Ponzi Scheme Detection Based on Control Flow Graph Feature Extraction

Shunhui Ji[1,2], Congxiong Huang[1,2], Pengcheng Zhang[1,2,*], Hai Dong[3], and Yan Xiao[4]

[1]Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing, China
[2]College of Computer and Information, Hohai University, Nanjing, China
[3]School of Computing Technologies, RMIT University, Melbourne, Australia
[4]School of Cyber Science and Technology, Sun Yat-sen University, Shenzhen, China
{*shunhuiji, hcx, pchzhang*}*@hhu.edu.cn, hai.dong@rmit.edu.au, xiaoyan.hhu@gmail.com*

***Abstract*—The blockchain ecosystem is expanding as a result of advancements in blockchain technology and the emergence of BaaS (Blockchain as a Service) platforms. Smart contracts are designed to carry out diverse business operations, but there is a risk of Ponzi schemes being concealed within them. These schemes masquerade as investment agreements and deceive users, resulting in substantial losses for the blockchain community. Detecting Ponzi schemes in smart contracts is crucial. This study introduces a machine learning approach to identify Ponzi schemes by extracting features from smart contracts using the control flow graph. During the construction of the control flow graph for the smart contract's bytecode, elements unrelated to its functionality are identified and eliminated. We utilize the control flow graph to extract n-gram Term Frequency and n-gram Term Frequency-Inverse Document Frequency features. These features are respectively employed to construct a Random Forest model for Ponzi scheme detection. To address the issue of imbalanced samples, the SVM_SMOTE oversampling algorithm is applied to balance the number of positive and negative samples. The results from experiments conducted on a real-world dataset demonstrate the effectiveness of our approach. The feature extraction method based on the control flow graph outperforms the method based on continuous text. Additionally, the Random Forest model utilizing SVM_SMOTE outperforms four existing models.**

***Index Terms*—Blockchain, Smart contract, Ponzi scheme detection, Control flow graph, Machine learning**

I. Introduction

Blockchain [1], as a distributed ledger technology, shows appealing advantages of decentralisation and anonymity. Blockchain technology has been applied to a variety of domains, such as healthcare [2] and energy [3]. However, the development of blockchain network is challenging since it requires multiple technologies, including network protocol, consensus mechanism, cryptography, etc. This sets a high bar for the development of blockchain business and makes it costly in terms of time and money. Therefore, the concept of BaaS (Blockchain as a Service) [4] emerged. BaaS aims to provide a cloud-based blockchain service for business owners, reducing the cost of constructing and maintaining the blockchain and allowing owners to focus on their business scenarios.

On the top of the BaaS system architecture, smart contracts are created to implement business functions. Ethereum can be considered as a provider of a decentralized BaaS [5]. For the smart contract on Ethereum, the transaction and the requirements required by the transaction are programmed in Solidity. The compiled bytecode of the Solidity code is then deployed on Ethereum, on which the transaction will be automatically executed once the requirements are satisfied [6]. The emergence of smart contract enables various functions to be implemented conveniently, thus forming rich application ecology for users on Ethereum [7]. With the continuous development, more and more applications can be realized on Ethereum, such as issuing virtual tokens, trading digital assets, playing smart contract games, etc. However, the criminals take advantage of flaws in the development to launch various scams in smart contracts for obtaining digital assets of the users. Due to the anonymity and immutability of Ethereum itself [6], once digital assets are stolen and transferred, it is an irrevocable deal. This has severely affected the development of Ethereum.

Ponzi scheme is one of the most famous blockchain schemes, which often causes huge economic losses. Ponzi scheme is usually disguised as a high-yield investment program, which promises investors and users much higher returns than ordinary investments. However, in a Ponzi scheme, only the previous investors are paid, and it will collapse if there are no enough new investors. Finiko [8], a Russian Ponzi scheme created in December 2019, was initially defined as a virtual currency investment fund that promised investors 5% return every day. By the end of 2020, about 200,000 investors participated the scheme. By August 2021, the scheme collapsed, making a profit of nearly $100 million from its victims. Ponzi schemes usually use smart contracts as carriers, disguised as investment projects, gambling games and other forms of fraud to defraud trust and money of users, affecting the overall reputation of blockchain platform. Therefore, detecting Ponzi schemes in smart contracts is crucial.

Various detection methods have been proposed to detect Ponzi schemes in smart contracts [9], [10]. The bytecodes of smart contracts are publicly available on Ethereum. Bytecode-oriented Ponzi scheme detection methods [11], [12] have made

* Corresponding Author.

S. Ji, C. Huang, P. Zhang, H. Dong and Y. Xiao, "Ponzi Scheme Detection Based on Control Flow Graph Feature Extraction," 2023 IEEE International Conference on Web Services (ICWS), Chicago, IL, USA, 2023, pp. 585-594, doi: 10.1109/ICWS60048.2023.00077.

significant progress in terms of detection performance. However, Ponzi scheme detection still suffers from the following challenges.

- *Bytecode complexity.* The processing of the bytecodes is challenging. Bytecodes are highly abstract and difficult to understand, with unique internal segmentation and concatenation. It is difficult to obtain complete function information from the bytecodes directly, since there are no structures such as functions and classes in the bytecodes and the bytecodes are highly overlapped. In addition, there is functionally unrelated information in the bytecode, which is difficult to identify. Existing detection methods [12], [13] usually take the opcode which is obtained by disassembling bytecode as continuous text to extract features for Ponzi scheme detection. Adjacent words are directly combined into a sequence after removing stop words in the opcode to construct the feature. This results in a lot of discontinuous operations in the bytecode being combined into a sequence and taken as the feature. However, such features are meaningless and affect the effectiveness of machine learning-based detection. In addition, these methods usually treat frequently appeared opcodes, such as *PUSH*, as stop words and remove them [14], [15]. However, these opcodes have operational meaning, especially when combined with other opcodes in a sequence. Handling bytecodes improperly may destroy the structural information of bytecode and produce misinformation when extracting features.
- *Data imbalance.* Among all the smart contracts on Ethereum, the difference between the numbers of Ponzi scheme and non-Ponzi scheme contracts is huge. Although this gap is artificially reduced in some datasets, it is far from the optimal 1:1 ratio. The imbalance between positive and negative samples may affect the performance of the machine learning model and cause poor *Recall* scores, which is evidently revealed in our experiments.

To address the above challenges in Ponzi scheme detection, we propose a novel Ponzi scheme detection method based on control flow graph feature extraction. It converts complex bytecode to opcode and constructs corresponding control flow graphs. Then n-gram TF (Term Frequency) and n-gram TF-IDF (Term Frequency-Inverse Document Frequency) features are extracted from the control flow graph. Finally, these features are respectively employed to construct a RF (Random Forest) model for Ponzi scheme detection. The main contributions are as follows:

- A control flow graph, which only contains function-related elements, is constructed for feature extraction. After translating the opcode into the control flow graph, the content and structural information are analyzed to identify and eliminate the elements that are not related to the function of the smart contract. In this way, meaningless features can be excluded.
- During the RF model construction for Ponzi scheme detection, the SVM_SMOTE (Support Vector Machine_Synthetic Minority Oversampling Technique) algorithm is used to balance the positive and negative features, which improves the performance of the detection model.
- Experimental evaluation is performed on a real-world dataset, by which 2-gram TF-IDF is selected as the feature type to perform Ponzi scheme detection. By comparing the control flow graph-based feature extraction method with the existing continuous text-based feature extraction method, and comparing the SVM_SMOTE-based RF model with four existing models, the experimental results demonstrate the superior performance of our method for Ponzi scheme detection, with *Precision*, *Recall*, *F1-score* and *AUC* reaching 95.96%, 92.23%, 94.06% and 95.84%.

The rest of this paper is organised as follows: Section II introduces the related work. Section III describes the detail of our method. Section IV presents the experimental evaluation. Finally Section V concludes the paper.

II. Related Work

Existing research about Ponzi scheme detection on blockchains can be divided into program analysis-based methods and data mining-based methods.

A. Program analysis-based methods

Bartoletti et al. [16] proposed a similarity measure-based Ponzi scheme detection method. They collected a set of Ponzi scheme contracts by manually analyzing the Solidity code of smart contracts. Then they used the NLD (Normalized Levenshtein Distance) [17] algorithm to measure the similarity between the bytecodes of two smart contracts to determine whether the bytecode of a contract is similar to that of some Ponzi schemes identified in the initial collection. Sun et al. [9] proposed a Ponzi scheme detection method based on contract behaviour similarity comparison. They used traditional software testing techniques to extract behavioural features, which are described as behavioural trees, in the execution of smart contracts. The smart contracts to be detected are compared with each of the collected Ponzi scheme contracts by using the AP-TED (All Path Tree Edit Distance) [18] algorithm to calculate the similarity between behavioural trees. Chen et al. [10] proposed a semantic-aware Ponzi scheme detection approach. They summarized the patterns of different kinds of Ponzi schemes and used semantic information extracted from the bytecode of a smart contract to match the defined Ponzi scheme patterns.

Program analysis-based methods are based on the pattern of existing Ponzi scheme contracts. As new Ponzi scheme patterns emerge or fraudsters deliberately add distractions, the accuracy of these methods will be affected. In addition, as the number of Ethereum smart contracts increases, the time cost of these methods for detecting Ponzi schemes will also increase.

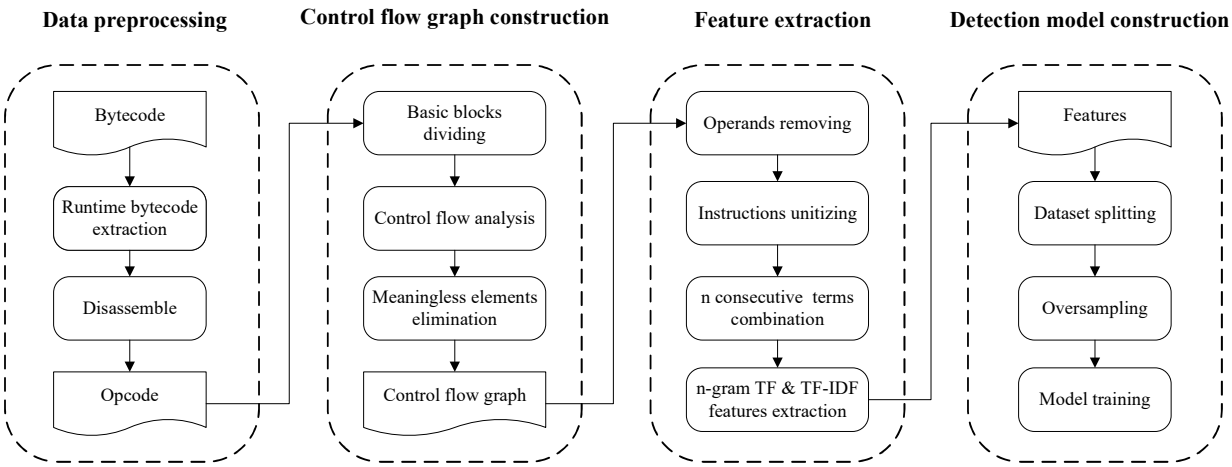


Fig. 1. Framework of the proposed Ponzi scheme detection method.

B. Data mining-based methods

Bartoletti et al. [19] used transaction features to detect Bitcoin Ponzi schemes. They extracted 11 transaction features, such as lifetime, Gini coefficient and maximum daily trading volume, to construct RIPPER, Bayes Network and RF for detecting Ponzi schemes. Chen et al. [20] proposed a detection method using code features and transaction features. They extracted the opcode frequencies and the Ether flow of contracts to construct the XGBoost model for Ponzi scheme detection. Fan et al. [12] used BOW (Bag of Words) [21] to extract n-gram features and used an ordered boosting algorithm to construct a Ponzi scheme detection model. The Borderline-SMOTE oversampling technology is used to synthesize more Ponzi schemes to balance the number of Ponzi and non-Ponzi schemes. Zhang et al. [22] extracted bytecode similarity, bytecode length and TF-IDF features from the smart contracts to construct the CatBoost model for Ponzi scheme detection. And they used the SMOTE_Tomek algorithm to perform data balance. Aljofey et al. [13] extracted four types of features from the contract opcodes, including opcode frequency, count vector, n-gram TF-IDF and opcode sequence features, for the training of a classification model.

Although the above data mining-based methods achieve positive results, most of them consider opcodes as continuous text and neglect the logic features of opcodes. This may result in redundant and meaningless features generated during sequence feature extraction, which affects the detection performance. Our method takes into account the logic features by constructing the control flow graph of the opcode in which functionally unrelated elements are identified and eliminated to extract more precise features.

III. Method

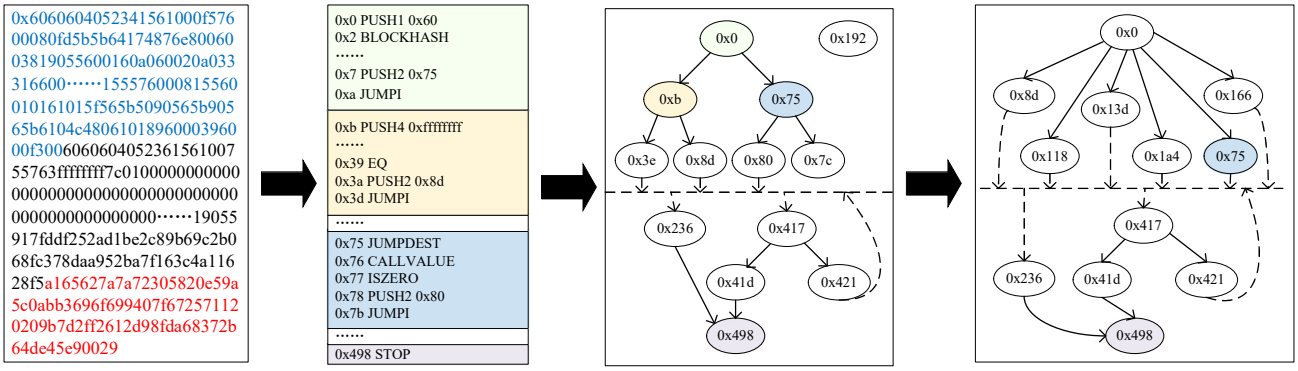
Figure 1 shows the framework of our method. Firstly, the runtime bytecode is extracted from the bytecode of a smart contract, which is consequently disassembled into the opcode. Secondly, a control flow graph is constructed based on the opcode, in which meaningless elements are identified and eliminated. Thirdly, the n-gram TF and TF-IDF features of the smart contract are extracted based on the control flow graph. Finally, an RF model is constructed for Ponzi Scheme detection, in which the feature set extracted from the collected dataset is balanced using SVM_SMOTE algorithm.

A. Data Preprocessing

The bytecode of a smart contract consists of three parts, namely, deployment bytecode, runtime bytecode and auxdata, as shown in blue, black and red parts respectively in Fig.2. The deployment bytecode, which contains information such as the address of the smart contract, is used to deploy the smart contract on the Ethernet platform. The auxdata, which is the encrypted data of the bytecode, is used for validation and will not be executed. Since these two types of data are not relevant to the execution of the smart contract, they are supposed to be removed from the bytecode for obtaining the runtime bytecode, which is the core of the bytecode. String matching is used to identify the deployment bytecode and auxdata from the bytecode. The deployment bytecode usually starts with “0x6060” or “0x6080”, and ends with “f3” or “f300”. The auxdata varies with Solidity versions, which usually starts with “a165”, “a264” or “a265” and ends with “29”, “32” or “33”. By matching the start and end strings, the deployment bytecode and auxdata can be identified and removed, with the runtime bytecode obtained.

For the extracted runtime bytecode, it is difficult to analyze the function information and the functionally unrelated information, since the bytecode is difficult to understand. Therefore, we convert the bytecode to the opcode for subsequent processing. The pyevmasm tool[1] is used to disassemble the bytecode into the opcode. The bytecode element and the opcode element are in a one-to-one relationship. For example, “0x60” is disassembled to *PUSH1* and “0xf3” is disassembled

[1]https://github.com/crytic/pyevmasm



(a) Bytecode (b) Opcode (c) Control flow graph (d) Control flow graph after elimination

Fig. 2. An example for illustrating the proposed Ponzi scheme detection method.

to *RETURN*. Table I shows the description of some important opcodes. Fig.2(b) shows the opcode corresponding to the sample bytecode of a smart contract.

B. Control flow graph construction

To identify the meaningless information and avoid discontinuous operations in the bytecode being combined as a feature, a control flow graph is constructed based on the opcode for feature extraction, making the internal structure of the smart contract become apparent. To construct the control flow graph, the opcode needs to be divided into basic blocks, for which the corresponding nodes will be created in the control flow graph. Then the control flow among the basic blocks needs to be analyzed, for which the edges will be established among corresponding nodes. Finally, nodes that correspond to meaningless blocks are identified and eliminated from the control flow graph.

1) Basic blocks dividing: The basic block is the basic functional unit in the opcode, in which no instruction changes the control flow. Each basic block starts with the first instruction of the opcode, or the *JUMPDEST* instruction, or the instruction following the previous basic block, and ends with the instruction before *JUMPDEST* or the instruction that changes the control flow, such as *JUMP, JUMPI, RETURN, STOP*, etc. The opcode is traversed to divide the basic blocks according to the above rules. For each basic block, a corresponding node is created in the control flow graph, for which the offset of the first instruction in the block is set as the identifier. For the opcode shown in Fig.2(b), the basic block in blue starts with *JUMPDEST* and ends with *JUMPI*, and the identifier of the corresponding node is 0x75.

2) Control flow analysis: With the divided basic blocks, it needs to analyze the control flow among them for establishing edges among corresponding nodes. The last instruction in the basic block determines which basic block will be executed subsequently. For the basic block A and its corresponding node n_A in the control flow graph, there are four cases as follows:

- If the last instruction of A is *JUMP*, the smart contract jumps unconditionally to the target address which is at the top of the EVM (Ethereum Virtual Machine) stack. It means the basic block B that corresponds to the target address is executed after A. Therefore, an edge is established between corresponding nodes n_A and n_B.
- If the last instruction of A is *JUMPI*, the smart contract jumps conditionally to the target address. If the condition is satisfied, the target basic block is the basic block B that corresponds to the address at the top of the EVM stack; otherwise, it is the basic block C next to A in the opcode. Edges $< n_A, n_B >$ and $< n_A, n_C >$ are established between corresponding nodes. As shown in Fig.2(c), the nodes 0xb and 0x75 are the target nodes of node 0x0 whose corresponding block ends with *JUMPI*.
- If the last instruction of A is of the termination type, including *STOP, RETURN, INVALID, REVERT and SELFDESTRUCT*, then there is no subsequent block and the control flow of the smart contract will terminate with the corresponding node.
- If the last instruction of A is neither jumping nor termination, then the smart contract will continue in the order of the offsets. Then, an edge will be established between n_A and n_B whose corresponding basic block B is the one next to A in the opcode.

When the last instruction of the basic block is *JUMP* or *JUMPI*, it requires identifying the target address. If the precedent instruction of *JUMP* or *JUMPI* is the *PUSH* instruction, the target address at the top of the EVM stack is obviously the operand of the *PUSH* instruction, since the *PUSH* instruction presses the operand to the top of the stack; otherwise, the target address is unknown. In this case, it is necessary to run the opcode to obtain the stack status. The opcode is simulated to run, in which a stack is built to record the status of the execution. To reduce the cost of execution, the operation manipulation method proposed in [23] is used in the execution, in which only the instructions related to the jump address are executed with the real operands considered, such as *PUSH,*

TABLE I
DESCRIPTION OF SOME IMPORTANT OPCODES

Bytecode	Opcode	Description
0x03	SUB	Subtract the top two stack items
0x10	LT	Less-than comparison
0x15	ISZERO	Simple not operator
0x16	AND	Bitwise AND operation
0x33	CALLER	Push the caller address (msg.caller) to the top of the stack
0x34	CALLVALUE	Push msg.value(Ether sent to smart contracts) to the top of the stack
0x35	CALLDATALOAD	Push the first 32 bytes of CALLDATA(msg.data) to the top of the stack
0x36	CALLDATASIZE	Return the size of msg.data (data information in Ethernet transactions)
0x50	POP	Pop an element from the top of the stack
0x52	MSTORE	Write a data of (u)int256 to memory
0x54	SLOAD	Load data from storage
0x56	JUMP	Unconditional jump
0x57	JUMPI	Conditional jump
0x5b	JUMPDEST	Mark an address that can jump to
0x60-0x7f	PUSHX	Push X byte elements to the top of the stack
0x80-0x8f	DUPX	Duplicate the Xth element from the stack and push the element to the stack
0xf1	CALL	Message-call into an account
0xf3	RETURN	Halt execution and return output data

DUP. For those irrelevant instructions, operands are set with "unknown" since only the times of push and pop operations are required. By selecting the value at the top of the stack as the desired target address, we can locate the specific basic block associated with that address. This identification process allows us to create an edge between the relevant nodes in the control flow graph. Fig.2(c) shows the control flow graph fragment of the example.

3) Meaningless elements elimination: In the constructed control flow graph, there are some elements that are functionally unrelated, which are called meaningless elements. When conducting feature extraction using the control flow graph while preserving meaningless elements, there is a possibility that the extracted features may contain impurities. This can result in inaccurate detection outcomes. Eliminating the meaningless elements can improve the effectiveness of the feature extraction without affecting the functionality of the smart contract. The meaningless elements in the control flow include isolated nodes and the nodes corresponding to dispatchers and the fallback function.

- Isolated node. For a smart contract, its execution paths correspond to the paths starting with node 0x0 in its control flow graph. So if the node is not contained in any execution paths of the control flow graph, it is functionally unrelated and regarded as an isolated node.
- Dispatcher. A dispatcher in the opcode is the entrance of a function in a smart contract [23]. During the program execution, the dispatcher is used to check the function signature, which is obtained by taking the function name, variable type and brackets as input to the KECCAK-256 algorithm [24] and extracting the first four bytes of the KECCAK-256 encoding, to determine the specific function to be executed. The dispatcher manages the entry of a function at the beginning of the opcode without affecting the functionality of the smart contract.
- Fallback function. A fallback function is a special function in a smart contract, which has no function name, arguments, or return value [24]. There is no corresponding function signature in the opcode. The fallback function is executed if there is no function that matches the called function signature or if Ether is sent to the contract in which there is no receive function for Ether. In some smart contracts, the fallback function is not used to process transaction data. The fallback function in Fig.3, which only contains *throw* statement, is used to throw exceptions. Such fallback functions are meaningless.

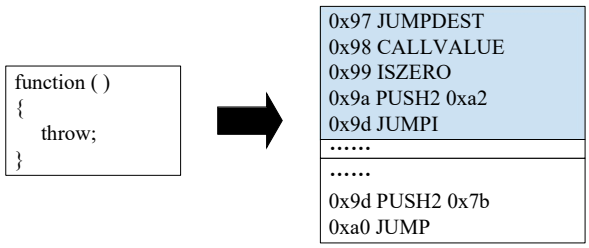
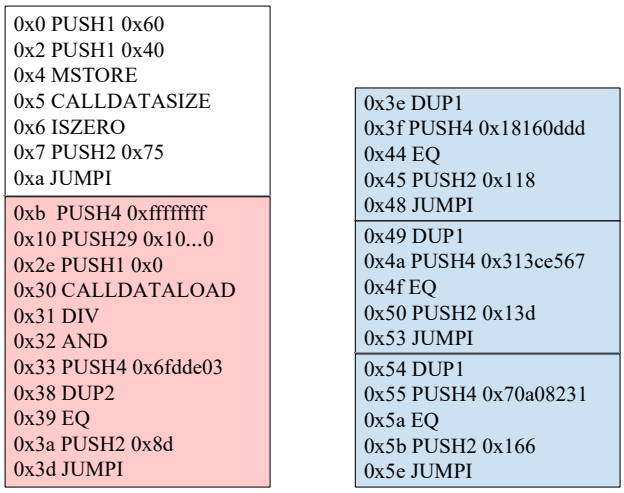


Fig. 3. An example of the fallback function.

To eliminate isolated nodes, the control flow graph is traversed in a depth-first manner starting from node 0x0. If the nodes are not visited in the traversal, they are identified as isolated nodes and eliminated from the control flow graph. As shown in Fig.2, node 0x192 is an isolated node.

To eliminate the nodes that correspond to dispatchers, we need to address two distinct situations as there are two types of dispatchers in the bytecode [25].

- The first type of dispatcher is located in the second block in the opcode. As shown in Fig.4(a), the basic block in red is an example of the first type of dispatcher. The *CALLDATALOAD* instruction at 0x30 reads the first 32 bytes of the input data of the transaction, which is compared with the signature 0x6fdde03 at 0x33 by *EQ* instruction. If the two groups of data are equal, it will jump to the block whose offset is 0x8d. To detect the first type of dispatcher, we examine the node in the control flow graph whose identifier is equal to the offset of the last instruction in the start node plus 1. If



(a) The first type

(b) The second type

Fig. 4. Two types of dispatchers.

the block corresponding to the node ends with *JUMPI* instruction that depends on the *EQ* instruction, then the node corresponds to such a dispatcher and is eliminated from the control flow graph.

- The second type of dispatcher contains five instructions and is usually located after the dispatcher of the first type. As shown in Fig.4(b), the basic blocks in blue are dispatchers of the second type. The element from the top of the stack is duplicated and compared with the function signature. If the two groups of data are equal, it will jump to the corresponding function. To identify the second type of dispatchers, each node in the control flow graph is checked. If the node contains such five instructions, it corresponds to such a dispatcher and is eliminated.

For the elimination of nodes corresponding to the fallback function, it requires to identify these nodes first. The fallback function consists of multiple blocks, which locate after the last dispatcher block and before the first function block in the opcode. During the identification of nodes corresponding to dispatchers, the offsets of the last dispatcher i_1 and the first function block i_2 are recorded. In the control flow graph, if the identifier of the node i satisfies $i_1 < i < i_2$, the node is considered as a part of the fallback function. With all the nodes that correspond to the fallback function being identified, they are further analyzed to determine whether the fallback function is meaningless. It is found that there usually exists the block, as shown in blue in Fig.3, which is used to check if Ether is sent to the contract, in the corresponding blocks of the fallback function. If there is no other *CALL*-related block, the fallback function is considered as meaningless. Each node n_i ($i_1 < i < i_2$) is checked. If there is no node corresponding to other *CALL*-related blocks, these nodes are regarded as meaningless and eliminated from the control flow graph.

As shown in Fig.2(d), the final control flow graph is obtained after eliminating the meaningless nodes in Fig.2(c), including isolated node 0x192 and nodes 0xb and 0x33 that correspond to dispatchers.

C. Feature extraction

Based on the control flow graph, two types of smart contract features are respectively extracted for Ponzi scheme detection, including n-gram TF and n-gram TF-IDF. These two types of features are widely used in Ponzi scheme detection [12], [26].

The n-gram TF feature is the frequency of the sequence which is combined by n consecutive terms in the smart contract's opcode. For extracting n-gram TF feature from the opcode, the operands need to be removed from each instruction and the instructions of the same type need to be unitized. For example, *DUP1*, *DUP2*, ..., *DUP16* are unitized as *DUP*, since only the operational semantics are focused in Ponzi scheme detection. After performing the above processing on blocks corresponding to all the nodes in the control flow graph, n consecutive terms in each block are combined and recorded. Then the frequency of each n-gram in all the blocks is computed.

TF-IDF, which is commonly used in natural language processing, assesses the importance of words within a specific document within a collection of documents. In Ponzi scheme detection, TF-IDF is used to evaluate the importance of n-grams within a smart contract within the dataset. The more the n-gram appears in the opcode of a smart contract, the more it can represent the logic of the smart contract; the more the n-gram appears in all the opcodes, the less representative it is [22]. The formula of TF-IDF for the *i*th n-gram term is shown as follows [27]:

$$TF\text{–}IDF_i = TF_i * IDF_i$$
$$IDF_i = log(\frac{m+1}{n(i)+1}) + 1$$

where TF_i is the frequency of the *i*th n-gram, m is the total number of smart contracts in the dataset, and $n(i)$ is the number of smart contracts that contain the *i*th n-gram. The TF-IDF value of the n-gram within a smart contract is determined by the whole dataset.

With the increase of the gram length, the detection accuracy is generally higher, but the memory consumption increases [26]. Therefore, only a gram whose length is less than 4 is considered. The n-gram TF feature and n-gram TF-IDF feature ($n \leq 3$) will be explored to determine which feature performs best in Ponzi scheme detection.

Compared with feature extraction based on continuous text, extracting n-gram TF features and n-gram TF-IDF features based on the control flow graph can obtain more precise results. This is due to the fact that combing n consecutive terms in the continuous text may generate meaningless features. For example, given two adjacent instructions *JUMP* and *JUMPDEST* that cannot be executed at the same time, continuous text-based feature extraction will generate *JUMP_JUMPDEST* 2-gram. However, the 2-gram is meaningless. With the control flow graph-based feature extraction,

there is no *JUMP_JUMPDEST* 2-gram since they are separated into two different blocks.

D. Detection model construction

With the extracted features of the smart contracts in the dataset, the Ponzi scheme detection model is constructed on top of an RF model [28]. RF is an ensemble machine learning model based on decision trees, which constructs multiple decision trees and combines their predictions to obtain more accurate predictions. RF models have been demonstrated to be effective in Ponzi scheme detection [29].

In machine learning-based Ponzi scheme detection, the ratio of Ponzi to non-Ponzi schemes in the training dataset is unbalanced. This imbalance can result in a disproportionate focus on the non-Ponzi scheme class and the potential neglect of the Ponzi scheme class. To alleviate the problem of class imbalance in a dataset, the SVM_SMOTE oversampling algorithm [30], which is an improved algorithm of SMOTE, is used to generate new samples of Ponzi scheme features. It constructs an SVM classifier with the original training dataset to obtain the classification boundary for the Ponzi and non-Ponzi samples. Then it generates new samples of Ponzi schemes located near this boundary, which are critical for estimating the optimal classification boundary. With the oversampling algorithm, the training dataset that consists of the same number of Ponzi and non-Ponzi samples is constructed for the forthcoming RF model training.

IV. Experimental Evaluation

In this section, an empirical study is performed to evaluate the effectiveness of the proposed Ponzi scheme detection method with a constructed dataset.

A. Experimental Setup

The experimental setup includes the following three aspects: Experimental dataset, evaluation metrics, and research questions.

1) Experimental Dataset: The dataset of smart contracts used in this paper is collected from three open source datasets, [29], [31] and XBlock[2]. Firstly, the dataset in Ref. [29], which contains a total of 3,788 smart contracts, including 200 Ponzi scheme contracts and 3,588 non-Ponzi scheme contracts, is chosen. To address the significant disparity between the quantities of positive and negative samples, we collect additional 167 Ponzi scheme contracts from the dataset referenced in [31] and another 180 Ponzi scheme contracts from XBlock. Eventually, the dataset consists of 547 Ponzi scheme contracts and 3,588 non-Ponzi scheme contracts, which is 4,135 smart contracts in total. We randomly divide the constructed dataset into 80% as the training set and 20% as the testing set. During the model training process, we perform parameter optimization to identify the most suitable values for crucial parameters. Each experiment is conducted ten times with the optimal model parameters.

[2]http://xblock.pro/

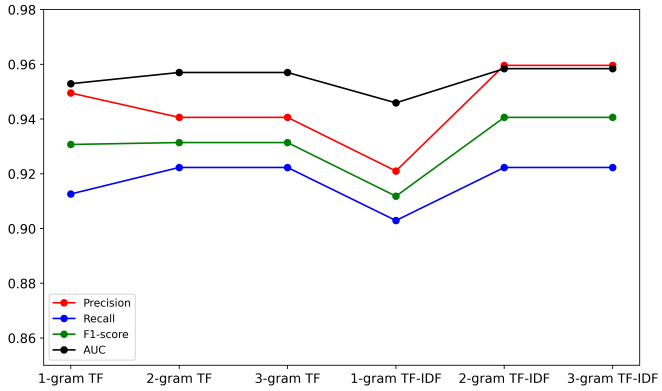


Fig. 5. Performance of six types of features in Ponzi scheme detection.

2) Evaluation metrics: The performance of the proposed method is evaluated upon *Precision*, *Recall*, *F1-score* and *AUC*.

Let TP denote the number of Ponzi scheme contracts that are correctly detected, FP denote the number of non-Ponzi scheme contracts that are misclassified as Ponzi scheme related, and FN denote the number of Ponzi scheme contracts that are misclassified as non-Ponzi scheme related, the evaluation metrics are described as follows.

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

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

$$F1\text{–}score = \frac{Precision * Recall}{Precision + Recall} * 2$$

AUC represents the probability that a positive sample has a higher estimated probability of belonging to the positive class than a negative sample, which is a more appropriate performance metric than accuracy [32].

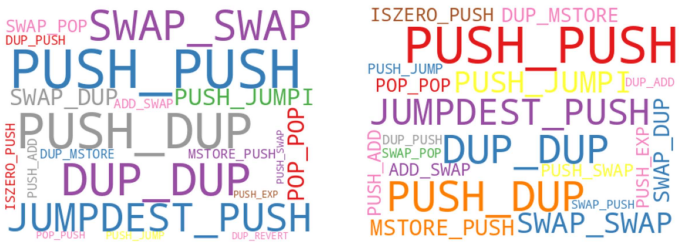
For each of the evaluation metrics, the higher the value is, the better the detection model performs.

3) Research questions: The following research questions are specified to evaluate the proposed method:

- *RQ1: Feature Selection.* Between the n-gram TF features and the n-gram TF-IDF features ($n \leq 3$), which performs better for Ponzi scheme detection?
- *RQ2: Performance of Feature Extraction Method.* How does the control flow graph-based feature extraction method perform in comparison with the existing feature extraction method?
- *RQ3: Performance of Detection Model.* How does the SVM_SMOTE based RF model perform in comparison with the existing detection models?

B. Feature Selection

To select the feature type that has better Ponzi scheme detection performance, we respectively extract n-gram TF features and n-gram TF-IDF features ($n \leq 3$) based on the control flow graph of a smart contract to construct the RF models for Ponzi scheme detection.



(a) Ponzi scheme (b) non-Ponzi scheme

Fig. 6. The 2-gram TF-IDF feature cloud in Ponzi scheme and non-Ponzi scheme contracts.

For the smart contracts in the collected dataset, 67 1-gram TF features, 805 2-gram TF features and 4,349 3-gram TF features are obtained. In addition, the same number of corresponding n-gram TF-IDF features ($n \leq 3$) are obtained. The extracted features of the six types of the dataset are respectively used to construct and test the Ponzi scheme detection models.

Fig.5 shows the performance of the RF based Ponzi scheme detection models constructed with six types of features. Among the six models, the models constructed with 2-gram TF-IDF features and 3-gram TF-IDF features perform best in terms of the four evaluation metrics. It is worth noting that constructing the detection model with 3-gram TF-IDF features costs more time since the number of 3-gram TF-IDF features is much larger than that of 2-gram TF-IDF features. Therefore, 2-gram TF-IDF is chosen as the feature type to perform Ponzi scheme detection.

With 2-gram TF-IDF as the feature type, we conduct statistical analysis on the numbers of different 2-gram TF-IDF features in Ponzi scheme and non-Ponzi scheme contracts of the dataset. Fig.6 shows the cloud graphs of the 2-gram TF-IDF features for Ponzi scheme and non-Ponzi scheme contracts, in which the top 20 features are extracted for comparison. Each word is a 2-gram sequence with its font size representing its average frequency. In both the Ponzi scheme and non-Ponzi scheme contracts, *PUSH_PUSH, PUSH_DUP* and *DUP_DUP* are the most frequent 2-gram sequences. It indicates that these three 2-gram sequences commonly exist in all smart contracts since all the computations are performed based on the stack. The rest of the features are ranked differently. For example, *POP_PUSH* and *DUP_REVERT* only exist in the top 20 list of the Ponzi scheme contracts, while *SWAP_PUSH* and *DUP_ADD* only exist in the top 20 list of the non-Ponzi scheme contracts.

Answer to RQ1: Among the six types of features, the 2-gram TF-IDF feature performs best in Ponzi scheme detection.

C. Performance of Feature Extraction Method

According to the finding for RQ1, 2-gram TF-IDF features are extracted based on the control flow graph of a smart contract with meaningless elements eliminated. To evaluate the proposed feature extraction method, we extract 2-gram TF-IDF features of smart contracts using the following methods. Then the features extracted by these extraction methods are respectively used to construct the RF models for Ponzi scheme detection.

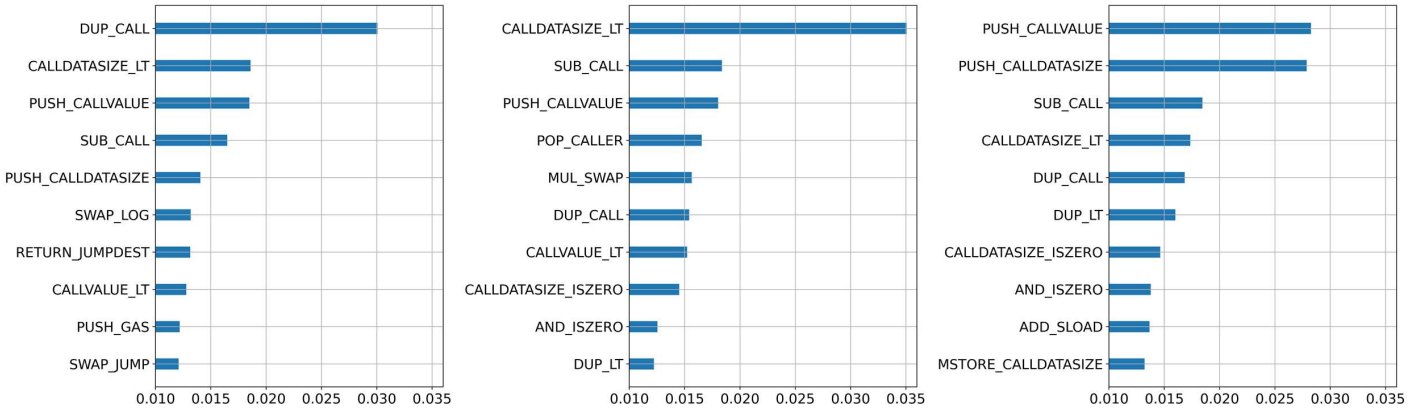
- *M1*. The features are extracted based on the opcode of a smart contract directly by treating the opcode as continuous text, which is commonly used in Ponzi scheme detection [15], [29].
- *M2*. The features are extracted based on the original control flow graph of a smart contract in which the functionally unrelated elements are not eliminated.

TABLE II
PERFORMANCE OF DIFFERENT FEATURE EXTRACTION METHODS IN PONZI SCHEME DETECTION.

Method	Precision	Recall	F1-score	AUC	Number
M1	0.9400	0.9126	0.9261	0.9522	1633
M2	0.9495	0.9126	0.9307	0.9529	1353
Our	**0.9596**	**0.9223**	**0.9406**	**0.9584**	**805**

Table II shows the performance of the RF based Ponzi scheme detection models with different feature extraction methods, in which the number of extracted 2-gram TF-IDF features is shown in the last column. The feature extraction method *M2* performs better than *M1* in terms of the four evaluation metrics. The main reason is that, compared with the text-based feature extraction method, extracting features based on control flow graphs can reduce the meaningless features. The number of 2-gram TF-IDF features decreases from 1633 to 1353 by using *M2* rather than *M1*. By eliminating the functionally unrelated elements in control flow graphs, the number of 2-gram TF-IDF features decreases from 1353 to 805, and its performance is further improved on all the metrics, since more precise features are extracted. The experimental results show that both the control flow graph construction and the meaningless element elimination have positive impacts on Ponzi scheme detection.

To further analyze the feature extraction methods, we list the ten most significant 2-gram TF-IDF features respectively learned by the RF models using the three feature extraction methods in Fig.7. The description of the related opcodes can be found in Table I. Among them, *PUSH_CALLVALUE*, *SUB_CALL*, *CALLDATASIZE_LT* and *DUP_CALL* are shown in the top six significant features of all the models, which indicates these four features play an important role in Ponzi scheme detection. In *M1* based model, *RETURN_JUMPDEST* is in the top ten significant features. However, *RETURN_JUMPDEST* is a meaningless feature since *JUMPDEST* instruction cannot be executed after *RETURN* instruction. Compared with the feature extraction method *M2*, our method eliminates 548 features, in which the importance value of 488 features is 0. This indicates that most of the features eliminated by our method lack significance when distinguishing between Ponzi scheme and non-Ponzi



(a) M1 (b) M2 (c) Our

Fig. 7. The importance of the ten most significant features.

TABLE III
PERFORMANCE OF DIFFERENT MODELS IN PONZI SCHEME DETECTION.

Method	Precision	Recall	F1-score	AUC
XGBoost	0.9293	0.8932	0.9109	0.9418
RF	0.9468	0.8641	0.9036	0.9286
LightGBM	0.9479	0.8835	0.9146	0.9383
CatBoost	0.9300	0.9029	0.9163	0.9466
Our	**0.9596**	**0.9223**	**0.9406**	**0.9584**

scheme contracts.

Answer to RQ2: The control flow graph-based feature extraction method performs better than the existing feature extraction method in Ponzi scheme detection.

D. Performance of Detection Model

To evaluate the performance of SVM_SMOTE based RF model, the following existing methods are chosen to perform comparative experiments:

- *XGBoost*: A method [20] that constructs a XGBoost (Extreme Gradient Boosting) [33] classification model for Ponzi scheme detection.
- *RF*: A method [29] that constructs a RF model for Ponzi scheme detection, in which no oversampling algorithm is used.
- *LightGBM*: A method [15] that constructs a LightGBM model [34] for Ponzi scheme detection, in which a mixed sampling method called SMOTE_Tomek is used.
- *CatBoost*: A method [22] that constructs a CatBoost model [35] for Ponzi scheme detection, in which the SMOTE_Tomek algorithm is used.

With the same dataset and the same 2-gram TF-IDF features extracted by the proposed feature extraction method, we reproduce the above four methods for comparative experiments. Table III shows the performance of the five models in Ponzi scheme detection. The following findings can be made from the experimental results. Firstly, our method has the best performance in terms of the four evaluation metrics, which indicates its effectiveness to detect Ponzi schemes. Secondly, compared with *RF*, our method exceeds 1.35%, 6.74%, 4.09% and 3.21% in *Precision*, *Recall*, *F1-score* and *AUC*, which indicates the effectiveness of the SVM_SMOTE algorithm to enhance the Ponzi scheme detection performance of the RF model.

Answer to RQ3: Compared with the existing four models, the proposed SVM_SMOTE based RF model has the best performance for Ponzi scheme detection.

V. CONCLUSION

This paper presented a Ponzi scheme detection method for bytecode-based smart contracts. For a machine learning-based detection method, on the one hand, it is difficult to extract features of smart contracts based on the bytecode directly due to the bytecode complexity; on the other hand, the model performance will be significantly affected by the imbalance problem of Ponzi scheme and non-Ponzi scheme contracts that exists among all Ethereum smart contract datasets. In this regard, we proposed a control flow graph-based Ponzi scheme detection method. After disassembling the bytecode of a smart contract into opcode, a control flow graph is generated for the opcode, in which the non-function-related elements are identified and eliminated. Then n-gram TF and TF-IDF features are extracted and used to construct the RF-based Ponzi scheme detection model in which the SVM_SOMTE algorithm is used to balance the proportion of positive and negative samples.

A dataset comprising 547 Ponzi scheme contracts and 3,588 non-Ponzi scheme contracts was collected to perform the experimental evaluation. The experimental results show that the 2-gram TF-IDF performs best in Ponzi scheme detection among the six types of features, including the n-gram TF features and n-gram TF-IDF features ($n \leq 3$). In addition,

the control flow graph-based feature extraction method outperforms the continuous text-based feature extraction method, and the SVM_SMOTE-based RF model outperforms the four existing models, including *XGBoost*, *RF*, *LightGBM* and *CatBoost*. With 805 2-gram TF-IDF features extracted from the dataset, our method achieves 95.96% *Precision*, 92.23% *Recall*, 94.06% *F1-score* and 95.84% *AUC*.

In future work, we will extract the other types of the features, such as text vector and structural features, to construct a Ponzi scheme detection model with better detection performance. In addition, we will expand the dataset by including more smart contracts for large-scale experiments. We will also carry out empirical studies with more machine learning or deep learning-based methods.

ACKNOWLEDGMENTS

The work is supported by the Fundamental Research Funds for the Central Universities of China under Grant No.B220202072, the National Natural Science Foundation of China under Grant No.U21B2016, No.62272145 and No.61702159, the CloudTech RMIT Green Bitcoin Joint Research Program/Laboratory, and the Cooperative Research Centres Projects (CRC-P) funding scheme "Fast and Secure Crypto Payments for E-Commerce Merchants" (CRCPXIII000145).

REFERENCES

[1] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International journal of web and grid services*, vol. 14, no. 4, pp. 352–375, 2018.

[2] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *2016 IEEE 18th international conference on e-health networking, applications and services (Healthcom)*. IEEE, 2016, pp. 1–3.

[3] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and sustainable energy reviews*, vol. 100, pp. 143–174, 2019.

[4] "Blockchain as a service." [Online]. Available: https://en.wikipedia.org/wiki/Blockchain_as_a_service

[5] J. Singh and J. D. Michels, "Blockchain as a service (baas): Providers and trust," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2018, pp. 67–74.

[6] "Ethereum Whitepaper." [Online]. Available: https://ethereum.org/en/whitepaper/

[7] B. K. Mohanta, S. S. Panda, and D. Jena, "An overview of smart contract and use cases in blockchain technology," in *2018 9th international conference on computing, communication and networking technologies (ICCCNT)*. IEEE, 2018, pp. 1–4.

[8] CHAINALYSIS, "The Chainalysis 2022 Crypto Crime Report," 2022. [Online]. Available: https://go.chainalysis.com/2022-crypto-crime-report.html

[9] W. Sun, G. Xu, Z. Yang, and Z. Chen, "Early detection of smart ponzi scheme contracts based on behavior forest similarity," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2020, pp. 297–309.

[10] W. Chen, X. Li, Y. Sui, N. He, H. Wang, L. Wu, and X. Luo, "Sadponzi: Detecting and characterizing ponzi schemes in ethereum smart contracts," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 2, pp. 1–30, 2021.

[11] X. Shen, S. Jiang, and L. Zhang, "Mining bytecode features of smart contracts to detect ponzi scheme on blockchain," *Computer Modeling in Engineering & Sciences*, vol. 127, no. 3, pp. 1069–1085, 2021.

[12] S. Fan, S. Fu, H. Xu, and C. Zhu, "Expose your mask: smart ponzi schemes detection on blockchain," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

[13] A. Aljofey, Q. Jiang, and Q. Qu, "A supervised learning model for detecting ponzi contracts in ethereum blockchain," in *Big Data and Security: Third International Conference, ICBDS 2021, Shenzhen, China, November 26–28, 2021, Proceedings*. Springer, 2022, pp. 657–672.

[14] L. Wang, H. Cheng, Z. Zheng, A. Yang, and X. Zhu, "Ponzi scheme detection via oversampling-based long short-term memory for smart contracts," *Knowledge-Based Systems*, vol. 228, p. 107312, 2021.

[15] Y. Zhang, W. Yu, Z. Li, S. Raza, and H. Cao, "Detecting ethereum ponzi schemes based on improved lightgbm algorithm," *IEEE Transactions on Computational Social Systems*, vol. 9, no. 2, pp. 624–637, 2021.

[16] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, "Dissecting ponzi schemes on ethereum: identification, analysis, and impact," *Future Generation Computer Systems*, vol. 102, pp. 259–277, 2020.

[17] L. Yujian and L. Bo, "A normalized Levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[18] E. Duesterwald, C. Cascaval, and S. Dwarkadas, "Characterizing and predicting program behavior and its variability," in *2003 12th International Conference on Parallel Architectures and Compilation Techniques*. IEEE, 2003, pp. 220–231.

[19] M. Bartoletti, B. Pes, and S. Serusi, "Data mining for detecting bitcoin ponzi schemes," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 75–84.

[20] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting ponzi schemes on ethereum: Towards healthier blockchain technology," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 1409–1418.

[21] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International journal of machine learning and cybernetics*, vol. 1, pp. 43–52, 2010.

[22] Y. Zhang, S. Kang, W. Dai, S. Chen, and J. Zhu, "Code will speak: Early detection of ponzi smart contracts on ethereum," in *2021 IEEE International Conference on Services Computing (SCC)*. IEEE, 2021, pp. 301–308.

[23] F. Contro, M. Crosara, M. Ceccato, and M. Dalla Preda, "Ethersolve: Computing an accurate control-flow graph from ethereum bytecode," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 127–137.

[24] "Solidity Docs." [Online]. Available: https://docs.soliditylang.org/en/v0.8.19/

[25] X. Li, T. Chen, X. Luo, T. Zhang, L. Yu, and Z. Xu, "Stan: Towards describing bytecodes of smart contract," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2020, pp. 273–284.

[26] S. Fan, S. Fu, H. Xu, and X. Cheng, "Al-spsd: Anti-leakage smart ponzi schemes detection in blockchain," *Information Processing & Management*, vol. 58, no. 4, p. 102587, 2021.

[27] G. Salton and C. T. Yu, "On the construction of effective vocabularies for information retrieval," *Acm Sigplan Notices*, vol. 10, no. 1, pp. 48–60, 1973.

[28] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[29] W. Chen, Z. Zheng, E. C.-H. Ngai, P. Zheng, and Y. Zhou, "Exploiting blockchain data to detect smart ponzi schemes on ethereum," *IEEE Access*, vol. 7, pp. 37 575–37 586, 2019.

[30] H. M. Nguyen, E. W. Cooper, and K. Kamei, "Borderline over-sampling for imbalanced data classification," *International Journal of Knowledge Engineering and Soft Data Paradigms*, vol. 3, no. 1, pp. 4–21, 2011.

[31] X. He, T. Yang, and L. Chen, "Ctrf: Ethereum-based ponzi contract identification," *Security and Communication Networks*, vol. 2022, 2022.

[32] Z. Yang, Q. Xu, S. Bao, X. Cao, and Q. Huang, "Learning with multiclass auc: theory and algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7747–7763, 2021.

[33] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[34] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[35] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.