https://library.dedaub.com. (Accessed 15 August 2021).
[47] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, 2021, URL https://ethereum.github.io/yellowpaper/paper.pdf. (Accessed 15 August 2021).
[48] Y. Huang, Q. Kong, N. Jia, X. Chen, Z. Zheng, Recommending differentiated code to support smart contract update, in: 2019 IEEE/ACM 27th International Conference on Program Comprehension, ICPC, IEEE, 2019, http://dx.doi.org/10.1109/icpc.2019.00045.
[49] T. Durieux, J.a.F. Ferreira, R. Abreu, P. Cruz, Empirical review of automated analysis tools on 47,587 Ethereum smart contracts, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, 2020, pp. 530–541.
[50] A. Ghaleb, K. Pattabiraman, How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection, in: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2020.

Lejun Zhang, received his M.S. degree in computer science and technology in Harbin Institute of Technology and the Ph.D. degrees in computer science and technology at Harbin Engineering University, where he was an professor at Guangzhou University. His research interests include computer network, social network analysis, dynamic network analysis and information security.

Jinlong Wang, is currently pursuing the M.S. degree with the College of Information Engineering, Yangzhou University, China. His main research interests lie in Blockchain security.

Weizheng Wang, received the B.S. degree in software engineering from Yangzhou University, Yangzhou, China, in 2019, the M.S. degrees in computer science and engineering from the University of Aizu, Aizu-Wakamatsu, Japan, in 2021. Now he is currently a Research Associate in University of Aizu and pursuing the Ph.D. degree in computer science at the City University of Hong kong, Hong Kong. His research interests include applied cryptography, blockchain technology and IoT system.

Zilong Jin, received the B.E. degree in computer engineering from Harbin University of Science and Technology, China, in 2009, and the M.S. and Ph.D. degrees in computer engineering from Kyung Hee University, Korea, in 2011 and 2016, respectively. He is currently an assistant professor of School of Computer and Software at Nanjing University of Information Science and Technology, China. His research interests include wireless sensor networks, mobile wireless networks, and cognitive radio networks.

Yansen Su, received the B.Sc. degree from Tangshan Normal University, Tangshan, China, in 2007, the M.Sc. degree from Shandong University of Science and Technology, Qingdao, China, in 2010, and the Ph.D. degree from Huazhong University of Science and Technology, Wuhan, China, in 2014.She is currently an Associate Professor in the School of Computer Science and Technology, Anhui University, Hefei, China. Her main research interests include complex networks, computational biology, and multi-objective optimization.

Huiling Chen is currently an associate professor in the college of computer science and artificial intelligence at Wenzhou University, China. He received his Ph.D. degree in the department of computer science and technology at Jilin University, China. His present research interests center on evolutionary computation, machine learning, data mining, and their applications to medical diagnosis. With more than 14000 citations and an H-index of 66, he is ranked worldwide among top scientists for Computer Science & Electronics prepared by Guide2Research. He is currently serving as the co-editors-in-chief of Computers in Biology and Medicine. He has published more than 200 papers in international journals and conference proceedings, including Information Sciences, Pattern Recognition, IEEE Systems Journal, Future Generation Computer System, Expert Systems with Applications, Knowledge-based Systems, PAKDD, and others. He has more than 20 ESI highly cited papers and 5 hot cited papers.