PonziFinder: Attention-Based Edge-Enhanced Ponzi Contract Detection

Yingying Chen, Bixin Li, Yan Xiao, and Xiaoning Du

***Abstract*—*Ponzi contracts* are fraudulent investment scams that promise high returns with little risk to investors. However, existing methods for detecting Ponzi contracts have several limitations. For example, they struggle to deal with the class imbalance problem, and their analysis of function call transactions is inadequate, resulting in redundant features. To tackle the challenges of detecting Ponzi contracts, we present PonziFinder, a novel approach that leverages convolutional-based edge-enhanced graph neural network and attention mechanism for the classification of contract transaction graphs. In contrast to previous methods, we not only consider transaction value and timestamp but also analyze transaction input to standardize and sort transactions. We extract node and edge features that capture the unique characteristics of Ponzi contracts. The edge feature, reflecting interaccount correlation, enhances the propagation and updating of node features for effective Ponzi contract detection. To prevent oversmoothing of node embedding caused by the shallow transaction graph and extract important account node information, we introduce an attention-based global layerwise aggregation mechanism (ALGA) for generating the final contract graph representation for classification. Moreover, we optimize the node feature set and use an effective strategy based on undersampling and ensemble learning to address the issue of class imbalance. Experimental results show that PonziFinder can detect all types of Ponzi contracts (100%) with 97% accuracy when there is sufficient transaction data, outperforming other models. The analysis of input values and the ALGA mechanism are experimentally shown to improve accuracy by 4% and 2%, respectively. In summary, PonziFinder is a novel and effective method for detecting Ponzi contracts. Our approach addresses the limitations of existing methods and demonstrates significant improvements in accuracy and efficiency.**

***Index Terms*—Graph neural network (GNN), imbalanced data classification, Ponzi contracts, undersampling.**

I. Introduction

Ethereum is a decentralized, global software platform that operates on blockchain technology. Its ability to support smart contracts enables it to function as a programmable data-sharing platform [1]. Once a smart contract is deployed, it cannot be modified. Therefore, any errors or pitfalls in deployed smart contracts can cause cryptocurrency loss. The security issues of smart contracts are primarily divided into code security and contract security. Code security focuses on smart contracts that are vulnerable to attackers due to code or other designs, such as reentrancy [2], timestamp dependence [3], and others [4]. On the other hand, contract security focuses on fraudulent events introduced into the blockchain domain, such as Ponzi schemes [5] and Honeypot [6]. Given the increasing prevalence of traditional scams, contract security is in urgent need of being addressed.

A Ponzi scheme is a classic form of fraud that has migrated to the blockchain ecosystem in smart contracts. Due to the close-source nature of most smart contracts, stakeholders find it challenging to comprehend the execution logic of smart contracts thoroughly. The unique execution mechanism of smart contracts offers numerous advantages to contract creators, including no maintenance costs after the implementation of smart contracts, operator anonymity, and stakeholder confidence in continuous repayment because smart contracts are automatically executed and cannot be terminated [7]. Shockingly, statistics reveal that Ponzi schemes operating through Bitcoin raised over $7 million between September 2013 and September 2014 [5]. The emergence of Ponzi contracts has resulted in significant losses to stakeholders and poses a severe threat to blockchain security.

To address this issue, researchers have proposed three kinds of methods to detect Ponzi contracts as follows.

1) *Code Feature-Based Methods:* mainly focus on analyzing Ponzi contracts with representative structures and logic from the perspective of source code or bytecode using machine learning [8], [9], [10], [11] and and other traditional methods [12], [13], [14], [15].
2) *Transaction Feature-Based Methods:* use transaction information for Ponzi contract detection, commonly employing transaction network analysis using GNNs [16] or analyzing transaction data by calculating the Gini coefficient [17].
3) *Combined Feature-Based Methods:* consider the combination of code features and transaction features, typically extracting code features by calculating the opcode frequency, extracting transaction features based on historical contract transaction information, and using machine learning methods for automatic detection [7], [18], [19], [20]. In addition, some works also combine bytecode features to improve detection accuracy [21].

Manuscript received 3 May 2023; revised 18 September 2023 and 18 December 2023; accepted 21 February 2024. This work was supported by the Key Research and Development Program of Jiangsu Province under Grant BE2021002-3. Associate Editor: H. K. T. Ng. *(Corresponding author: Bixin Li.)*

Yingying Chen and Bixin Li are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China (e-mail: bx.li@seu.edu.cn).

Yan Xiao is with the School of Cyber Science and Technology, Shenzhen Campus of Sun Yat-sen University, Shenzhen 518107, China.

Xiaoning Du is with Monash University, Clayton, VIC 3800, Australia.

Digital Object Identifier 10.1109/TR.2024.3370734

However, code feature-based methods can only target Ponzi contracts with representative structures and logics, *making it challenging to deal with contracts that vary greatly in code implementation*. Combined feature-based methods utilize code features and transaction features in a simple way, *neglecting the structural information of the transaction network*. In contrast, transaction feature-based methods often analyze historical transaction information of contracts and effectively detect Ponzi contracts. Recent studies attempt to construct transaction flow graphs using historical transaction information of contracts and extract suitable node features based on GNN algorithms for classification. However, they lack in-depth mining and analysis of transaction information and have some drawbacks, 1) *including redundant and inadequate extracted features*, for example, five features of minimum, maximum, sum, mean, and variance are extracted only for the amount of income, 2) *only taking edge as the transmission medium of information*, and 3) *ignoring the importance of edges in the transaction network*. Moreover, the large difference in the number of Ponzi contracts and non-Ponzi contracts makes it challenging to solve *the imbalanced data classification problem*, which remains a focus of current research. Therefore, Ponzi contract detection methods still need improvement to cope with undiscovered Ponzi contracts with diverse implementations.

To address the limitations of existing methods, we propose a novel method called PonziFinder, which models the detection problem as a graph classification task and starts by crawling and preprocessing the historical transaction data for each contract in the dataset. *Unlike existing methods that rely on code implementation, PonziFinder considers only real transaction records, allowing for more accurate detection by analyzing the topology of the transaction network*. We extract node and edge features based on the unique characteristics of Ponzi contracts and construct a contract transaction graph using contract and external accounts as nodes and transactions as connecting edges. *We also select an optimal node feature set to improve model efficiency*. Our observation of Ponzi contract transactions reveals that 68.5% of them only call functions with no amount, and their originators are often Ponzi contract beneficiaries, such as the contract creator. *We extract this information as an edge feature to enhance account node features, as beneficiaries are more likely to be associated with Ponzi contracts*. We then use convolutional-based edge-enhanced graph neural network (EGNN(C)) to propagate and update node information through transaction edges. In each convolution layer, we apply Set2Set to aggregate informative node representations using attention. The resulting representations are then fed into the attention-based global layerwise aggregation mechanism (ALGA) to obtain the final graph representation for classification. *To address the class imbalance problem, we provide single-model and multimodel approaches based on undersampling and ensemble learning.*

The contributions of this article are summarized as follows.

1) We analyze the input, timestamp, and value data of the contract transactions and perform specialized preprocessing tailored to the characteristics of Ponzi contracts. This preprocessing improves the quality of the data and enables the model to better identify Ponzi contracts.
2) We distinguish between transfer and nontransfer transactions and calculate their sums as transaction edge features, which enhances the account node information and improves the model's ability to detect Ponzi contracts.
3) We present the design of PonziFinder, including the core EGNN(C) model with the Set2Set graph-level feature extraction module to extract global features of the smart contract graph. We also adopt the ALGA to overcome the node feature oversmoothing problem that arises from the shallow trading graphs. Our experimental results demonstrate that our model achieves the best overall performance, with precision exceeding 95%, and accuracy and F1-score exceeding 97%, while achieving a recall of 100%.

The rest of this article is organized as follows. In Section II, we give a brief introduction of some background knowledge. A detailed description of our method is presented in Section III. Experimental results and analysis are shown in Section IV. The power of PonziFinder is demonstrated in Section V through recent Ponzi contracts. We summarize existing work related to Ponzi contracts detection and compare them with our method in Section VI. Finally, Section VII concludes this article.

II. Background Knowledge

This section briefly introduces some key background knowledge involved in this study, including Ethereum and smart contracts, transaction and input data, Ponzi contracts, taxonomy of Ponzi schemes, and graph neural networks.

A. Ethereum and Smart Contract

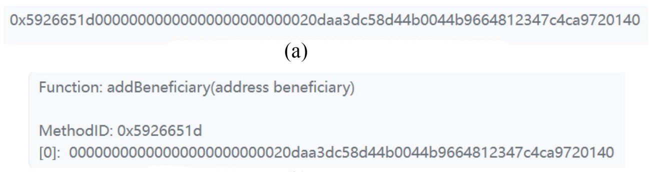
Ethereum is an open-source public blockchain platform that executes contract programs written in a Turing-complete bytecode language.

Currently, smart contracts in Ethereum are mainly written in *Solidity*, consisting of functions, user-defined modifiers, events, and data. To be executed in Ethereum virtual machine (EVM), the source code of a smart contract must be compiled into bytecode. Once deployed on the blockchain, the bytecode is transparent to all developers and the execution of the smart contract is fully compliant with the prerequisite logic and is not even interfered with by the creator. All methods can be called by anyone, except those marked as internal or private [2].

B. Transaction and Input Data

Users can send transactions to the Ethereum network in order to: 1) create a new contract; 2) invoke functions of the contract; 3) transfer ether to contracts or to other users [5].

Input data are a part of the transaction, representing the function and argument to the invoked smart contract. In particular, the input data used in the contract creation transaction represent the constructor argument. Generally, input data have a basic structure of function identifier and parameters, and its raw and parsed forms are shown in Fig. 1. Function identifier is the function selector, which is the first 4 B of the Keccak hash of a function signature, and is used as a unique identifier for the function. Parameters represent the parameter input of the



0x5926651d00000000000000000000000020daa3dc58d44b0044b9664812347c4ca9720140

(a)

Function: addBeneficiary(address beneficiary)

MethodID: 0x5926651d
[0]: 00000000000000000000000020daa3dc58d44b0044b9664812347c4ca9720140

(b)

Fig. 1. Two forms of a transaction input data. (a) Raw form of a transaction input data. (b) Parsed form of a transaction input data.

corresponding function, generally a parameter occupies 32 B, with the high bit complemented by zero.

C. Ponzi Contracts

A Ponzi scheme is a typical fraudulent investment activity named after the originator of a massive scam. U.S. Securities and Exchange Commission [22] defines Ponzi scheme fraud in general as "A Ponzi scheme is an investment fraud that involves the payment of purported returns to existing stakeholders from funds contributed by new stakeholders. Ponzi scheme organizers often solicit new stakeholders by promising to invest funds in opportunities claimed to generate high returns with little or no risk. With little or no legitimate earnings, Ponzi schemes require a constant flow of money from new stakeholders to continue. Ponzi schemes inevitably collapse, most often when it becomes difficult to recruit new stakeholders or when a large number of stakeholders ask for their funds to be returned."

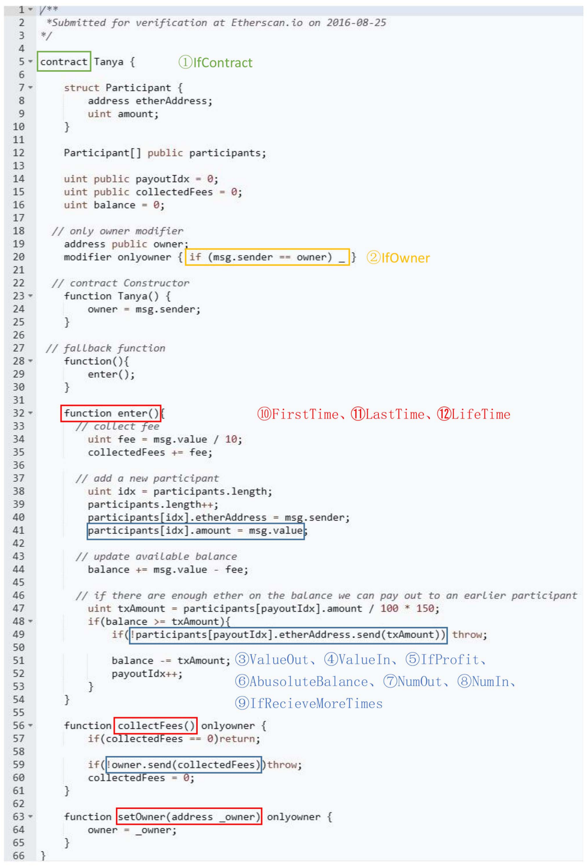
Today, many Ponzi schemes use smart contracts to disguise themselves because smart contracts are self-executing and cannot be terminated, which will mislead stakeholders into believing that they are guaranteed continuous repayment. More importantly, promoters can remain anonymous on the blockchain. We call these Ponzi schemes as *smart Ponzi schemes*, and the corresponding smart contract as *Ponzi contracts*.

Fig. 2 shows an example of a Ponzi contract code. The creator of the contract charges a 10% fee to subsequent investors and can withdraw the money via collectFees(). When the balance of the contract is sufficient, it will give the early investors a 1.5× return. When the user transfers the contract or calls the contract function, the initiator, receiver, transaction amount, transaction time, and other information will be stored in Ethereum as a transaction. As shown in the annotations of Fig. 2, similar information extraction on those transaction data can help us detect Ponzi contracts.

D. Taxonomy of Ponzi Schemes

According to the existing work [5], most Ponzi contracts can be roughly classified into four categories as follows.

1) *Tree-Shaped Schemes:* A tree structure is used to record the addresses of users. When a user joins the scheme, he needs to specify a user as the inviter who becomes his parent node. If no inviter is indicated, the parent will be the root node. The money of the new user is distributed among his ancestors, and the distribution logic is that the closer the ancestor, the larger his share. So if the node has more descendants, he will get more money.


①IfContract
②IfOwner
⑩FirstTime、⑪LastTime、⑫LifeTime
③ValueOut、④ValueIn、⑤IfProfit、
⑥AbusoluteBalance、⑦NumOut、⑧NumIn、
⑨IfRecieveMoreTimes

```
/**
 *Submitted for verification at Etherscan.io on 2016-08-25
*/

contract Tanya {

    struct Participant {
        address etherAddress;
        uint amount;
    }

    Participant[] public participants;

    uint public payoutIdx = 0;
    uint public collectedFees = 0;
    uint balance = 0;

  // only owner modifier
    address public owner;
    modifier onlyowner { if (msg.sender == owner) _ }

  // contract Constructor
    function Tanya() {
        owner = msg.sender;
    }

 // fallback function
    function(){
        enter();
    }

    function enter(){
      // collect fee
        uint fee = msg.value / 10;
        collectedFees += fee;

      // add a new participant
        uint idx = participants.length;
        participants.length++;
        participants[idx].etherAddress = msg.sender;
        participants[idx].amount = msg.value;

      // update available balance
        balance += msg.value - fee;

      // if there are enough ether on the balance we can pay out to an earlier participant
        uint txAmount = participants[payoutIdx].amount / 100 * 150;
        if(balance >= txAmount){
            if(!participants[payoutIdx].etherAddress.send(txAmount)) throw;

            balance -= txAmount;
            payoutIdx++;
        }
    }

    function collectFees() onlyowner {
        if(collectedFees == 0)return;

        if(!owner.send(collectedFees))throw;
        collectedFees = 0;
    }

    function setOwner(address _owner) onlyowner {
        owner = _owner;
    }
}
```

Fig. 2. Example of Ponzi contracts.

2) *Chain-Shaped Schemes:* They are a special case of tree-shaped schemes, where each node of the tree has exactly one child. These scams usually promise to redeem a certain multiple of funds when the user raises enough money from later stakeholders.
3) *Waterfall Schemes:* Similar to the chain scheme, but with a different logic for currency allocation. Each new investment is poured along the investor chain so that each investor can get his share.
4) *Handover Schemes:* Only the address of the last user is stored, and when a new investor joins the scheme the entry toll is increased. Only one investor will be receiving money at a time, and once he is paid, he hands over the privilege to the next user.

E. Graph Neural Networks

In recent years, GNNs have been extensively studied in various fields. The downstream tasks of graph neural network (GNN) are generally divided into node-level tasks and graph-level tasks. Node-level tasks include node classification and link prediction. Graph-level tasks include graph generation and graph classification.

The existing GNN methods are divided into two categories as follows: 1) Spectral-based methods, which use the Fourier transform and convolution theorem to define the graph convolution from the spectral domain, such as graph convolutional network (GCN) [23]. 2) Spatial-based methods, which implement the

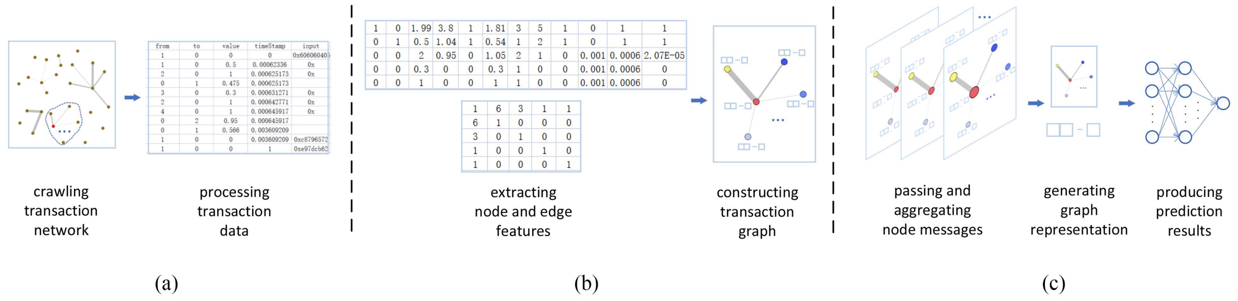

crawling transaction network
processing transaction data
extracting node and edge features
constructing transaction graph
passing and aggregating node messages
generating graph representation
producing prediction results
(a)
(b)
(c)

Fig. 3. Overview of PonziFinder. (a) Data preprocessing. (b) Feature extraction and transaction graph construction. (c) Ponzi contract detection.

graph convolution by defining the aggregation function in the node domain, such as graph attention network (GAT) [24]. With the application of GNN, researchers have proposed graph isomorphism network (GIN) [25] for graph isomorphic problems, self-attention graph pooling (SAGPool) [26] for optimization of graph pooling, and EGNN [27] networks for graphs with edge features.

For graphs with edge features, there has been a lot of research. EGNN [27] constructs formulations different from GCN and GAT, thus enabling them to exploit multidimensional edge features and achieve better performance in denoising. R-GCN [28] further considers the type and direction of edges based on GCN, and used different weight matrices for different types of edges. However, since there is only one type of transaction edge, R-GCN is not suitable for our scheme. Therefore, we choose EGNN to enhance the classification performance of Ponzi contracts using the number of transactions as the correlation between accounts.

III. Design of PonziFinder

We propose a novel approach, PonziFinder, for detecting Ponzi contracts by modeling it as a contract transaction graph classification task using EGNN(C) [27] and Set2Set [29]. Our method enables comprehensive learning of both local and global structure information within the contract graph. To capture the characteristics of Ponzi contracts' transaction flow, we conduct special preprocessing on the transaction data and extract useful node and edge features. Specifically, we treat trading edges as a reflection of the correlation degree between account nodes, rather than just a medium of information transmission. By utilizing EGNN(C) in this context, we demonstrate its potential for application in the field of Ponzi contract detection. Our approach represents an innovative and effective method for addressing this problem.

The overview of our proposed method is depicted in Fig. 3, which consists of three phases.

1) Data Preprocessing Phase: We crawl transaction data for each contract from the Ethereum platform and perform operations, such as filtering and sorting. We assume that each sample has sufficient transaction data.

2) Feature Extraction and Transaction Graph Construction Phase: We extract node features and edge features from transaction records, taking into account the characteristics of Ponzi contracts. We then construct an undirected graph for each contract, with contract accounts and external accounts as nodes and transactions between accounts as connecting edges.

3) Ponzi Contract Detection Phase: We use the EGNN(C) model to update the information of each account node by propagating it through transaction edges. For each EGNN(C) layer, we aggregate all node representations using Set2Set. Subsequently, we apply the ALGA to extract the final representation of the entire contract graph. By learning information about the flow of funds in accounts associated with contracts, PonziFinder can automatically detect Ponzi contracts.

We will now provide more details on each of these three phases.

A. Data Preprocessing

Algorithm 1 describes the process of preprocessing the smart contract transaction data. We crawl the transaction records of each contract in the dataset on the real Ethereum platform. The inputs to Algorithm 1 are a set of contract addresses and their first-order transaction data table. Valid transactions that are screened according to *is Error* value is not 1 (Line 2). To optimize the model's detection speed and reduce information redundancy, we conduct special sorting and numbering of transaction records and related accounts based on the essential characteristics of Ponzi schemes.

1) Data Normalization: To prevent data overflow caused by the large values in the "value" and "timestamp" fields of the transaction records, we perform data normalization on each contract's transactions. Specifically, we first calculate the maximum transaction amount (max) for each contract and then reduce all "value" fields in that contract's transaction records to 1/max of the original value (Line 3). Next, we normalize all the "timestamp" data in the transaction records of each contract by mapping the values between 0 and 1 (Line 4). This preprocessing step ensures that the data are in a suitable range for further analysis and prevents computational errors.

2) Transaction Sorting: To improve the efficiency of Ponzi contract detection, we prioritize transactions with fund flows and accounts with monetary interactions over nonmonetary transactions in the contract transaction history. Furthermore, we perform innovative analysis of the input value in the contract transaction, inspired by the transaction data of some Ponzi contracts. As shown in Fig. 4, the input data of a transaction of the Ponzi contract 0x4fb663c1616bfe80b5b6d5a214efa81d5a121801 is a string, where 0x5926651d represents the function selector,

Algorithm 1: Contract Transaction Data Preprocessing.

```
    Input: A set of contract addresses and their first-order
           transaction data table
           ${(contract_k, T_k[hash, from, to, contractAddress, value, timeStamp, isError, input])}_{k=1}^{n}$
    Output: Standardized transaction of contracts
           ${S_k[from, to, value, timeStamp]}_{k=1}^{n}$
1   for $k \leftarrow 1$ to $n$ do
2       $T_k \leftarrow T_k[isError! = 1]$;
        // Data normalization
3       $T_k[value] \leftarrow T_k[value]/max(T_k[value])$;
4       $T_k[timeStamp] \leftarrow min\_max\_normalize(T_k[timeStamp])$;
        // Rank the contract creation
           transaction first
5       $TransFirst_k \leftarrow T_k[T_k[contractAddress] == contract_k]$;
6       $Trans_k \leftarrow T_k[T_k[contractAddress]! = contract_k]$;
7       $TransValue_k \leftarrow Trans_k[Trans_k[value]! = 0]$;
        // Add monetary transactions
8       $TransSecond_k \leftarrow TransValue_k.sortby(timeStamp)$;
9       $TransFunc_k \leftarrow Trans_k[Trans_k[value] == 0]$;
10      $TransParaFunc_k \leftarrow TransFunc_k[len(TransFunc_k[value]) > 10]$;
        // Add Non-monetary transactions
           with function parameters
11      $TransThird_k \leftarrow TransParaFunc_k.sortby(TimeStamp)$;
        // Add other transactions
12      $TransOthers_k \leftarrow TransFunc_k[len(TransFunc_k[value]) <= 10]$;
13      $TransForth_k \leftarrow TransOthers_k.sortby(TimeStamp)$;
14      $TransSort_k \leftarrow concat(TransFirst_k, TransSecond_k, TransThird_k, TransForth_k)$;
        // Account numbering
15      $j \leftarrow 0$;
16      $nodeIndex[contract_k] \leftarrow j++$;
17      $paraAddr, paraTimeStamp \leftarrow extract(T_k[input, TimeStamp])$;
18      for $addr : TransSort_k[from, to]$ do
19          if $suit(paraTimeStamp, timeStamp)$ then
20              $nodeIndex[addr] \leftarrow j++$;
21          end
22          else if $addr : nodeIndex$ then
23              break;
24          end
25          else
26              $nodeIndex[addr] \leftarrow j++$;
27          end
28      end
29      $S_k = TransSort_k[from, to, value, timeStamp]$
30  end
31  return ${S_k[from, to, value, timeStamp]}_{k=1}^{n}$
```

```
0x5926651d0000000000000000000000003ecfceb33aa98d6f56a9bb3b3e5dbf5b83fca706
```

Fig. 4. Example of "input": A function call taking the account address as an argument.

followed by the parameters of the function. In this specific case, the address of 0x3ecfceb33aa98d6f56a9bb3b3e5dbf5b83fca706 account is added to the investors list of the contract in this transaction by function call, indicating that the account did not invest but received payment. We consider this type of input data to be of significant importance in detecting Ponzi contracts. Therefore, we further prioritize function call transactions with parameters among nonmonetary transactions. Specifically, we rank the contract creation transaction first in the transaction set of each contract (Line 5). Next, we filter out transactions with no amounts and sort the remaining ones by "timestamp" order (Lines 6–8). Nonmonetary transactions with function parameters are then sorted by "timestamp" and added after the monetary transaction records (Lines 9–11). Finally, all the remaining transactions are sorted last by "timestamp" (Lines 12–14).

This approach helps to reduce the amount of irrelevant data and ensure that the model focuses on the most relevant information for detecting Ponzi contracts.

3) Account Numbering: The account nodes are assigned a number based on their first participation time in the sorted transactions, with the earliest participant assigned the smallest number (Lines 23–28). In addition, for function call transactions where an account address appears as a parameter in the input data, we extract the address and add it to the transaction network as a node. This new node is then inserted into the execution position of the transaction (Lines 18–22), which prioritizes accounts that have a special relationship with the contract.

After preprocessing the transaction data of all contracts in the dataset, we proceed with feature extraction and transaction graph construction.

B. Feature Extraction and Transaction Graph Construction

There are two kinds of features to be extracted based on the characteristics of Ponzi contracts including node and edge features.

1) Node Feature Extraction: We initially extract the following 12 account node features around the essential characteristics of Ponzi contracts "using the funds of recent stakeholders to generate revenue for early stakeholders," which can be traced to the code logic shown in Fig. 2.

① *IfContract:* Whether the node is the target contract account (1: yes; 0: no). The trading behavior of the contract account provides useful information for Ponzi contract detection.

② *IfCreator:* Whether the node is the contract creator account (1: yes; 0: no). The creator of the contract is often the beneficiary of a Ponzi scheme, whose trading practices require special attention.

③ *ValueOut:* The total amount of value transferred out of this node. For the external account, this reflects the amount invested in the contract account. For the contract account, this represents the amount of money it sends out to other accounts.

④ *ValueIn:* Total revenue amount for this node. For the external account, this reflects the amount of collection received from the contract account. For the contract account, this represents the amount of investment received.

⑤ *IfProfit:* Whether the total revenue amount of this node is greater than the total transfer-out amount (1: yes; 0: no). In the transaction records of Ponzi contracts, contract creators and early investors often gain profits, while other investors often suffer heavy losses.

⑥ *AbusoluteBalance:* The difference between the revenue and expenditure of this node (|**ValueIn** - **ValueOut**|). The absolute value of the difference between income and expenditure is supplementary information on whether it is profitable or not.

⑦ *NumOut:* Number of outgoing transfers for this node. For external accounts, the number of outward transfers reflects the number of investments in contracts. For contract accounts, the number of outward transfers reflects the number of remittances to other accounts.

⑧ *NumIn:* Number of incoming transfers for this node. For external accounts, it represents the number of times revenue is received from the contract account. For contract accounts, it represents the number of times the contract is invested.

⑨ *IfReceiveMoreTimes:* Whether the number of incoming transfers is greater than outgoing transfers for this node (1: yes; 0: no). In Ponzi contracts, later investors may not receive a transfer from the contract or receive returns less than they invest, which can help identify a Ponzi contract.

⑩ *FirstTime:* The first transaction time of this node. In Ponzi contracts, accounts are more likely to make a profit if they enter contracts early, especially the creators and partners of the contract.

⑪ *LastTime:* The last transaction time of this node. Trading time plays an important role in Ponzi contracts because the investment time can have a big impact on the return.

⑫ *LifeTime:* Transaction time difference for this node (**LastTime** - **FirstTime**). Contract creators and early investors tend to have a long lifetime in contract transactions because they always receive contract payouts. On the contrary, the late investment may receive little or no transfer from the contract, the lifetime is relatively short.

In the transaction graph of a Ponzi contract, the contract account often appears to display frequent outward transfer and has small income and expenditure differences because it requires timely payment to the contract creator and early investors. Contract creator and early investor accounts tend to have a small number of investments, but frequently receive transfers, and have a long life cycle. Late-stage investor accounts tend to show little or even no return after investment and short life cycles. In contrast, relevant accounts tend not to have such differences in behavior based on transaction time in the transaction graph of non-Ponzi contracts.

In addition, an optimal node feature set {**IfContract**, **ValueIn**, **IfProfit**, **NumOut**, **IfReceiveMoreTimes**, **FirstTime**} is found for our scheme through experiments to improve efficiency, as discussed in Section IV-E.

2) Edge Feature Extraction: We extract the edge feature as the total number of transactions between two account nodes (**TotalTimes**). Different from **NumOut** and **NumIn** in the node feature, **TotalTimes** includes transactions without transaction amounts that are only used to call the function, which reflects the degree of association between accounts.

3) Transaction Graph Construction: We construct the undirected graph structure based on contract history transactions, taking contract accounts and external accounts as nodes and transactions between accounts as connecting edges.

C. *Ponzi Contract Detection*

In this subsection, we present our PonziFinder in four steps. The process is illustrated in Fig. 5, where the red nodes represent the target contracts, the yellow nodes represent the contract creators, the blue nodes represent other accounts that have transactions with the target contracts, and the shade of blue indicates the transaction time.

Next, we will describe some important steps of the detection phase.

1) Input: Suppose the transaction graphs of K contracts in the dataset are {Graph_1, Graph_2,..., Graph_K}. Given the transaction graph Graph_k of a contract *k*, the inputs to the model are the node feature matrix $\mathbf{X}_k \in \mathbf{R}^{N\times F}$ and the edge feature matrix $\mathbf{E}_k \in \mathbf{R}^{N\times N}$. N represents the number of nodes in the graph (in our experiments, we specify the same N for all contract graphs) and F represents the number of node features. In addition, the node and edge features are obtained from the first two phases of PonziFinder discussed in Section III-B and III-C.

2) Passing and Aggregating Node Messages: EGNN(C) [27] is a convolution-based edge-enhance graph neural network that can utilize edge features, including those of undirected, directed, or multidimensional edges.

In our scheme, the edge represents the total number of transactions between accounts, reflecting the degree of correlation between different account nodes. The EGNN(C) layer enhances the node features with the edge features, resulting in a more effective Ponzi contract detection. The input to the network is denoted by $\mathbf{X}_k \in \mathbf{R}^{N\times F}$ and $\mathbf{E}_k \in \mathbf{R}^{N\times N}$. After passing through the first EGNN(C) layer, X_k is filtered to produce an $N \times F$ new node feature matrix X_k^1. This procedure is repeated for every subsequent layer. Finally, we get the final representation of the node X_k^L in the L layer.

3) Generating Graph Representation: In the Ponzi contract transaction graph, the importance of information in different account nodes varies greatly. For example, the Ponzi contract account frequently accepts transfers, and the Ponzi contract creator account calls the contract function many times. Simply accumulating or averaging the presentation of account nodes would weaken the expressiveness and lose most information. To solve this problem, we use Set2Set [29], a graph readout method based on the attention mechanism, to pool the representation

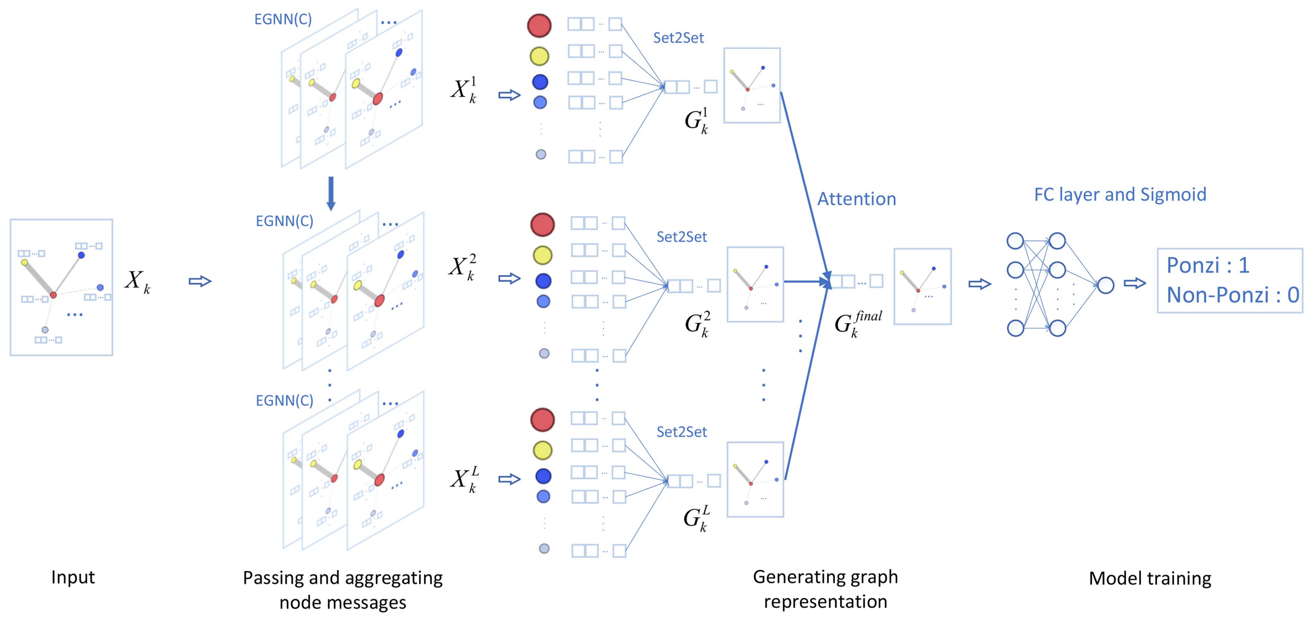


Fig. 5. Process of Ponzi contract detection.

of the entire graph. Through Set2Set, we select the information of account nodes that are more important for detecting Ponzi contracts, ensuring that the most critical information is captured.

Traditionally, graph-level representations are produced by feeding node embeddings learned by multiple GNN layers to the global readout. However, as the depth of the network increases, the node embedding may become oversmoothed, leading to poor generalization performance of the graph-level output. This problem is particularly evident in the Ponzi contract graph detection problem because we focus on the first-order trading graphs of contracts where the graph's depth is shallow. Inspired by [30], we introduce an AGLA that employs a self-attention mechanism to aggregate the layerwise graph-level features. This approach effectively extracts and utilizes all depths of global information.

We apply Set2Set to the output of each EGNN(C) layer. Typically, we use the node representations X_k^i (where $(i = 1, 2, \ldots L)$) obtained from the previous step as input to Set2Set. The result is a graph-level representation vector G_k^i for each layer i. To obtain the final graph-level output, we use a layer-focused attention mechanism. This mechanism computes an attention score α_k^i for each layer i, which reflects the importance of global features learned at that layer. We then compute the final aggregated representation G_k of the contract graph k by taking a weighted average of the layer features.

4) Model Training: The prediction $\hat{y}_k$ of the smart contract k is generated by inputting G_k to the fully connected (FC) layer and the activation function Sigmoid, which is used to determine whether the smart contract is a Ponzi contract or not.

The output $\hat{y}_k$ is then compared with the true label y_k of the smart contract k, and the performance of the model is measured by calculating the cross-entropy loss.

However, the number of non-Ponzi contracts far exceeds that of Ponzi contracts in the blockchain, resulting in an imbalanced data classification problem. Simply classifying all samples as non-Ponzi schemes will still result in high classification accuracy. Therefore, Ponzi contract detection is essentially an imbalanced data classification problem.

To address this issue, data resampling techniques, such as undersampling and oversampling have been proposed [31]. Oversampling methods increase the number of minority class samples, but random oversampling can lead to overfitting. Synthesizing minority class data [32] is challenging as it is difficult to ensure that synthetic data are as valid as real data. This is particularly difficult for the Ponzi contract graph detection problem, since we need to generate not only node information but also edge information to form the graph structure.

Simple random undersampling randomly discards majority class samples, leading to significant loss of information. However, in Ethereum smart contracts, a large number of duplicate transaction streams exist. Thus, we manually audit the transaction data in the dataset and design Algorithm 2 to automatically perform stratified undersampling of non-Ponzi contracts based on transaction data. After obtaining the valid contract transaction (Line 3), we calculate the number of accounts interacting with the contract (Line 4), the total transaction amount of the contract (Line 5), the sum of the contract income amount (Lines 6–7), the sum of contract outcome amount (Lines 8–9), and classify non-Ponzi contracts into following four categories (Lines 10–25).

1) Only have transactions with the creator.
2) All transactions have no monetary flow.
3) Only receive money and do not transfer outward.
4) Have normal transactions.

Finally, we conduct stratified sampling according to the undersampling coefficient to obtain the undersampling result (Line 26), which reduces the loss of data information to some extent.

Algorithm 2: Stratified Undersampling.

```
Input: undersampling rate $r$, a set of Non-Ponzi
       contract addresses and their first-order
       transaction data table
       $\{(contract_m, T_m[hash, from, to, value, timeStamp, isError])\}_{m=1}^{M}$
Output: List of Non-Ponzi contracts after stratified
        undersampling $\{contract_a\}_{a=1}^{A}$
1  Var $CreatorOnly \leftarrow \{\}$, $NoMonetary \leftarrow \{\}$,
     $OnlyReceive \leftarrow \{\}$, $NormalTrans \leftarrow \{\}$;
2  for $m \leftarrow 1$ to $M$ do
3      $T_m \leftarrow T_m[isError! = 1]$;
4      $nodeNum \leftarrow Count(T_m[from, to])$;
5      $money \leftarrow Sum(T_m[value]$;
6      $contractIn \leftarrow T_m[T_m[to] == contract_m]$;
7      $contractInValue \leftarrow sum(contractIn[value])$;
8      $contractOut \leftarrow T_m[T_m[from] == contract_m]$;
9      $contractOutValue \leftarrow sum(contractOut[value])$;
       // Only have transactions with
          creator
10     if $nodeNum <= 2$ then
11         $CreatorOnly.add(contract_m)$;
12         break;
13     end
       // All transactions have no money
14     else if $money == 0$ then
15         $NoMonetary.add(contract_m)$;
16         break;
17     end
       // Only receive money
18     else if $contractInValue! = 0$ &&
         $contractOutValue == 0$ then
19         $OnlyReceive.add(contract_m)$;
20         break;
21     end
       // Other contracts with normal
          transaction
22     else
23         $NormalTrans.add(contract_m)$;
24         break;
25     end
       // Stratified undersampling
26     Var $UnderSampling \leftarrow$
         $randomselect(CreatorOnly, NoMonetary,$
         $OnlyReceive, NormalTrans, r)$;
27     return $UnderSampling$ $\{contract_a\}_{a=1}^{A}$;
28 end
```

After the undersampling is used to balance the sample categories, the model can be input for learning. As shown in Fig. 5, after learning all the data in the training set, the loss values are fed back to the Adam optimizer to update and optimize the parameters in the model and perform a new round of training.

IV. Evaluation

We have developed a prototype tool called PonziFinder, which automates the detection of Ponzi contracts. Our research seeks to answer the following six questions.

- *RQ1:* How effective is PonziFinder in comparison to other representative Ponzi scheme detection methods and GNN methods? Can the addition of edge features to the transaction graph lead to a significant improvement in detection accuracy?
- *RQ2:* Is PonziFinder capable of detecting various types of Ponzi contracts compared to other methods?
- *RQ3:* Can preprocessing steps for analyzing trading input values improve the effectiveness of Ponzi contract detection? And can the addition of ALGA mechanism at the graph readout stage enhance the detection performance?
- *RQ4:* What is the optimal node feature set for our scheme?
- *RQ5:* How can PonziFinder avoid randomness and mitigate information loss with the undersampling method we used?
- *RQ6:* How do we select optimal hyperparameters?

Next, we first present the experimental settings, followed by answering the above research questions one by one.

A. Experimental Settings

1) Datasets: To ensure the reliability of the dataset labels, we combine the labeled dataset provided in literature [7] with the publicly available Ponzi contracts dataset in literature [5], and filter some data according to the requirements of our method.

The following three types of contracts are removed from our dataset.

① *Contracts that have no transaction records since created:* We consider contracts that have been created for a long time without any other transaction information to be almost abandoned, so there is no practical point in detecting them.

② *Non-Ponzi contracts that are highly similar to Ponzi trading streams:* They are usually crowdfunding contracts, which also generate "money concentrated in the creator's account or in a few accounts." To prevent interference with the training process, this type is removed from the dataset. In the future, if such a contract is encountered and predicted to be a Ponzi contract, PonziFinder will give an alert based on the contract type "the contract transaction flow is highly similar to a Ponzi scheme, so if the contract is an investment contract, you need to be vigilant."

③ *Ponzi contracts where no account interacts with them financially besides the creator:* Contracts of this type indicate that no other investor has invested in the contract for a long time. They are almost abandoned and can hardly cause real harm.

After deleting these three categories of contracts, we conduct experiments on 2420 non-Ponzi contracts and 113 Ponzi contracts.

2) Implementation Details: All the experiments are conducted on a computer equipped with an Intel Core i7 CPU at 2.9 GHz, and 32 GB of Memory. We write about 1800 lines of code to accomplish the main functions of PonziFinder. The ratio of train set, validation set, and test set is 6:2:2.

The number of trading accounts associated with each contract is inconsistent, but we want to build a contract trading graph classification task, so the number of input nodes should be fixed. We set the number of input nodes to be *maxnode*.

Contract transaction records with nodes greater than *maxnode* will be truncated. Because the accounts have been specially sorted during preprocessing, most truncated accounts have no financial interaction with the contract, which are not very useful for detecting Ponzi contracts.

Contract transaction records with nodes less than *maxnode* will be expanded by 0. This is equivalent to adding virtual nodes without any information in the graph, so there is no impact on the transfer and aggregation of node information.

Since the nodes are specially numbered during preprocessing, *maxnode* changes have little effect on the experimental results. We set *maxnode* to 212, which is the maximum number of accounts in the dataset that have monetary transactions with Ponzi contract accounts. The N in the node feature matrix $\mathbf{X}_k \in \mathbf{R}^{N\times F}$ we mentioned in Section III-D will be unified as the *maxnode* here.

Other parameter settings are shown as follows. The number of EGNN(C) layers is 2, the learning rate is initialized to 0.0005, the number of hidden units is 64, the dropout rate is set to 0.2, the weight decay coefficient is 5e-4, and the undersampling rate is 5%.

3) Evaluation Metrics: In order to accurately evaluate the performance of our proposed method, we use six metrics: 1) Precision, 2) Recall, 3) Accuracy, 4) F1-score, 5) TimeCost, and 8) Stdev. The formulas for the first four indicators are shown as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} \tag{1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} \tag{2}$$

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{TP + TN + FP + FN}} \tag{3}$$

$$F1 - \text{score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision + Recall}} \tag{4}$$

where TP represents the number of Ponzi contracts that are correctly identified, TN represents the number of non-Ponzi contracts that are correctly identified, FP represents the number of non-Ponzi contracts that are misjudged as Ponzi contracts, and FN represents the number of Ponzi contracts that are detected as non-Ponzi contracts.

In addition, TimeCost refers to the average time to detect whether a contract is a Ponzi contract and includes the entire process from data preprocessing to model detection. Stdev stands for standard deviation and reflects the distribution of the difference between the predicted value of the model and the actual label value. A smaller standard deviation may indicate that the model fits the data more consistently.

TABLE I
PERFORMANCE COMPARISON WITH EXISTING METHODS AND OTHER GNN ALGORITHMS

Methods	Precision	Recall	Accuracy	F1-score	TimeCost(s)	Stdev
PCD-ICNN [9]	0.9524	0.8696	0.9149	0.9091	0.1056	0.3341
RFPonzi [7]	0.9524	0.8696	0.9149	0.9091	0.0844	0.3637
DNNPonzi [34]	**1.0000**	0.8696	0.9362	0.9302	**0.0425**	0.3135
PSD-OL [18]	**1.0000**	0.9130	0.9574	0.9545	0.1088	0.2934
PSD-TN [16]	0.8462	0.9565	0.8936	0.8980	0.1274	0.3520
GAT	0.9167	0.9565	0.9362	0.9362	0.1323	0.3845
GCN	0.9545	0.9130	0.9362	0.9333	0.1111	**0.2703**
EGNN(A)	0.9200	**1.0000**	0.9574	0.9583	0.1389	0.2750
GCN-SAGPool	0.9565	0.9565	0.9574	0.9565	0.1067	0.3042
GIN	0.9583	**1.0000**	**0.9787**	**0.9787**	0.0933	0.2894
PonziFinder	0.9583	**1.0000**	**0.9787**	**0.9787**	0.0779	0.2721

The bold entities serve to highlight the optimal values within each respective column.

B. Comparison With Existing Methods and Other GNN Algorithms (RQ1)

On the test set, we compare PonziFinder with advanced methods in existing studies, and other GNNs, to demonstrate whether our method has an advantage in the Ponzi contracts detection task. The compared methods are summarized as follows.

1) *PCD-ICNN [9]:* A Ponzi contract detection method based on code features via an improved convolutional neural network.
2) *RFPonzi [7]:* A Ponzi contract detection method based on combined features using random forest [33].
3) *DNNPonzi [34]:* A Ponzi contract detection method based on combined features using deep neural network.
4) *PSD-OL [18]:* A Ponzi scheme detection method based on combined features via oversampling-based long short-term memory (LSTM) [35] for smart contracts.
5) *PSD-TN [16]:* A Ponzi scheme detection method based on transaction features in the Ethereum transaction network.
6) *GAT:* Graph attention network [24], introducing the attention mechanism to GNNs based on the spatial domain.
7) *GCN:* Graph convolutional network [23], implementing layer convolution based on graph Laplacian.
8) *EGNN(A):* Attention-based edge-enhanced GNN [27], which can more fully exploit the undirected, directed, and multidimensional edge features.
9) *GCN-SAGPool:* Self-attention graph pooling [26] is a graph pooling method that uses a self-attention mechanism to selectively retain important nodes in a graph, typically based on GCN.
10) *GIN:* Graph isomorphism network [25], having a strong representation learning ability and the ability to deal with graph isomorphism.

1) Results and Analyses: From the experimental results shown in the Table I, it can be seen that although PCD-ICNN [9] has high Precision, there are relatively more missing reports. RFPonzi [7] and PSD-OL [18] methods take longer to detect Ponzi contracts because they need to combine code information with transaction information . The experimental results in [7] showed that the addition of transaction features does not improve the effect well, so it is speculated that the reason for their

TABLE II
PONZI TYPES

Type	Number
Tree-shaped schemes	3
Chain-shaped schemes	86
Waterfall schemes	3
Handover schemes	1
Others	20

higher underreporting may be that there are still problems with extracted transaction features. Although DNNPonzi [34] is fast, it does not have good detection ability because of poor model fitting. The detection ability of PSD-TN [16] method is weak because of the problematic feature extraction and the neglect of transaction edge importance.

We perform graph-level classification based on GAT, GCN, EGNN(A), GCN-SAGPool, and GIN using the 12 node features and one edge feature extracted in our scheme. It can be seen that PonziFinder is most suitable for Ponzi contract detection problems compared to other models. It is worth mentioning that by comparing GCN and PonziFinder as well as GAT and EGNN(A), we can conclude that adding edge features help detect Ponzi contracts.

***Answer to RQ1:** Compared with existing methods and other graph neural network methods, the accuracy of PonziFinder is relatively higher and takes less time. The experimental results reveal that adding edge features help detect Ponzi contracts.*

C. Comparison of the Ability to Detect Different Types of Ponzi Contracts (RQ2)

The purpose of this experiment is to compare the effectiveness of the models in detecting different types of Ponzi contracts. However, the test set is relatively small because there are few real Ponzi contracts on Ethereum. Moreover, when we count the number of Ponzi contracts in each category in the test set, we find that many categories are absent. In addition, a smaller test set weakens the difference in the fitting effect of different models on the train set and validation set. Therefore, since the previous experiment has already validated the model's effectiveness in detecting new data, we utilize all Ponzi contracts in the dataset to further test the ability of each model to detect various types of Ponzi contracts.

Based on the existing work [5], we can classify existing Ponzi contracts into four categories: Tree-shaped schemes, Chain-shaped schemes, Waterfall schemes, and Handover schemes. There are also other Ponzi schemes with variable forms of realization that cannot be categorized.

We refer to publicly available information from paper [5] to count the number of different Ponzi contract types in our dataset. The results are shown in Table II. We input these data into each model separately to test the ability of detection for different classes of Ponzi contracts.

1) Results and analyses: The experimental results in Table III show that PonziFinder can not only detect all four types of Ponzi contracts, but also has good detection ability for other types

TABLE III
PONZI TYPE COMPARISION

Methods	Tree_shaped	Chain_shape	Waterfall	Handover	Others
PCD-ICNN [9]	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8500
RFPonzi [7]	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8000
DNNPonzi [34]	**1.0000**	0.9767	**1.0000**	**1.0000**	0.7000
PSD-OL [18]	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8000
PSD-TN [16]	**1.0000**	0.9535	**1.0000**	**1.0000**	0.8500
GAT	**1.0000**	0.9535	**1.0000**	**1.0000**	0.9000
GCN	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8500
EGNN(A)	**1.0000**	0.9767	**1.0000**	**1.0000**	0.9500
GCN-SAGPool	**1.0000**	0.9535	**1.0000**	**1.0000**	0.9500
GIN	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9500
PonziFinder	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**

The bold entities serve to highlight the optimal values within each respective column.

TABLE IV
ABLATION EXPERIMENTS FOR INPUT VALUE ANALYSIS

Cases	Precision	Recall	Accuracy	F1-score	TimeCost(s)
NonInput	0.9200	**1.0000**	0.9362	0.9583	**0.0735**
PonziFinder	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.0779

The bold entities serve to highlight the optimal values within each respective column.

TABLE V
ABLATION EXPERIMENTS FOR AGLA

Cases	Precision	Recall	Accuracy	F1-score	TimeCost(s)
NonAGLA	0.9565	0.9565	0.9574	0.9565	**0.0764**
PonziFinder	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.0779

The bold entities serve to highlight the optimal values within each respective column.

with variable implementation forms. In the previous experiment, EGNN(A) and GIN, which have the same Recall value of 100%, can only detect 95% of the Ponzi contracts in the unclassified "Others" when inputting all Ponzi contracts in the dataset. Therefore, PonziFinder has stronger fitting ability compared to other models and is truly able to detect all classes of Ponzi contracts.

***Answer to RQ2:** When the transaction records are sufficient, PonziFinder can detect all types of Ponzi contracts.*

D. Ablation Experiment (RQ3)

1) Input Value Analysis: In Section III-B, we analyze the transaction input values and extract useful parameters of the function call transaction. To evaluate the effectiveness of this approach in detecting Ponzi contracts, we conducted ablation experiments where we isolated the input value analysis method from other treatments. The goal was to determine whether the input value analysis method can improve data quality and enhance Ponzi contract detection.

2) Attention-Based Global Layerwise Aggregation Mechanism: In Section III-D, we introduced the ALGA mechanism to generate the final graph-level representation. To evaluate the effectiveness of this mechanism, we conducted ablation experiments while controlling other treatments in the experiment, solely focusing on AGLA.

3) Results and Analyses: Experimental results presented in Table IV demonstrate that incorporating input value analysis enhances the data quality, leading to a significant improvement in detection precision. On the other hand, as shown in Table V, the AGLA mechanism effectively mitigates the problem of oversmoothing of node embeddings in shallow contract graphs,

TABLE VI
IMPORTANCE RANK OF NODE FEATURES

Feature	Initial Importance	Final Importance
IfContract	**6**	**3**
IfCreator	2	7
ValueOut	11	11
ValueIn	**10**	**6**
IfProfit	**1**	**1**
AbusoluteBalance	5	9
NumOut	**9**	**5**
NumIn	8	10
IfReceiveMoreTimes	**3**	**2**
FirstTime	**7**	**4**
LastTime	12	12
Lifetime	4	8

The bold entities serve to highlight the optimal values within each respective column.

TABLE VII
PERFORMANCE OF DIFFERENT NODE FEATURE SET

NumFeatures	Precision	Recall	Accuracy	F1-score	TimeCost(s)
12	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.1029
11	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.1003
10	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.0953
9	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.0889
8	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.0878
7	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.0816
6	**0.9583**	**1.0000**	**0.9787**	**0.9787**	0.0779
5	0.9565	0.9565	0.9574	0.9565	**0.0754**

The bold entities serve to highlight the optimal values within each respective column.

resulting in increased detection of Ponzi contracts and a considerable improvement in recall rate.

Answer to RQ3: *Incorporating input value analysis in data preprocessing improves Precision by 3.8%. And including Attention-Based Global Layerwise Aggregation mechanism improves Recall by 4.3%.*

E. Optimal Node Feature Set (RQ4)

We initially extracted 12 node features, but we noticed that some information they expressed was duplicated. In order to further improve operation speed, we simplify the model by reducing the number of node features to carry out experiments. A preliminary importance ranking of 12 node features obtained by observing the trading characteristics of real Ponzi contracts is performed in Table VI. And we try to remove the feature with the lowest importance and perform the experiment each time.

If the F1-score of the experimental results does not decrease, it indicates that the feature is redundant. However, if the F1-score decreases, we attempt to remove the higher ranked feature, conduct further experiments, and update the importance ranking table. If the removal of all features in a top-down traversal does not identify a feature to simplify, the current feature set is considered the optimal feature set obtained by this deletion order.

1) Results and Analyses: Experimental results in Table VII show that removing the features rank in the bottom 6 of the final importance ranking will not affect the experimental results. When the number of node features is reduced to 6, the F1-score value of the model decreases regardless of which feature is deleted. We also note that when the number of features is

TABLE VIII
PERFORMANCE OF MULTIMODELS

NumModels	Precision	Recall	Accuracy	F1-score	TimeCost(s)
1	0.9583	**1.0000**	0.9787	0.9787	**0.0779**
2	0.9200	**1.0000**	0.9574	0.9583	0.0897
3	0.9200	**1.0000**	0.9574	0.9583	0.0922
4	0.9200	**1.0000**	0.9574	0.9583	0.0965
5	0.9583	**1.0000**	0.9787	0.9787	0.1025
6	0.9200	**1.0000**	0.9574	0.9583	0.1054
7	0.9583	**1.0000**	0.9787	0.9787	0.1139
8	0.9583	**1.0000**	0.9787	0.9787	0.1204
9	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.1277

The bold entities serve to highlight the optimal values within each respective column.

reduced, the time taken by the model to detect Ponzi contracts is further reduced, which means that feature reduction can greatly improve model efficiency. The optimal node feature set obtained by initial importance deletion order is {**IfContract**, **ValueIn**, **IfProfit**, **NumOut**, **IfReceiveMoreTimes**, **FirstTime**}.

Answer to RQ4: *The optimal node feature set we found for our scheme to improve the detection efficiency is {* **IfContract** *,* **ValueIn** *,* **IfProfit** *,* **NumOut** *,* **IfReceiveMoreTimes** *,* **FirstTime** *}.*

F. Undersampling (RQ5)

This experiment is conducted on the test set. In previous experiments, the undersampling is performed only once and most non-Ponzi contracts are discarded. To reduce the randomness and make full use of the information in the dataset, we optimize our scheme based on the idea of ensemble learning. Specifically, after setting aside the same test set as the previous experiments, we repeatedly stratify the data extraction for non-Ponzi contracts to obtain samples with a similar number of Ponzi contracts, and those are then combined and trained. After training M models, we use maximum voting, an integration method, to combine the model results. We consider a contract to be a Ponzi scheme when half or more results of models represent Ponzi contracts. The experimental results are shown in Table VIII.

1) Results and Analyses: The experimental results demonstrate that combining two models results in a slight decrease in accuracy due to the misreporting of different contracts by the two models. The accuracy of the combined results fluctuates until nine models are used, after which the accuracy reaches 100%. However, using all nine models may be time-consuming. Thus, we propose a method to meet different demands for accuracy and speed. For instance, if users want a quick detection to avoid potential Ponzi schemes, a lightweight single-model approach can be used. On the other hand, for more accurate detection regardless of time, a multimodel approach can be employed to reduce both false positives and false negatives.

Answer to RQ5: *By combining the idea of stratified undersampling and ensemble learning, PonziFinder reduces the information loss and improves the effectiveness of the Ponzi contract detection.*

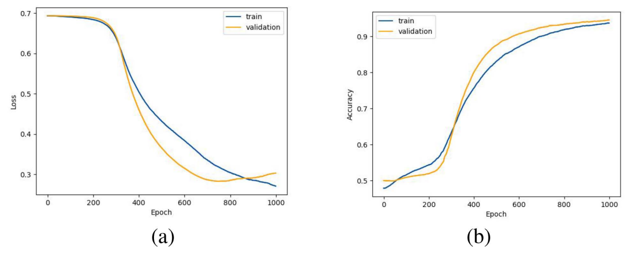


Fig. 6. Training and validation loss and accuracy affected by epoches. (a) Loss affected by epoches. (b) Accuracy affected by epoches.

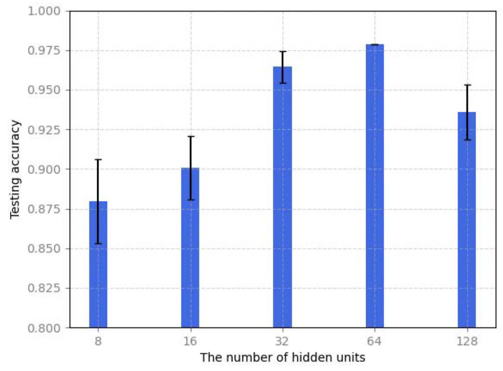


Fig. 7. Testing acuuracy affected by number of hidden units.

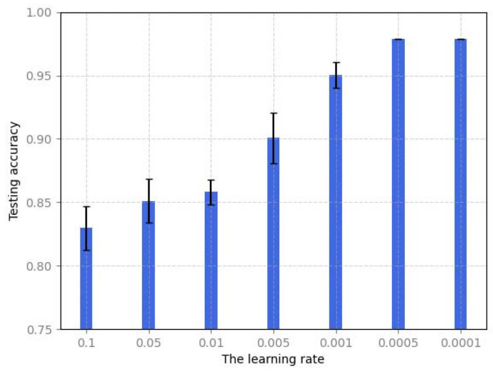


Fig. 8. Testing acuuracy affected by learning rate.

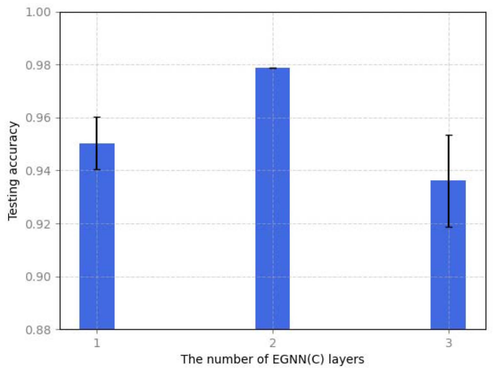


Fig. 9. Testing accuracy affected by the number of EGNN(C) layers.

G. Hyperparameter Selection (RQ6)

In order to optimize the model performance as much as possible, control-variable method is used to select the relatively optimal combination of hyperparameters.

1) Impact of Epochs: In this experiment, we set the initial epoch value to 1000. Other parameters are described in Section IV-A. The training loss and validation loss as the epoch changes are shown in Fig. 6(a), and the training accuracy and validation accuracy as the epoch changes are shown in Fig. 6(b).

Through many experiments, the effect of PonziFinder is found to stabilize before 800 epoch and the validation loss will be minimized. As the epoch continues to increase, although the training loss decreases, the validation loss gradually increases, indicating that the model is overfitting. So the epoch is finally set to 800.

2) Impact of the Number of Hidden Units: If there are too few hidden units in the network, the model will be underfitted. Conversely, too many hidden units can lead to overfitting. we keep other parameters unchanged, and change the number of hidden units within $\{8, 16, 32, 64, 128\}$. Fig. 7 shows the testing accuracy under different number of hidden units.

When the number of hidden units is relatively small, the testing accuracy increases with the number of hidden units. When the number of hidden units reaches 128, the testing accuracy is found to greatly decrease caused by overfitting after manual analysis, so 64 is finally selected as the number of hidden units.

3) Impact of Learning Rate: A high learning rate may cause the model effect to fluctuate during training. Low learning rate takes longer to train. In order to find the most suitable learning rate, we set the value as the following $\{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$ and conduct a comparative experiment. Referring to the experiment part in [27], we set the minimum learning rate to 0.0001. Other parameters remain unchanged.

As can be seen from Fig. 8, when the learning rate is small, the effect of PonziFinder on the testing set is unstable due to large fluctuations. When the learning rate reaches 0.0005, the model with better effect can be trained stably. The reason why we prefer 0.0005 to 0.0001 is that when the learning rate is 0.0001, even if the epoch is set to 1000, the model's loss curve is still unstable. But with a learning rate of 0.0005, the model stabilizes within 800 epoch.

4) Impact of the Number of EGNN(C) Layers: If the number of EGNN(C) layers is too small, the model can only extract simple information in transaction graph, resulting in poor detection ability. But when the EGNN(C) layer deepens, oversmoothing occurs and the representation of different nodes will be weakened. We, respectively, adopt the EGNN(C) layer number as $\{1, 2, 3\}$ and conduct experimental comparison. Other parameters in the experiment remain unchanged. As shown in Fig. 9, when the number of EGNN(C) layers is set to 2, the effect of PonziFinder will be optimal.

5) Results and Analyses: As can be seen from the experimental results in Figs. 6–9, the hyperparameters we selected can make PonziFinder reach the optimal level.

TABLE IX
BASIC INFORMATION OF 167 NEWLY COLLECTED PONZI CONTRACTS

	NumAccounts	TransValue	Number of Ponzi contracts
	1	0	29
	1	>0	5
	>1	0	3
	>1	>0	130
Sum			167

Answer to RQ6: *In order to select the optimal combination of hyperparameters for PonziFinder as much as possible, we use the control-variable method to carry out numerous experiments on important hyperparameters.*

V. CASE ANALYSIS

Considering that our dataset used in Section V only includes Ponzi contracts before 2017, in order to demonstrate the power of PonziFinder in detecting Ponzi contracts and understand the development of Ponzi contracts in recent years, we refer to a recent research [36] and collect a new dataset containing 318 Ponzi contracts from *Xblock*.[1] After comparing the dataset used in section V, we remove the duplicate Ponzi contracts. A total of 167 Ponzi contracts deployed in 2018 and 2019 remain in the newly collected dataset, and their basic information are shown in Table IX.

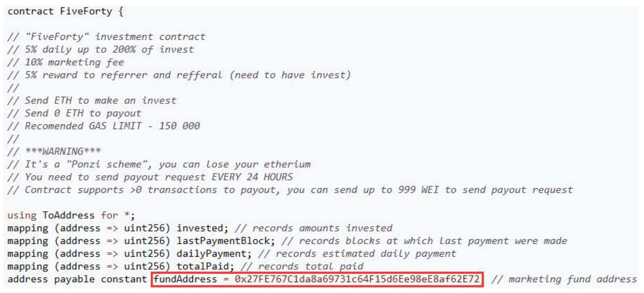
The first two lines in Table IX indicate that these contracts only have transactions with the creator after they are created, and the third line represents that although there are other accounts interacting with the contract, there is no transaction amount. PonziFinder does not consider these Ponzi contracts, because they do not produce actual fraud facts.

The remaining 130 Ponzi contracts are tested using PonziFinder, and the results show that PonziFinder is able to successfully label 128 Ponzi contracts with an accuracy of 98.5%. The two undetected Ponzi contracts are 0xDdDC5B65208287168BB640E8B303eaAaFE2Ea95F and 0xaa6eaE4Fb91DF553291160f41888CB0cf1f20948.

A manual review of the transaction data and contract codes for the two contracts finds that their codes are nearly identical, with the same contract name "FiveForty" and a similar warning "This is a Ponzi scheme" in the comments of codes, suggesting that both are deployed by researchers for testing. A snippet of the smart contract "FiveForty" is shown in Fig. 10.

After analysis, we believe that PonziFinder fails to detect them for two reasons. *First, the marketing fund address set in code cannot be analyzed from the transaction, causing interference with the results.* Both contracts in the code specifies the address 0x27FE767C1da8a69731c64F15d6Ee98eE8af62E72 as marketing fund address, this causes some disruption to PonziFinder because there is an account that has not invested but receives contract transfers in the transaction data, which could not be analyzed from the function

[1]Xblock:http://xblock.pro/#/dataset/25



```
contract FiveForty {

// "FiveForty" investment contract
// 5% daily up to 200% of invest
// 10% marketing fee
// 5% reward to referrer and refferal (need to have invest)
//
// Send ETH to make an invest
// Send 0 ETH to payout
// Recomended GAS LIMIT - 150 000
//
// ***WARNING***
// It's a "Ponzi scheme", you can lose your etherium
// You need to send payout request EVERY 24 HOURS
// Contract supports >0 transactions to payout, you can send up to 999 WEI to send payout request

using ToAddress for *;
mapping (address => uint256) invested; // records amounts invested
mapping (address => uint256) lastPaymentBlock; // records blocks at which last payment were made
mapping (address => uint256) dailyPayment; // records estimated daily payment
mapping (address => uint256) totalPaid; // records total paid
address payable constant fundAddress = 0x27FE767C1da8a69731c64F15d6Ee98eE8af62E72 // marketing fund address
```

Fig. 10. Snippet of the smart contract "FiveForty."

parameters in transactions. *Second, no user actually loses money or users have returned more than 86% of the invested funds, which interferes with the result.* The 0xDdDC5B65208287168BB640E8B303eaAaFE2Ea95F contract has no investors except the contract. The sole investor in the contract 0xaa6eaE4Fb91DF553291160f41888CB0cf1f20948 gets 86% of his money back. Therefore, the two underreported Ponzi contracts do not actually reflect the characteristics of fraud in the transaction.

The 98.5% accuracy on the new dataset proves that even as Ponzi contracts continue to evolve, PonziFinder remains powerful.

VI. RELATED WORK

Many kinds of research have been conducted to extract features for the detection of Ponzi contracts from the following perspectives: Code-based features, transaction-based features, and combined features.

1) Code-Based Features: The current research based on code features for Ponzi contracts detection is mainly divided into source-code-based and bytecode-based.

The source code of smart contracts has well-defined structural and semantic information, which can capture the characteristic information of Ponzi contracts more accurately than handcrafted features. Chen et al. [8] used a multichannel TextCNN and transformer to detect Ponzi contracts. But this approach relies too much on the source code, some of which is not publicly available in the blockchain.

More approaches used the bytecode feature of smart contracts since the bytecode is compiled from the source code and can also contain all the information. Lou et al. [9] converted the contract bytecodes to the corresponding image in the form of a single channel, and finally completed the classification using an improved convolutional neural network. Shen et al. [13] extracted features from the bytecode, then transformed the Ponzi contract detection into an anomaly detection problem. Chen et al. [12] proposed a new semantic-aware method based on symbolic execution by identifying investor-related transfer behaviors and the adopted distribution strategies. Peng et al. [14] used a systematic modeling strategy to model Ponzi contract detection based on the opcode sequence feature.

The code-based methods can realize the early detection of a Ponzi contract, but Ponzi contracts vary greatly in code implementation according to the research [5].

Compared with *existing methods based on code features*, PonziFinder is able to capture the characteristics of Ponzi contracts "using later stakeholders' funds to generate revenue for early stakeholders and operators" more precisely from the transaction flow, without the interference of the code implementation form.

2) Transaction-Based Features: Transactions are the execution result of a smart contract. Many researchers extract features from historical transaction information to detect Ponzi contracts. Jung et al. [17] extracted new transaction features combined with the Gini coefficient and conducted experiments, which strengthened the ordinary statistical transaction features, such as mean and variance. Yu et al. [16] provided a Ponzi contract detection method based on the GCN model, which used 14 transaction features.

The transaction-based methods can accurately detect the Ponzi contract according to the actual cash flow of contracts. However, existing methods have great information loss in extracting contract transaction features. For example, they solely rely on counting the number of transfer transactions to detect Ponzi contracts. However, our observations show that 68.5% of Ponzi contracts involve nontransfer transactions that serve as calling functions, and often, the traders are the primary beneficiaries of Ponzi schemes.

Different from *existing methods based on transaction features*, we build contract transaction graphs and classify contracts at the graph level to better preserve the original transaction network. Moreover, we add edge feature extraction to enhance the transaction network, and optimize the node features to improve the efficiency of our method. Finally, we innovatively extract information from input values in contract transactions, and the experiment indicates that the input value has a positive effect on Ponzi contracts detection, which makes up for the current transaction information loss caused by manual feature extraction.

3) Combined Features: Most of the current studies have been conducted by combining code features with transaction features. Specifically, Chen et al. [7] extracted code features by generating opcodes and calculating their frequencies. Then, they extracted statistical transaction features from the transaction history of smart contracts. Finally, the random forest algorithm was used as a classification model to detect Ponzi contracts [7]. Wang et al. [18] proposed a method to detect smart contracts by oversampling-based LSTM using opcode and transaction features. Zhang et al. [21] proposed an improved LightGBM algorithm using features extracted from opcodes, bytecodes, and transaction information. Liu et al. [19] constructed a heterogeneous information network using transaction features and code features.

The combined features methods take into account source code, bytecode, opcode, transaction data, and other useful information for Ponzi contracts detection, which can realize the Ponzi contract detection in the early stage and improve the accuracy. However, most existing methods based on combined features utilize code features and transaction features in a simple way, and the detection ability still needs to be improved. Some experiments [7] even suggest that adding transaction features has no positive effect on detection results, revealing that these methods still have room for improvement in dealing with code features and transaction features.

Compared with *existing methods based on combined features*, we only need to analyze the transaction flow information to achieve high accuracy, and do not need to obtain and analyze the contract code, bytecode, or other information, which saves time greatly. Moreover, the experiment proves that our scheme has great advantages in detecting Ponzi contracts with various forms when the contract transaction is sufficient. Our approach can provide a new perspective for utilizing transaction information, such as combining our extracted contract graph features with its code features to identify Ponzi contracts more accurately at an early stage.

VII. Conclusion

In this article, we propose a PonziFinder based on the EGNN(C) model for automatic detection of Ponzi contracts. This approach enables accurate detection without requiring access to the contract's source code, thereby improving detection efficiency. We extract node and edge features based on Ponzi contract characteristics and construct a transaction graph for each contract. The EGNN(C) model is used to propagate node information and layerwise graph representations are generated by aggregating node features with Set2Set. The ALGA mechanism is adopted to extract the final representation of the entire contract. Finally, we learn the flow of funds to detect Ponzi contracts.

We conduct comparative experiments with existing methods and other GNN models, which demonstrate that PonziFinder significantly outperforms others when contract transaction volume is sufficient. Furthermore, PonziFinder can detect all four classic types of Ponzi contracts and all other Ponzi contracts.

Our work has limitations, such as the inability to automatically detect Ponzi contracts at an early stage. However, this limitation is less impactful since Ponzi schemes rely on constantly attracting new stakeholders, and if there are no stakeholders, then the scam collapses. Moreover, we can solve this limitation by combining it with code information. Our approach has a deeper mining degree of transaction information compared to existing methods, which may promote current research progress on feature combinations.

Our work has significant implications for Ponzi contract detection. In future work, we will focus on improving PonziFinder's detection precision and efficiency.

References

[1] Y. Yuan and F. Wang, "Blockchain: The state of the art and future trends," *Acta Automatica Sinica*, vol. 42, pp. 481–494, 2016.

[2] B. Li, Z. Pan, and T. Hu, "ReDefender: Detecting reentrancy vulnerabilities in smart contracts automatically," *IEEE Trans. Rel.*, vol. 71, no. 2, pp. 984–999, Jun. 2022.

[3] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay, and M. Thompson, "Making smart contracts smarter," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency*, 2021, pp. 1–3.

[4] T. Hu, Z. Li, B. Li, and Q. Bao, "Contractual security and privacy security of smart contract: A system mapping study," *Chin. J. Comput.*, pp. 2485–2514, 2021.

[5] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, "Dissecting Ponzi schemes on Ethereum: Identification, analysis, and impact," *Future Gener. Comput. Syst.*, vol. 102, pp. 259–277, Jan. 2020.

[6] C. F. Torres, M. Steichen, and R. State, "The art of the scam: Demystifying honeypots in Ethereum smart contracts," in *Proc. 28th USENIX Conf. Secur. Symp.*, 2019, pp. 1591–1607.

[7] 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. 37575–37586, 2019.

[8] Y. Chen, H. Dai, X. Yu, W. Hu, Z. Xie, and C. Tan, "Improving Ponzi scheme contract detection using multi-channel TextCNN and transformer," *Sensors (Basel, Switzerland)*, vol. 21, 2021, Art. no. 6417.

[9] Y. Lou, Y. Zhang, and S. Chen, "Ponzi contracts detection based on improved convolutional neural network," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2020, pp. 353–360.

[10] L. Bian, L. Zhang, K. Zhao, H. Wang, and S. Gong, "Image-based scam detection method using an attention capsule network," *IEEE Access*, vol. 9, pp. 33654–33665, 2021.

[11] S. Fan, S. Fu, H. Xu, and X. Cheng, "AL-SPSD: Anti-leakage smart Ponzi schemes detection in blockchain," *Inf. Process. Manag.*, vol. 58, 2021, Art. no. 102587.

[12] W. Chen et al., "Sadponzi: Detecting and characterizing Ponzi schemes in Ethereum smart contracts," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 2, pp. 1–30, Jun. 2021.

[13] X. Shen, S. Jiang, and L. Zhang, "Mining bytecode features of smart contracts to detect Ponzi scheme on blockchain," *Comput. Model. Eng. Sci.*, vol. 127, pp. 1069–1085, 2021.

[14] J. Peng and G. Xiao, "Detection of smart Ponzi schemes using opcode," in *Blockchain and Trustworthy Systems*. Singapore: Springer 2020, pp. 192–204.

[15] W. Sun, G. Xu, Z. Yang, and Z. Chen, "Early detection of smart Ponzi scheme contracts based on behavior forest similarity," in *Proc. IEEE 20th Int. Conf. Softw. Qual., Rel. Secur.*, 2020, pp. 297–309.

[16] S. Yu, J. Jin, Y. Xie, J. Shen, and Q. Xuan, "Ponzi scheme detection in Ethereum transaction network," in *Blockchain and Trustworthy Systems*. Singapore: Springer 2021, pp. 175–186.

[17] E. B. Jung, M. L. Tilly, A. Gehani, and Y. Ge, "Data mining-based Ethereum fraud detection," in *Proc. IEEE Int. Conf. Blockchain*, 2019, pp. 266–273.

[18] L. Wang, H. Cheng, Z. Zheng, A. Yang, and X. Zhu, "Ponzi scheme detection via oversampling-based long short-term memory for smart contracts," Knowl.-Based Syst., vol. 228, no. C., Sep. 2021, Art. no. 107312.

[19] L. Liu, W. T. Tsai, M. Z. A. Bhuiyan, H. Peng, and M. Liu, "Blockchain-enabled fraud discovery through abnormal smart contract detection on Ethereum," *Future Gener. Comput. Syst.*, vol. 128, pp. 158–166, 2021.

[20] Y. Liang, W. Wu, K. Lei, and F. Wang, "Data-driven smart Ponzi scheme detection," *CoRR*, vol. abs/2108.09305, 2021. [Online]. Available: https://arxiv.org/abs/2108.09305

[21] Y. Zhang, W. Yu, Z. Li, S. Raza, and H. Cao, "Detecting Ethereum Ponzi schemes based on improved lightGBM algorithm," *IEEE Trans. Comput. Social Syst.*, vol. 9, no. 2, pp. 624–637, Apr. 2022.

[22] U.S. Securities and Exchange Commission, "Ponzi schemes: What you need to know." [Online]. Available: https://www.sec.gov/spotlight/enf-actions-ponzi.shtml

[23] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[24] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 12–26.

[25] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–17.

[26] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3734–3743.

[27] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9203–9211.

[28] M. Schlichtkrull et al., "Modeling relational data with graph convolutional networks," in *Proc. Extended Semantic Web Conf.*, 2017, pp. 593–607.

[29] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–11.

[30] W. Liu, M. Gong, Z. Tang, A. K. Qin, K. Sheng, and M. Xu, "Locality preserving dense graph convolutional networks with graph context-aware node representations," *Neural Netw.*, vol. 143, pp. 108–120, 2021.

[31] R. Barandela, J. S. Sánchez, V. García, and F. J. Ferri, "Learning from imbalanced sets through resampling and weighting," in *Proc. Iberian Conf. Pattern Recognit. Image Anal.*, 2003, pp. 80–88.

[32] A. Fernández, S. García, F. Herrera, and N. V. Chawla, "Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary," *J. Artif. Int. Res.*, vol. 61, no. 1, pp. 863–905, Jan. 2018.

[33] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

[34] Y. Zhang and Y. Lou, "Deep neural network based Ponzi scheme contract detection method," *Comput. Sci.*, vol. 48, pp. 273–279, 2021.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[36] Z. Zheng, W. Chen, Z. Zhong, Z. Chen, and Y. Lu, "Securing the Ethereum from smart Ponzi schemes: Identification using static features," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, pp. 1–28, 2023.

Yingying Chen is currently working toward the M.S. degree in ponzi scheme detection based on graph neural networks with the School of Computer Science and Engineering, Southeast University, Nanjing, China, under the supervision of Dr. Bixin Li.

Her research interests include blockchain security and software engineering, etc.

Bixin Li received the Ph.D degree from Nanjing University, Nanjing, China, in 2001.

He is currently a Full Professor with the School of Computer Science and Engineering, Southeast University, Nanjing, China, where he is also the header of the Software Engineering Institute. He is the Chairman of the Technology Committee of Software Engineering Standards of Jiangsu Province. His research interests include program slicing and its application, software evolution and maintenance, software testing and verification, software safety, and security techniques etc.

Yan Xiao received the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2019.

She is currently an Associate Professor with the School of Cyber Science and Technology, Shenzhen Campus of Sun Yat-sen University, Shenzhen, China. Her research interests include trustworthiness of deep learning system and AI applications in software engineering.

Xiaoning Du received the bachelor's degree in software engineering from Fudan University, Shanghai, China, in 2014, and the Ph.D. degree in computer science from Nanyang Technological University, Singapore, in 2020.

She is currently a Lecturer with Monash University, Clayton, VIC, Australia. She has authored or coauthored publications in top-tier venues including International Conference on Software Engineering, IEEE International Conference on Automated Software Engineering, ACM International Conference on the Foundations of Software Engineering, Conference on Neural Information Processing Systems, AAAI Conference on Artificial Intelligence, S&P, and TDSC. Her research interests include the security and quality assurance of both traditional software and intelligent software with learning-based components, which covers but is not limited to software testing, program analysis, vulnerability detection, and runtime verification.