Information Processing and Management 58 (2021) 102462

Contents lists available at ScienceDirect

Information Processing and Management

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

Transaction-based classification and detection approach for Ethereum smart contract

Teng Hu[a,b], Xiaolei Liu[b], Ting Chen[a], Xiaosong Zhang[a,c,*], Xiaoming Huang[d], Weina Niu[a], Jiazhong Lu[e], Kun Zhou[a,b], Yuan Liu[b]

[a] *Institute for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, China*
[b] *Institute of Computer Application, China Academy of Engineering Physics, China*
[c] *Cyberspace Security Research Center, Peng Cheng Laboratory, China*
[d] *CETC Cyberspace Security Research Institute Co., Ltd., China*
[e] *School of Cybersecurity, Chengdu University of Information Technology, China*

ARTICLE INFO

Keywords:
Blockchain
Ethereum
Smart contract
Classification
Security

ABSTRACT

Blockchain technology brings innovation to various industries. Ethereum is currently the second blockchain platform by market capitalization, it's also the largest smart contract blockchain platform. Smart contracts can simplify and accelerate the development of various applications, but they also bring some problems. For example, smart contracts are used to commit fraud, vulnerability contracts are deliberately developed to undermine fairness, and there are numerous duplicative contracts that waste performance with no actual purpose. In this paper, we propose a transaction-based classification and detection approach for Ethereum smart contract to address these issues. We collected over 10,000 smart contracts from Ethereum and focused on the data behavior generated by smart contracts and users. We identified four behavior patterns from the transactions by manual analysis, which can be used to distinguish the difference between different types of contracts. Then 14 basic features of a smart contract are constructed from these. To construct the experimental dataset, we propose a data slicing algorithm for slicing the collected smart contracts. After that, we use an LSTM network to train and test our datasets. The extensive experimental results show that our approach can distinguish different types of contracts and can be applied to anomaly detection and malicious contract identification with satisfactory precision, recall, and f1-score.

1. Introduction

Blockchain is a decentralized, distributed technology that is used to record the transactions (Li, Jiang, Chen, Luo, & Wen, 2020b). This brings the blockchain many benefits, such as decentralization, persistence, anonymity, and auditability (Zheng, Xie, Dai, Chen, & Wang, 2018). Therefore, blockchain technology has achieved rapid development in recent years, and application scenarios have also extended from the initial electronic virtual currency distribution transaction to finance, medicine, IoT, software engineering, and other fields (Li et al., 2020b). Bitcoin (Nakamoto, 2019) is the first successful application of blockchain technology. It was named the best

* Corresponding author at: Institute for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China

E-mail addresses: mailhuteng@gmail.com (T. Hu), johnsonzxs@uestc.edu.cn (X. Zhang).

https://doi.org/10.1016/j.ipm.2020.102462
Received 13 May 2020; Received in revised form 2 December 2020; Accepted 3 December 2020
Available online 17 December 2020

performing currency in 2015 (Desjardins, 2016) and the best performing commodity in 2016 (Ghosh, Gupta, Dua, & Kumar, 2020). Ethereum has a market capitalization of around 25 billion USD (as of June 2019), which is the second-largest blockchain platform after Bitcoin. At the same time, Ethereum is the first blockchain platform to support smart contracts, and also, it is the largest one (Chen et al., 2018a). As of June 2019, Ethereum had more than 67 million accounts, increasing approximately 70k new addresses per day, and there are millions of smart contracts in Ethereum (Wood et al., 2014). Many studies on security (Chen et al., 2017b; Li et al., 2020b), performance (Chen, Li, Luo, & Zhang, 2017a), and application (Chen, Srivastava, Parizi, Aloqaily, & Al Ridhawi, 2020a; Li, Wu, Jiang, & Srikanthan, 2020a; Oham, Michelin, Jurdak, Kanhere, & Jha, 2020) of Ethereum have already achieved great progress. Moreover, many organizations and businesses have started using blockchain technology to enhance the functionality and security of their information systems, as well as to expand new business (Baniata, Anaqreh, & Kertesz, 2021; Berdik, Otoum, Schmidt, Porter, & Jararweh, 2012; Zhao, Chen, Liu, Baker, & Zhang, 2020). For example, organizations and businesses can use DApp (i.e. Decentralized Applications) to achieve these goals, which is implemented through smart contracts. However, DApp may also introduce new vulnerabilities into the information system, as smart contracts themselves may have security issues.

Necessity: The identification and classification of smart contracts can help us to better understand the behavior of smart contracts and figure out the vulnerability, such as confirm fraud contracts. However, low attention has been paid to these works. We believe that the classification and detection approach in this paper is important from the aspects of enhancing security, performance, and management of blockchain-based information systems. The reasons are as follows:

- First of all, smart contracts are one of the most important components of the blockchain. It is also the basis for many blockchain-based systems to implement their functions. Studying the security of smart contracts is a necessary way to improve the security of blockchain systems.
- Classifying the contracts is the first step of malicious contract detection. Different contracts (eg., the detection of game contracts and finance contracts) have different behavior characters, thus the corresponding detection focus is different. The classify scheme and features extracted in this paper may provide a reference to studies of malicious smart contracts detection.
- Classifying and examining the most common application cases can help those new smart contract developers in specific fields to understand which areas are worth more investment.
- The deployer of a contract may conceal their true purpose, for example, some Ponzi contracts are masked by investment plan. The classification and detection approach in this paper can help users to verify the probability of a contract belonging to a certain type, thus contribute to discovering contracts with ambiguous purposes.
- Some contracts are flawed in their design and may consume too much GAS, thus causing financial loss to users, which simultaneously affects the performance of the blockchain. Our approach can help users to discover these contracts and thus avoid losses.
- The detection of malicious smart contracts can mitigate the security problems in the blockchain. Our detection approach can be extended to anomaly detection, malicious contract detection, and unknown smart contract identification, which can contribute to alleviating the security issues caused by smart contracts.

Contribution: In this paper, we focus on the research of data behavior generated by smart contracts and users in Ethereum, including the trend of Balance and the changes of Ether, transaction activities generated by users and smart contracts. In order to do this, we collected over 10,000 smart contracts from Ethereum and removed the contracts that contain too few transactions. Based on our analysis of these contracts, we find that the transaction behavior has independent characteristics between different types of contracts. For example, 1) Game-type contracts usually show the behavior of their Balance cliff-like reduction. That is because there is a winner in the game, and part of the Balance sends to the winner as a bonus. 2) In a Gambling-type contract, like coin flips, participants either win fixed-odds bonuses or lose all of their bets. So that the ether flow in and out shows a fixed ratio relationship. 3) In different types of contracts, the statistical features of transactions also can reflect their unique behavior. Some contracts have a large number of incoming transactions but have only a few numbers of outgoing transactions or even no outgoing transactions, such as a social contract. Other types of contracts may have a large gap between the number of external transactions and internal transactions. 4) Account activity can also reflect the behavior of different types of contracts. In a Ponzi contract, the sooner the account participates in the contract, the more times they are rewarded, which is significantly higher than those who participate later. This is consistent with the Ponzi scam phenomenon, in which the early participants benefit from the investments of later participants and it is less likely that the later participants will make a profit. In most cases, we can distinguish the types of contracts based on their single behavioral characteristics. However, sometimes we need multiple characteristics to determine the type of contract. For example, Gambling-type contracts may behave similarly to Game-type contracts.

In summary, we make the following major contributions.

(1) We collected more than 10,000 smart contracts from Ethereum and manually analyzed the behavior of their transactions. We finally find out four behavior patterns that can help us to have a better understanding of the contracts' transaction behavior, and they could be used to distinguish different types of contracts.
(2) To further research on this topic, we constructed 14 basic features to describe the transaction behavior. They are time-series data and constructed based on the major activities of a contract.
(3) We propose a data slicing approach for slicing the collected smart contracts to solve the problem of insufficient datasets. Then we use them for training an LSTM network and the results show the effectiveness of our approach.

The remaining of this paper is organized as follows. Section 2 introduces the background of the Ethereum, the account and smart contract, and the transaction of the contract. We describe in detail data collection, transaction analysis methods, data processing, and model in Section 3. After that, Section 4 summarizes the experimental configuration, the evaluation metrics, and the experiment results. We review related studies in Section 5 and conclude the paper in Section 6.

2. Background

2.1. Ethereum

Ethereum is an open-source blockchain platform proposed in 2014, which has now developed into the largest blockchain platform supporting smart contracts. It is also the blockchain platform with market capitalization second only to Bitcoin. As of this writing, the Ethereum blockchain has over 8 million blocks and each block in Ethereum contains many transactions (Team, 2017). Ethereum presented its dedicated cryptocurrency named Ether (referred to as "ETH"). Like other digital currencies, ETH can be traded on cryptocurrency exchanges, or developers pay ETH to keep their applications running, including transaction fees and computational services (Chen et al., 2018a). Compared with Bitcoin, another feature of Ethereum is the introduction of the Ethereum virtual machine (referred to as "EVM"). The EVM is the runtime environment for smart contracts (Solidity, 2019). Developers usually build smart contracts in a high-level language (e.g., Solidity (2019)) and then convert them to bytecode, which can be executed in EVM.

2.2. Account and smart contract

The account is the basic unit in Ethereum. Each account has a unique address with a length of 20 bytes. The state transitions in the Ethereum blockchain is the transfer of information and value between accounts (Buterin et al., 2014). There are two types of accounts: externally owned account (EOA) and contract account (CA). The main difference is that the CA contains executable code, while the EOA does not.

For the EOA, users can create an externally owned account at will and control it with a private key. To generate the address of the EOA, the public key is first generated by calculating the hash of the private key, and then the last 160 bits of the public key is used to form the EOA address (usually expressed as a 40-bit hexadecimal string, that is, 20 bytes in length). The address is the unique identifier of the externally owned account. Though EOA does not have executable code, it can be used to store current ether balances or to transfer Ether, deploy contracts, and call smart contracts by sending transactions.

The contract account, or called the smart contract, was first introduced in 1997 (Szabo, 1997), and now is a special account on Ethereum, controlled by the smart contract code. Like EOA, a smart contract also has a current ether balance field and a unique address, but its address is determined by the contract creator's address and the transaction that occurred at that time when the contract was created. In Ethereum, the smart contract is essentially an execution program composed of bytecode, which can be executed automatically when the trigger condition is met. Moreover, once the smart contract is deployed on Ethereum, even the developers themselves cannot modify it. Developers usually use high-level language for source code development, convert it into bytecode through the EVM compiler, and then upload the compiled bytecode to Ethereum through the client. After the smart contract is deployed in Ethereum, it is called and run in the EVM in the form of bytecode. In addition, the contract also has a database-like space called storage for storing persistent information (e.g., global variables, the bytecode for smart contracts) (Wood et al., 2014).

Smart contracts deployed in Ethereum cannot be actively executed. They need to be triggered by transaction or message invocation and will be run by EVM on every node participating in the network for verifying new blocks. Therefore, smart contracts are either called by externally owned accounts or by other smart contracts, but their specific calling methods are different. Although smart contracts cannot be modified after deployment, Ethereum allows contracts to be self-destructed, which can be used to stop buggy contracts, etc. When the contract self-destructs, its runtime bytecode will be removed from the blockchain, and the remaining Ether in the contract will be sent to the specified account (Wood et al., 2014). In general, smart contracts can be easily deployed in Ethereum. It can be said that the smart contract is a collection of function codes and data states stored in a specific address in the Ethereum blockchain (Solidity, 2019).

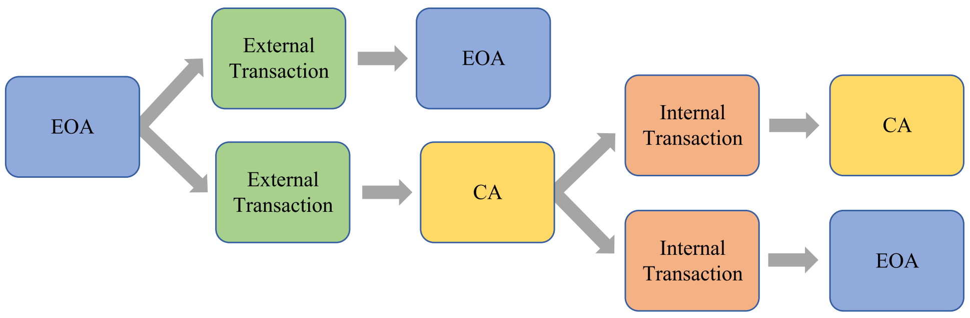

Fig. 1. The external transaction and internal transaction.

2.3. Transaction

The blockchain is a globally shared transactional database (Solidity, 2019). A block contains many transactions, and a transaction is a message sending to the blockchain, which carries import information (e.g., the function parameter, or a contract's bytecode, etc.). Anyone can synchronize data on the blockchain by participating in the network. However, if someone wants to change something in the blockchain, they must create a transaction that must be accepted by all the other participants. Also, when a transaction is applied in the blockchain, no other transaction can change it (Solidity, 2019).

In Ethereum, a transaction is a message sent from one account address to another. We focus on the transaction because Ethereum's primary activities are triggered by transactions, including Ether transfers, invoking smart contracts, and creating contracts. To avoid ambiguity, we refer to the transaction that from the EOA as an external transaction (Chen et al., 2019b), while the transaction sends from the smart contract to other accounts as the internal transaction. Fig. 1 shows the external transaction and internal transaction. Note that the internal transactions are not kept in the blockchain, so they cannot get directly through the parsing blocks (Chen et al., 2019b).

Each transaction contains some basic fields, where the *txdata* field contains the main transaction information, including: the *Recipient* field specifies a recipient address of a transaction; the *Signature* field specifies the signature of the transaction sender; The *Amount* field indicates the number of Ethers transferred, and the unit is Wei; the *Payload* field specifies the bytecode of the smart contract or the parameter when invoking the contract. Transaction delivery may also fail, such as insufficient transfer balance, insufficient GAS, invalid bytecode, and so on (Chen et al., 2018a). The effect of the applied transaction will be rolled back when it fails. Therefore, we analyze smart contracts only based on those successful transactions (Chen et al., 2018a). Creating a smart contract means deploying the contract on the blockchain, the creator of which can be EOA or another smart contract, and sending the transaction to realize the deployment. In this case, the *Recipient* field address should be empty, and the *Payload* field contains the bytecode of the contract. After successful deployment, any Ethereum user can invoke the contract by sending a transaction to the contract's address.

3. Data, analysis and model

This section outlines data collection, transaction analysis methods, data processing, and model. First, the source and collection method of smart contract data are briefly introduced. Then, the transaction characteristics of the contract are analyzed to find out the behavior characteristics that can reflect the smart contract. Finally, the data processing method and model are explained.

3.1. Data collection

We collect the contract data in two ways. One is to synchronize all historical transaction data through the Ethereum client. We use the Parity client and set it to tracing-on mode so that it can compute and store tracing data. All transaction data from Jul-30-2015 to Feb-1-2018 (about 5 million block height) were collected as the research object. The other is to use the APIs provided by Etherscan (Team, 2017) to download the transactions of a smart contract and store the data in JSON format. Although the provided API claims that it can only download the last 10,000 transactions, it is possible to obtain all transactions of a contract by modifying the block range. Due to network laggy, contract transactions after Feb-1-2018 are collected by the second method. According to the different applications of smart contracts, we also refer to the DApp publishing website (Dap, 2019; 2020; 2020; Sta, 2020) and finally sort out the six most common types in Ethereum. We collected a total of 11,593 contracts, and the detail is in Table 1.

3.2. Transactions-based analysis

Every transaction of a smart contract, since its creation, was kept on Ethereum. Therefore, we can restore various state changes and trends of smart contracts. For example, Fig. 2 shows a contract[1] on Ethereum called Wrapped Ether. We could restore all the transactions since its creation, including the external and the internal transactions. Also, we could distinguish the incoming transactions and outgoing transactions, which are invoked by EOAs or contracts.

Different types of contracts have different transaction behavior. The application purpose of a contract determines its specific behavior, which in turn helps us identify the type of the contract. Similarly, malicious contracts (e.g., Ponzi contracts) also have their own special behavior and can be detected. We manually investigated several contracts with different application purposes and summarized the behavior patterns from transactions which can reflect the differences between them.

3.2.1. Changes in balance

Balance refers to how much Ether is in an account at the time. The account includes EOAs and contracts, but we only consider the Balance of the contract here. Changes in Balance can reflect differences in different types of contracts. A game contract[2] usually has the behavior shows in Fig. 3.

With the addition of game participants, the contract's Balance grows steadily in a while. However, after a period of time, Ether will

[1] 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
[2] 0x7fDcD2a1E52F10C28cB7732f46393e297eCaDDa1

Table 1
The number of each type of contracts.

Types	Gambling	Game	Finance	Exchange	High-risk	Social	Total
Count	3056	2830	3997	682	790	238	11,593

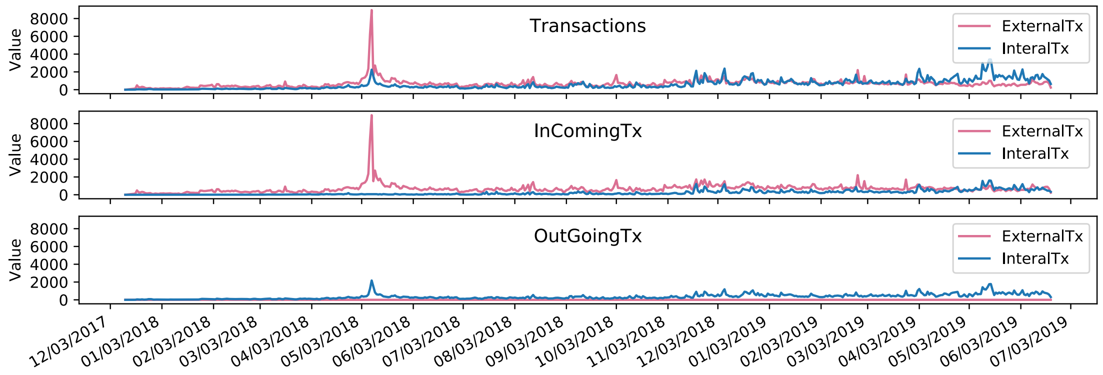

Fig. 2. The transactions of a contract on Ethereum since its creation.

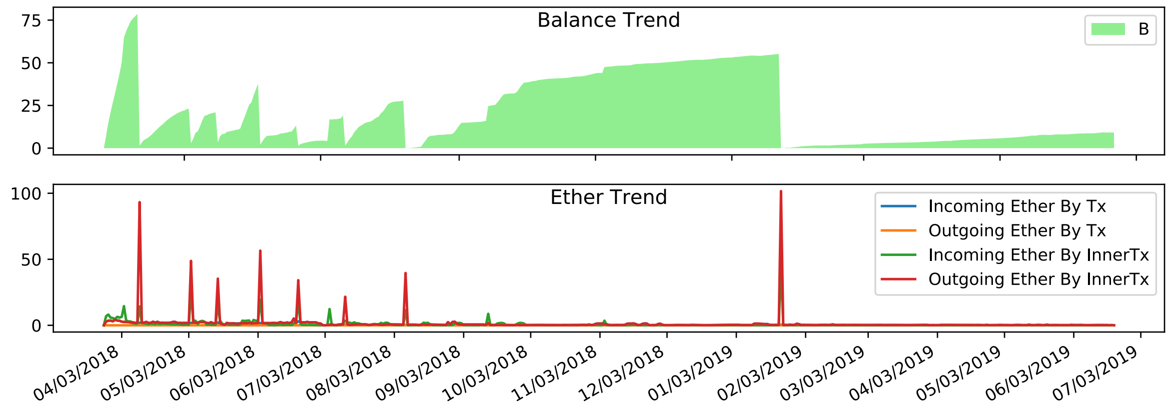

Fig. 3. A game contracts Balance trend and Ether trend.

have a “cliff” type of outflow, which is drastically reduced to a certain amount of Balance (and possibly reduced to zero), and this phenomenon will occur periodically. We found that this is because as the game progresses, the “winner” appears in the game, and the contract sends the Balance to the winner as a bonus. This is not or rarely the case in other types of contracts.

3.2.2. *Correlation between inflow ether and outflow ether*

There is a fixed correlation between the inflow and outflow of Ether in some contracts, and this correlation can reflect the independent transaction characteristics of this type of contract, such as the gambling contract. The Gambling contract on Ethereum can be roughly divided into several forms: lottery tickets, slot machines, coin flips, dice, etc. They all have a feature that the contract itself charges a gambling participant a certain percentage of commission. This means that in a gambling contract, there is a fixed ratio between the Ether bet and the Ether gained when participants win a bet. For example, a gambling player makes a coin-flip bet, the bet is X Ether, the winner will receive A times the bet bonus (assuming no extra fees), and the commission rate of the contract is $B\%$, then if the participant wins this bet, he will actually earn Y bonus. Hence, all players who win a bet bonus in this contract will have this feature: $Y = X * A - X * B\%$. Fig. 4 shows a gambling contract[3], the red line represents the relationship between the bet and the result, and presents a linear relationship, indicating that the ratio between them is fixed.

3.2.3. *The statistical features of transactions*

In different types of contracts, their transactions also have some unique statistical features. For example, in some social contracts,

[3] 0x777777c3b5dDE20E09Fe40B37622AE2Cd3b055b7

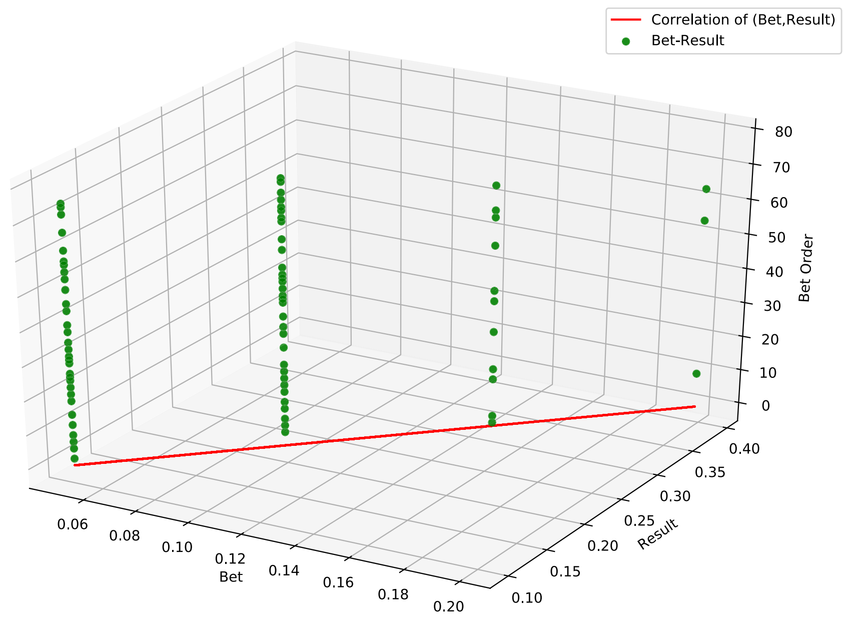

Fig. 4. Correlation between inflow Ether and outflow Ether of a gambling contract.

the primary behavior of users is to register accounts and publish information. So, there will be only incoming transactions and no outgoing transactions, and users do not need to transfer Ether when they register or post information. They only need to pay for transaction gas, so the contract balance is always 0, as an example[4] in Fig. 5. For some other types of contracts, the number of incoming transactions may be significantly higher than the number of outgoing transactions. There may also be a large gap between the number of external transactions and the number of internal transactions. Take a game contract[5] as an example, shows in Fig. 6, this phenomenon is common in a game and gambling contract.

3.2.4. *Features of account activity*

The activities of the accounts in a contract can also reflect the unique characteristics of the contract. For example, Figs. 7 and 8 are the distribution of accounts activities of a Ponzi contract[6] and a gambling contract[7], respectively.

Fig. 7 shows the frequency of investments and the amount of investment Ether of a Ponzi contract. The two sub-figures above represent the frequency of account investment (shown in red) and the frequency of investment return (shown in blue). The two sub-figures below represent the amount of Ether invested (shown in red) and the amount of Ether returned (shown in blue). In every sub-figure, each small square represents the statistics in one day. And we use the color depth to distinguish between them, the darker the color, the greater the number, and vice versa. From the top left to bottom right corner indicates the chronological order in the contract, i.e. the square in the top left corner represents the first day and the bottom right corner represents the last day. The following features can be observed. First, as can be seen from the two sub-figures above, the small squares in the red sub-figure are very light in color, but the corresponding blue sub-figure are very dark in color, especially the first one is the most obvious. This shows that the earlier the account invested in the contract, the higher the frequency of returns, which is significantly higher than those who participated in the contract later. Second, from the two sub-figures below, most of the small squares in the front have color in the red sub-figure and the blue sub-figure. The small squares in the back are colored in the red sub-figure, while the corresponding small squares in the blue sub-picture almost have no color. This indicates that the early investment accounts have a higher proportion of return, while the late investment accounts have almost no investment income. Both of the points are consistent with the Ponzi scam phenomenon.

In other types of contracts, for example, Fig. 8 is a gambling contract. First of all, from the two sub-figures above, most of the small squares in the red sub-figure have roughly the same shades of color as the small squares in the blue sub-picture. However, there are also cases where the red sub-figure has darker small squares and the corresponding blue sub-figure has lighter small squares. This shows that the frequency of user investment and return is basically proportional, i.e., the probability of getting a return is higher if you bet more often. Of course, it is also possible to bet and keep losing without any return. This is pretty much the same as in real gambling. Secondly, the two sub-figures above also show another phenomenon that is consistent with real gambling. The number of bets placed is far greater than the number of winnings from an overall perspective. Thirdly, from the two sub-figures below, the small squares in the

[4] 0x5d4ABC77B8405aD177d8ac6682D584ecbFd46CEc
[5] 0x91b9d2835ad914bc1dcfe09bd1816febd04fd689
[6] 0xe82719202e5965Cf5D9B6673B7503a3b92DE20be
[7] 0x777777d1123591df7f68e053bc182de3cd4fa6de

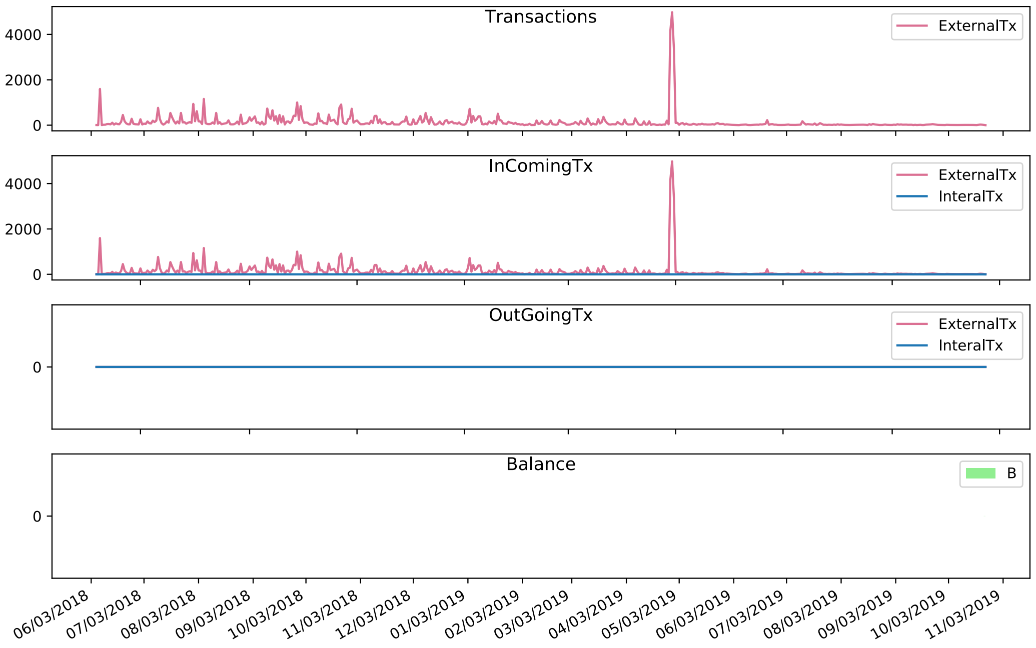

Fig. 5. The transactions' statistical feature of a social contract.

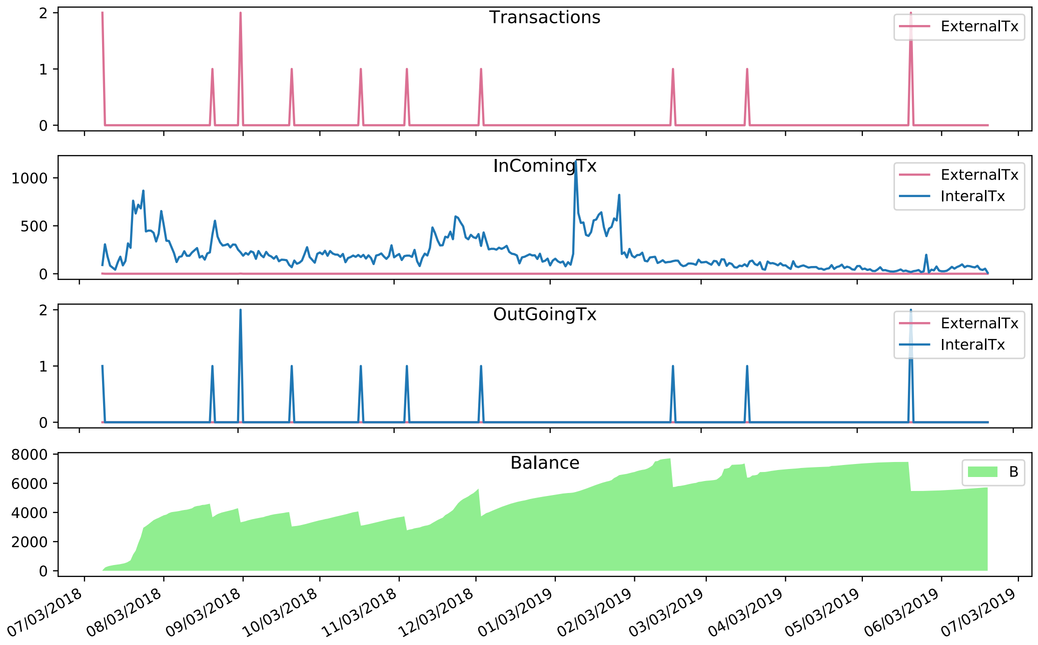

Fig. 6. The transactions' statistical feature of a game contract.

blue sub-figure corresponding to the red sub-figure are either slightly darker or absent. This suggests the Ether betted by the account either receives a fixed return or has no income, which is consistent with the characteristic of gambling.

3.2.5. Transactions-based features

According to the four behavior patterns of smart contracts we found, it is clear that transaction-based analysis can indeed reflect the characteristics of different application types of contracts. However, a single feature cannot completely judge the category of a contract accurately, so it is necessary to combine multiple behavioral features to improve the accuracy of the judgment.

Through statistical analysis of a large number of smart contract transactions on Ethereum, we constructed the following 14 basic

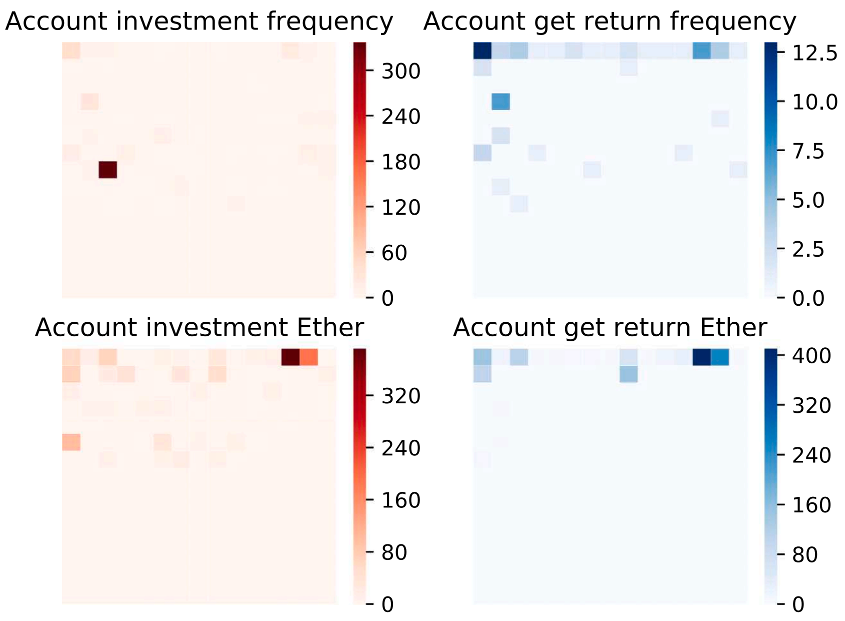

Fig. 7. The account activity features of a Ponzi contract.

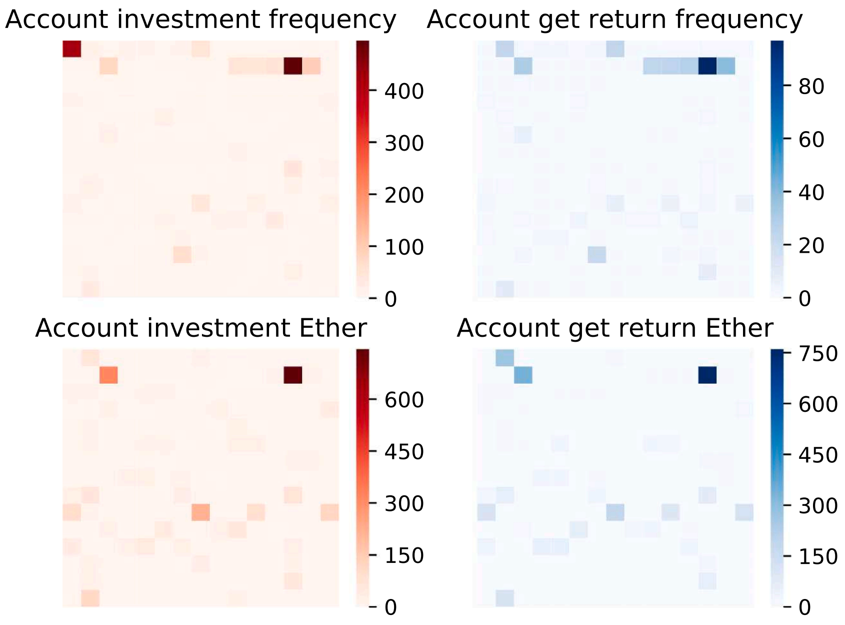

Fig. 8. The account activity features of a gambling contract.

features that can truly reflect smart contracts. These 14 features are constructed in the chronological order in which contract transactions occur, so they are time-series data. These features are as follows.

Feature 1: The number of a contract's Balance;
Feature 2: The number of inflow Ether from an external transaction;
Feature 3: The number of inflow Ether from an internal transaction;
Feature 4: The number of outflow Ether from an external transaction;
Feature 5: The number of outflow Ether from an internal transaction;
Feature 6: The number of a contract's total transactions;
Feature 7: The number of incoming transactions of external transactions;

Feature 8:	The number of outgoing transactions of external transactions;
Feature 9:	The number of incoming transactions of internal transactions;
Feature 10:	The number of outgoing transactions of internal transactions;
Feature 11:	The number of unique incoming addresses of external transactions;
Feature 12:	The number of unique outgoing addresses of external transactions;
Feature 13:	The number of unique incoming addresses of internal transactions;
Feature 14:	The number of unique outgoing addresses of internal transactions.

3.3. *Data preprocessing and model*

3.3.1. *Data preprocessing*

Based on these 14 features, we can distinguish between two different types of contracts. We take the 14 features of a contract as time-series data. Therefore, each contract can actually be regarded as a two-dimensional matrix. Each row is used to represent the 14 features mentioned above, i.e. there are 14 rows in total. Each column represents each time interval.

Because the creation time of each contract is different, the number, time, frequency of the generated transactions are different, that is, the number of columns of each matrix is different. The transactions of each contract need to be sliced. The method and length of the slice will affect the accuracy and efficiency of model classification. It is worth noting that some contracts generate too little data and only have a few transactions. We believe this kind of data cannot truly represent the behavior characteristics of a contract. Therefore, we will ignore such contracts in subsequent experiments.

Fig. 9 shows the flow of data preprocessing and Algorithm 1 describes the data slicing algorithm. The process of the data slicing algorithm can be summarized as the following: 1) Set a timestep according to a fixed time T. Then, the data generated in a timestep is merged into one. A contract can be represented by a two-dimensional array $(n, 14)$ with a size of $n * 14$, where n indicates that there are n timesteps, which is also the length of the array, 14 means that each n has 14 data, i.e. 14 feature data on this timestep. 2) Set a minimum data length $nMin$ and a maximum data length $nMax$. A contract with data length n less than $nMin$ will be ignored and will not participate in subsequent experiments. 3) For the case of $nMin <= n <= nMax$, padding the insufficient data with 0, and the array size

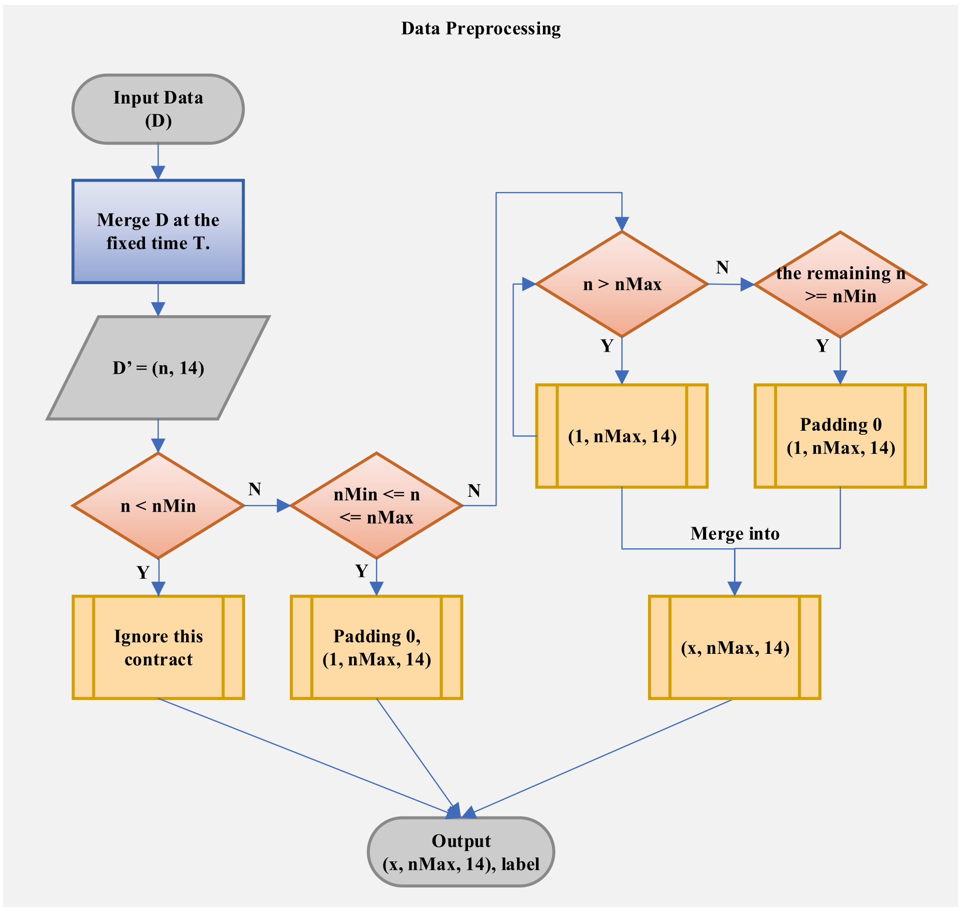

Fig. 9. The flowchart of data preprocessing.

```
Input: contract data: $D$, fixed time: $T$, minimum data length: $nMin$, maximum data length: $nMax$, timesteps interval: $m$
Output: the preprocessed data: $(x, nMax, 14)$
 1: Merge $D$ at the fixed time $T$. $\leftarrow$ Generate a new $D'$ with a length of $n$.
 2: The $D'$ can be represented by a two-dimensional array $(n, 14)$ $\leftarrow$ $n$ indicates that there are $n$ timesteps.
 3: if $n < nMin$ then
 4:     The contract with data length $n$ less than $nMin$ will be ignored and will not participate in subsequent experiments.
 5:     return
 6: else if $nMin \le n \le nMax$ then
 7:     Padding the insufficient data with 0, and the array size becomes $(nMax, 14)$.
 8:     return $(1, nMax, 14)$ $\leftarrow$ Generate a three-dimensional array, 1 means the contract has only one data slice.
 9: else if $n > nMax$ then
10:     while Every $m$ timesteps interval do
11:         Get A slice of data of length $xMax$. $\leftarrow$ Each slice is a two-dimensional array $(nMax, 14)$.
12:         if the remaining $n \ge nMin$ then
13:             continue. $\leftarrow$ the data length less than $nMax$ is still padding with 0
14:         else
15:             return $(x, nMax, 14)$ $\leftarrow$ Generate a three-dimensional array, $x$ represents the contract has a total of $x$ data slices.
16:         end if
17:     end while
18: end if
```

Algorithm 1. Algorithm for Data Slicing.

becomes $(nMax, 14)$. Generate a three-dimensional array $(1, nMax, 14)$, 1 means the contract has only one data slice. 4) For the case of $n > nMax$. From the beginning of the data, obtain a slice of data $(nMax, 14)$ at every m timesteps interval, until the remaining n is less than $nMin$ (the data length less than $nMax$ is still padding with 0). Therefore, one contract can generate multiple $(nMax, 14)$ at this time. This has the advantage of increasing the amount of data in the experiment. Converting all the slices into an array can form a three-dimensional array $(x, nMax, 14)$, where x represents a total of x data slices. The above data slicing algorithm can complete the preprocessing of the data. The specific value of $T, nMin, nMax$, and m in the algorithm will be mentioned in Section 4.

3.3.2. Model structure and training parameters

We process the smart contract into many time series slices, each of which is part of a contract data and can be regarded as a mapping of the characteristic behavior of a contract. Fig. 10 shows the flow of dataset generation and model training. First, take each type of contract as input data separately and each contract in the type is processed by the data slicing algorithm (as mentioned in Section 3.3.1). Then, the $(x, nMax, 14)$ generated from all the contracts is finally combined into one $(X, nMax, 14)$. Repeat the above process until all types have been processed. Finally, we can obtain the dataset $((X, nMax, 14), label)$ containing type labels for training models and experiments. After that, each type in the $((X, nMax, 14), label)$ dataset is divided into a training dataset and a test dataset at a ratio of 7:3, and set aside 20% from the training dataset as the validation dataset. Note that the data should be randomly taken during division to prevent overfitting. The training dataset is used to train the model, the validation dataset is used to tune and optimize the model parameters during training, and the test dataset is used to test the effectiveness of the model.

We use the LSTM (Hochreiter & Schmidhuber, 1997) network to train our dataset and conduct the experiments. The specific training parameters are set as follows (based on Keras and TensorFlow): $dropout = 0.2$ and $recurrent\_dropout = 0.2$, these two parameters are used to mitigate overfitting; the loss function uses *binary_crossentropy* and the optimizer uses *Adam*; the Dense layer sets the $activation =$ '*sigmoid*' for outputting the result. Other parameters are given in each set of experiments in Section 4, including T, $nMin$, $nMax$, m, $batch\_size$, $epochs$.

4. Experimental and results analysis

4.1. Evaluation metrics

We use the $Precision(P)$, $Recall(R)$, and $F1 - score(F1)$ to evaluate the performance of our models. *TP* (true positive) refers to the number of correct predictions for smart contracts. *FP* (false positive) refers to the number of misclassifications of other types to this type. And *FN* (false negative) refers to the number of misclassifications of this type to other types. The higher the value of *precision*,

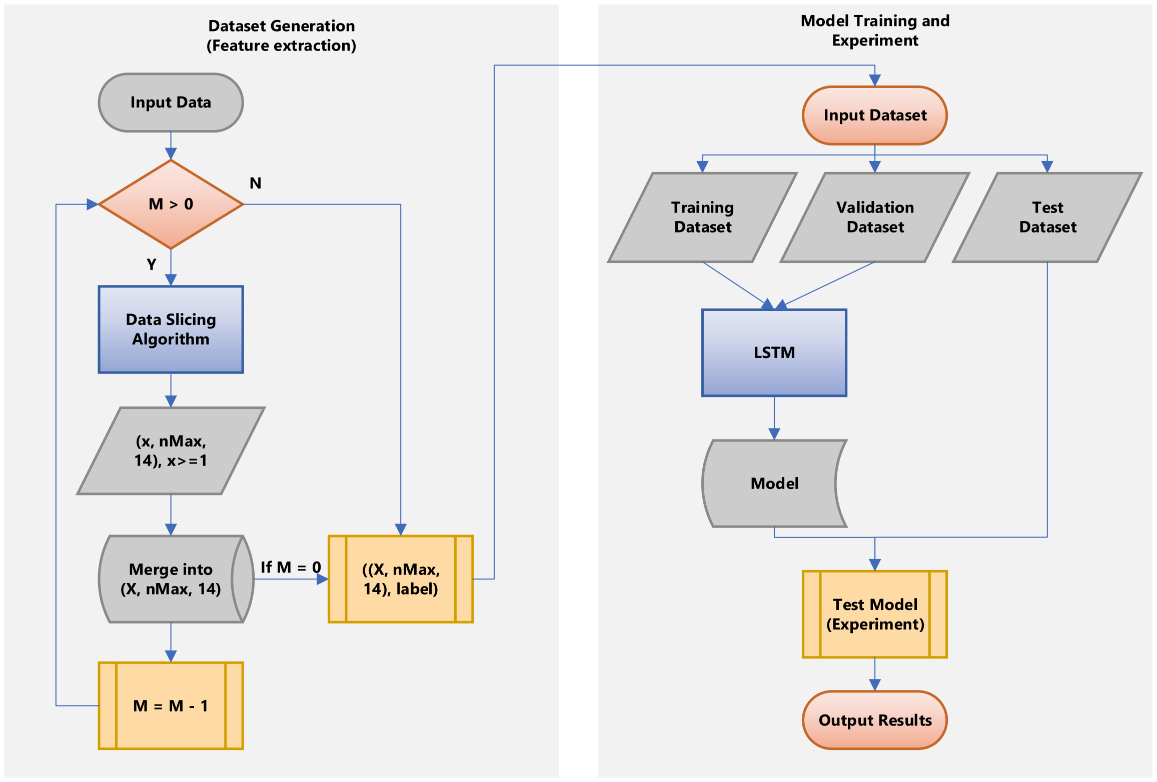

Fig. 10. The flowchart of dataset generation and model training.

recall, and $F1$, the better to distinguish the different types of smart contracts.

$$Precision(P) = \frac{TP}{TP + FP}$$

$$Recall(R) = \frac{TP}{TP + FN}$$

$$F1 - score(F1) = 2 \times \frac{P \times R}{P + R}$$

4.2. Experiment 1: evaluation of differences between different types of contracts

In order to verify our analysis conclusions on different types of smart contracts in Section 3, we performed pairwise classification experiments on different types of contracts. Specifically, the experimental parameters are set as follows: $T = 24(hours)$, $nMin = 64$, $nMax = 128$, $m = 10$, $batch\_size = 64$, $epochs = 2000$. T refers to timesteps, *nMin* means a minimum data length and *nMax* a maximum data length, *m* stands for *m* timesteps. These four parameters are mentioned in Section 3.3. And *batch_size* is the batch size when training the LSTM network. The number of *epochs* is a hyperparameter that defines the number of times that the learning algorithm will work through the entire training dataset. Each group of experiments was repeated ten times. The results of the mean (m) and standard deviation (std) are summarized in the form of m (+/- std) in Tables 2–7.

Tables 2 –7 respectively showed the difference between one type and other different types of contracts, which illustrated that the features we extract from the contract transaction can characterize the behavior patterns of different types of contracts. Taking Table 2 as an example, the Precision of Game-type contracts relative to other types of contracts is between 0.902–0.955, which means that more than 90% of Game-type contracts can be distinguished from other types by the features. The Recall ranged between 0.909 and 0.967, meaning that more than 90% of Game-type contracts were correctly identified as Game-type. The F1-score is between 0.906–0.972, indicating that the classification effect of our model is well. We use all these 14 features to perform our experiment and the results show that they can truly reflect the characteristics of different types of contracts. But we also found that if some contracts have too few transactions, it will affect the accuracy of the experimental results. This approach can be extended to anomaly detection, malicious contract detection, and unknown smart contract identification.

4.3. Experiment 2: applied to anomaly detection

In practical applications, the issuer of the contract may conceal the true purpose of the contract, such as some contracts that claim to be an investment plan but turn out to be typical ponzi scams. Another example is that some gambling contracts actually have a much lower win rate than they claim. This experiment can be used to verify the possibility that a contract belongs to a certain type, and thus to detect contracts with unclear purposes. We set one type of contract to be a positive dataset, and all the remaining contracts to be negative dataset. We performed six groups of experiments, and each group of experiments was repeated ten times. The results of the mean (m) and standard deviation (std) are summarized in the form of m (+/- std) in Table 8. Specifically, the experimental parameters are set as follows: $T = 24(hours)$, $nMin = 64$, $nMax = 128$, $m = 10$, $batch\_size = 64$, $epochs = 5000$.

From Table 8, it can be seen that Precision ranges from 0.827 to 0.965, i.e., the detection results are 82.7%-96.5% correct, with the highest being the High-risk-type and the lowest being the Game-type. For example, in the test results of the Game-type, 82.7% of the contracts are of the Game-type, and the remaining 17.3% are misidentified as the Game-type. The Recall is between 0.696 and 0.876, which means that a contract's type is correctly identified ranged from 69.6% to 87.6%, with the highest being High-risk-type and the lowest being Social-type. For example, in the Game-type samples, 83.8% of Game-type contracts are correctly identified, and the remaining 16.2% of Game-type contracts are identified as other types of contracts. The F1-score is between 0.776 and 0.918. Compared with Experiment A, the overall P, R, and F1-score has decreased, which is caused by the imbalance of the datasets. But this does not hinder the effectiveness of this method for detecting the contract with an unclear purpose.

4.4. Experiment 3: malicious contract identification

Our method can also be used to detect malicious contracts, such as Ponzi contracts. We use the Ponzi contract data from Paper

Table 2
The results of Game type contracts and other types of contracts.

	Game		
	Precision	***Recall***	***F1-score***
Gambling	0.939 (+/-0.004)	0.967 (+/-0.003)	0.953 (+/-0.002)
Exchange	0.902 (+/-0.003)	0.909 (+/-0.005)	0.906 (+/-0.003)
Finance	0.934 (+/-0.003)	0.956 (+/-0.002)	0.945 (+/-0.002)
High-risk	0.967 (+/-0.002)	0.977 (+/-0.001)	0.972 (+/-0.001)
Social	0.955 (+/-0.001)	0.964 (+/-0.003)	0.959 (+/-0.001)

Table 3
The results of Gambling type contracts and other types of contracts.

	Gambling		
	Precision	***Recall***	***F1-score***
Game	0.922 (+/-0.010)	0.816 (+/-0.015)	0.866 (+/-0.009)
Exchange	0.935 (+/-0.007)	0.866 (+/-0.014)	0.899 (+/-0.007)
Finance	0.945 (+/-0.005)	0.914 (+/-0.008)	0.929 (+/-0.005)
High-risk	0.950 (+/-0.007)	0.946 (+/-0.006)	0.948 (+/-0.005)
Social	0.950 (+/-0.005)	0.933 (+/-0.009)	0.941 (+/-0.006)

Table 4
The results of Exchange type contracts and other types of contracts.

	Exchange		
	Precision	***Recall***	***F1-score***
Game	0.910 (+/-0.005)	0.903 (+/-0.003)	0.907 (+/-0.003)
Gambling	0.950 (+/-0.004)	0.969 (+/-0.002)	0.959 (+/-0.002)
Finance	0.952 (+/-0.002)	0.967 (+/-0.002)	0.959 (+/-0.002)
High-risk	0.969 (+/-0.002)	0.976 (+/-0.001)	0.973 (+/-0.001)
Social	0.961 (+/-0.003)	0.974 (+/-0.001)	0.967 (+/-0.001)

Table 5
The results of Finance type contracts and other types of contracts.

	Finance		
	Precision	***Recall***	***F1-score***
Game	0.891 (+/-0.006)	0.816 (+/-0.012)	0.852 (+/-0.007)
Gambling	0.918 (+/-0.007)	0.947 (+/-0.005)	0.932 (+/-0.005)
Exchange	0.922 (+/-0.009)	0.861 (+/-0.010)	0.890 (+/-0.008)
High-risk	0.953 (+/-0.004)	0.965 (+/-0.003)	0.959 (+/-0.002)
Social	0.916 (+/-0.010)	0.923 (+/-0.013)	0.920 (+/-0.008)

Table 6
The results of High-Risk type contracts and other types of contracts.

	High-Risk		
	Precision	***Recall***	***F1-score***
Game	0.955 (+/-0.006)	0.872 (+/-0.018)	0.912 (+/-0.009)
Gambling	0.916 (+/-0.011)	0.924 (+/-0.014)	0.920 (+/-0.010)
Exchange	0.955 (+/-0.007)	0.910 (+/-0.012)	0.932 (+/-0.008)
Finance	0.951 (+/-0.006)	0.930 (+/-0.008)	0.941 (+/-0.004)
Social	0.954 (+/-0.009)	0.955 (+/-0.011)	0.954 (+/-0.007)

Table 7
The results of Social type contracts and other types of contracts.

	Social		
	Precision	***Recall***	***F1-score***
Game	0.875 (+/-0.015)	0.827 (+/-0.010)	0.850 (+/-0.005)
Gambling	0.904 (+/-0.013)	0.931 (+/-0.009)	0.917 (+/-0.009)
Exchange	0.942 (+/-0.006)	0.859 (+/-0.017)	0.898 (+/-0.009)
Finance	0.882 (+/-0.021)	0.870 (+/-0.019)	0.876 (+/-0.014)
High-risk	0.956 (+/-0.010)	0.955 (+/-0.009)	0.956 (+/-0.006)

(Chen, Zheng, Ngai, Zheng, & Zhou, 2019d) as a malicious contract dataset for our experiments. However, the dataset of Ponzi contracts is too small for training an LSTM network. Even our data slicing algorithm is used, it is still too small compared to other contracts. Therefore, we increase the training data by reducing nMin, due to the decrease of nMin, each data slice may not truly represent the characteristics of the contract behavior. We set Ponzi dataset to be a positive dataset and another type of contract to be a negative dataset. We performed six groups of experiments, and each group of experiments was repeated ten times. The results of the

Table 8
The results of the difference between one type of contract and the rest.

	Others		
	Precision	***Recall***	***F1-score***
Game	0.827 (+/-0.003)	0.838 (+/-0.005)	0.833 (+/-0.004)
Gambling	0.915 (+/-0.007)	0.786 (+/-0.012)	0.845 (+/-0.005)
Exchange	0.930 (+/-0.003)	0.848 (+/-0.006)	0.887 (+/-0.004)
Finance	0.909 (+/-0.011)	0.708 (+/-0.016)	0.796 (+/-0.012)
High-risk	0.965 (+/-0.005)	0.876 (+/-0.027)	0.918 (+/-0.015)
Social	0.878 (+/-0.010)	0.696 (+/-0.016)	0.776 (+/-0.010)

mean (m) and standard deviation (std) are summarized in the form of m (+/- std) in Table 9. Specifically, the experimental parameters are set as follows: $T = 24(hours)$, $nMin = 10$, $nMax = 128$, $m = 10$, $batch\_size = 64$, $epochs = 5000$.

In Table 9, the F1-score is between 0.691-0.825 and the overall results are not very satisfactory. This is caused by the Ponzi dataset is too small, resulting in insufficient training of the LSTM network. Nevertheless, in the absence of contracts' source code, our method can still effectively detect Ponzi malicious contracts. From Table 9, it can be seen that Precision ranges between 0.795 and 0.925, i.e., the correct rate is 79.5%–92.5%, with the highest being the High-risk-type and the lowest being the Exchange-type. The Recall is between 0.553–0.778, meaning that a contract's type is correctly identified between 55.3% and 77.8%, with the highest being the Social-type and the lowest being the High-risk-type. The relatively high Precision and low Recall indicate that Ponzi contracts have high similarity in behavior patterns with other types of contracts. For example, the Recall of the High-risk-type is the lowest, which is consistent with reality, as Ponzi contracts are also often classified as High-risk-type. This may also indicate the existence of undisclosed Ponzi contracts in the High-risk-type. In addition, Ponzi contracts are also often disguised as Game-type and Gambling-type, so the Recall is also lower with Game-type and Gambling-type. We repeat the experiment using an improved model of LSTM, i.e. the GRU (Chung, Gulcehre, Cho, & Bengio, 2014) model, with the same parameter settings and experiment environment. The results are shown in Table 10. The results are slightly worse than those of the LSTM model. On our dataset, the time cost of GRU training is about 25% less than that of LSTM. This is because GRU reduces the computational complexity compared to LSTM, but can obtain training results close to those of LSTM. In Table 11 (our results are expressed in average.), the experimental results of our method are compared with the method of Chen et al. (2019d). Our method has better P, R, and $F1$ when (Chen et al., 2019d) only uses account features. Their results are better than ours after adding the opcode feature to its method. However, only a few contracts in Ethereum will publish the source code, which has a significant impact on the practicality of the method of Chen et al. (2019d), so our approach has better versatility.

5. Related work

The Ethereum platform can be viewed as a transaction-based state machine, which begins with a genesis state and incrementally executes transactions to morph it into some final state (Wood et al., 2014). Therefore, many researchers have a deeper insight into blockchain technology by studying transaction data on Ethereum. The transaction analysis research can also reveal the behavioral patterns on the blockchain, verify existing theories, or prove the effectiveness of new methods. In general, they can be broadly divided into four types of research directions, data acquisition study, empirical study, performance and security study, prediction and classification study.

5.1. Data acquisition study

Ethereum contains a large amount of heterogeneous data, so it is not easy to efficiently obtain data while ensuring its completeness and correctness. Thus, many researchers have focused on data acquisition on the Ethereum platform and use them to support further study. Such as Kiffer, Levin, and Mislove (2017) and Nikolić, Kolluri, Sergey, Saxena, and Hobor (2018) collects transaction data by downloading and parsing block files. This method is simple and fast to implement, but they can't collect complete data. That's because the internal transactions are not stored in the blockchain and hence cannot be obtained by parsing blocks. This method will miss the interactions among smart contracts. Ethereum platform provides detailed Web3 APIs, and some researchers directly invoke them to collect data (Bartoletti, Lande, Pompianu, & Bracciali, 2017; Chen et al., 2017b). But there are two obvious shortcomings. One is that some of the APIs return data slowly due to their design, and the other is that they cannot obtain internal transactions. There are also some researchers to obtain data by crawling Ethereum explorer websites. Kalra, Goel, Dhawan, and Sharma (2018) and Bartoletti, Carta, Cimoli, and Saia (2020) both crawl relevant information from Etherscan (Team, 2017) and no secondary processing is required, which is relatively convenient. However, the website's server usually only lists partial data and discourage automated crawlers to guarantee the availability to all visitors. Some other researchers obtain data by instrumenting Ethereum nodes (Chen et al., 2019b; 2018a; Grossman et al., 2017; Zheng, Zheng, Dai, Chen, & Zheng, 2020). This method can collect more complete and accurate data than the other three methods. But the effectiveness of this method depends on the knowledge of Ethereum that researchers and developers possess. Chen et al., in their previous work (Chen et al., 2018a), only collects the sender, receiver, and amount of money transferred in each transaction to investigate money transfer, contract creation, and contract invocation. But in their subsequent work (Chen et al., 2019b), they propose DataEther, a systematic and high-fidelity data exploration framework for Ethereum by exploiting its internal mechanisms. For now, Dataether is more efficient and comprehensive than other methods in terms of collecting data on

Table 9
The results of Ponzi type contracts and other types of contracts (with LSTM).

	Ponzi		
	Precision	***Recall***	***F1-score***
Game	0.873 (+/-0.038)	0.688 (+/-0.046)	0.768 (+/-0.029)
Gambling	0.889 (+/-0.037)	0.658 (+/-0.046)	0.755 (+/-0.034)
Exchange	0.795 (+/-0.042)	0.773 (+/-0.026)	0.783 (+/-0.029)
Finance	0.878 (+/-0.043)	0.722 (+/-0.039)	0.791 (+/-0.028)
High-risk	0.925 (+/-0.041)	0.553 (+/-0.056)	0.691 (+/-0.045)
Social	0.883 (+/-0.037)	0.778 (+/-0.055)	0.825 (+/-0.027)

Table 10
The results of Ponzi type contracts and other types of contracts (with GRU).

	Ponzi		
	Precision	***Recall***	***F1-score***
Game	0.867(+/-0.043)	0.664(+/-0.082)	0.761(+/-0.057)
Gambling	0.869(+/-0.034)	0.640(+/-0.044)	0.737(+/-0.039)
Exchange	0.793(+/-0.038)	0.802(+/-0.036)	0.797(+/-0.027)
Finance	0.881(+/-0.041)	0.731(+/-0.041)	0.798(+/-0.027)
High-risk	0.920(+/-0.028)	0.532(+/-0.042)	0.674(+/-0.032)
Social	0.873(+/-0.035)	0.762(+/-0.041)	0.813(+/-0.022)

Table 11
Compare with the results in Chen et al. (2019d).

	Precision	***Recall***	***F1-score***	***Features***
Chen et al. (2019d)	0.59	0.22	0.32	Account(XGBoost)
	0.91	0.73	0.81	Code(XGBoost)
	0.90	0.67	0.76	Account+Code(XGBoost)
	0.64	0.20	0.30	Account(RF)
	0.94	0.73	0.82	Code(RF)
	0.95	0.69	0.79	Account+Code(RF)
Ours	0.88	0.70	0.77	14 Features(LSTM)
	0.86	0.68	0.76	14 Features(GRU)

Ethereum.

5.2. Empirical study

There are many phenomena on Ethereum that can reflect user characteristics, economic behavior patterns, and mathematical problems. Therefore, some researchers have an empirical study by analyzing data from the Ethereum. Kiffer et al. (2017) explores the consequences of the hard fork, showing how the fork leads to unintentional incentives and security vulnerabilities. Bai, Zhang, Xu, Chen, and Wang (2020) study the evolutionary behavior of Ethereum transactions from a temporal graph point of view and found that the user's wealth on Ethereum is very unfair from the very beginning. Ferretti and D'Angelo (2020) analyze the Ethereum blockchain with the complex networks modeling framework to verify if radical changes in the blockchain evolution happened. Somin, Gordon, and Altshuler (2018) analyze the network properties of the ERC20 protocol compliant crypto-coins' transactions, which demonstrates that the network displays strong power-law properties and these results coincide with current network theory expectations. Pierro and Rocha (2019) aim to investigate whether and to what extent different variables influence the Ethereum transaction fees, and they shed light on how different variables might interact and influence the Gas Price. Spain, Foley, and Gramoli (2020) study the impact of a series of ICOs to understand the relationship between transaction fees, throughput, and latency in Ethereum and results showed that transaction costs are inversely proportional to waiting time. Liang, Li, and Zeng (2018) conduct a dynamic network analysis of three representative blockchain-based cryptocurrencies, which reflect the evolutionary characteristics and competitiveness of different cryptocurrencies. Sovbetov (2018) examines factors that influence the most common five cryptocurrencies price, and the relationship between the price of cryptocurrency and the crypto market. Fenu, Marchesi, Marchesi, and Tonelli (2018) analyze the factors that influence the success of ICO, which shows that the success of ICO has a certain relationship with the evaluation of related websites and the management of ICO tokens on the Ethereum blockchain. Torres, Steichen et al. (2019) conduct the systematic analysis of honeypot smart contracts by studying the universality and behavior of honeypot smart contracts and their impact on the Ethereum blockchain.

5.3. Performance and security study

The security and performance of blockchain have always been a hot topic of concern, and researchers have already achieved many results in this aspect. For example, Chen et al. (2017b) have done many works to alleviate the security issues of Ethereum. They propose a novel gas cost mechanism in 2017, which dynamically adjusts the costs of EVM operations according to the number of executions, to thwart DoS attacks. And in 2018, their group use graph analysis to conduct the first systematic study on Ethereum and proposes new approaches based on cross-graph analysis to address two security issues (Chen et al., 2018a). Followed in 2019, they investigate such inconsistent token behaviors with regard to ERC-20, the most popular token standard, and revealed 11 major reasons behind the inconsistency. Then they propose a novel approach to automatically detect such inconsistency by contrasting the behaviors derived from three different sources (Chen et al., 2019c). In their follow-up work (Chen et al., 2020b), they develop a smart contract online analysis framework named SODA and they also develop eight detection apps based on the SODA to detect the major vulnerabilities in smart contracts.

Some research works focus on specific security issues in the blockchain. Nikolić et al. (2018) discover three kinds of vulnerabilities in smart contracts by transaction analysis from Ethereum. Bartoletti et al. (2020) explained the definition of the Ponzi scheme and the Ponzi contract on Ethereum, described how the Ponzi scheme deceived users, and finally classified the existing Ponzi scheme. Grossman et al. (2017) detected the reentrancy vulnerability in smart contracts through internal transactions and storage operations. Sun, Ruan, and Liu (2019) apply machine learning to Ethereum analysis and group users and smart contracts by using transaction information in existing blocks and proposes a new method for user identification and malicious user detection based on the clustering results. Chen et al. (2018b) use transaction features and code opcode features with machine learning to detect Ponzi schemes on Ethereum. Cheng et al. (2019) deploy the honeypot for six months and they observed an interesting type of transaction called zero gas transaction, which has been leveraged by attackers to steal ERC20 tokens. Rouhani and Deters (2017) study Ethereum transactions and analyze two of the most popular Ethereum clients, Geth and Parity, on a private blockchain to better understand the impact of different clients on Ethereum performance.

Some researchers have developed specific tools to detect or mitigate security issues on Ethereum. Kalra et al. (2018) present a framework called ZEUS to verify the correctness and validate the fairness of smart contracts. Krupp and Rossow (2018) develop a generic definition of vulnerable contracts and use this to build a tool called TEETHER, which can help in finding, understanding, and preventing exploits before they cause losses. Liu et al. (2018) develop EClone, a semantic clone detector for Ethereum, to detect whether the contract is a clone contract. Because cloning contracts may amplify security threats or waste resources. Jiang, Liu, and Chan (2018) present a novel fuzzer called ContractFuzzer to test Ethereum smart contracts for security vulnerabilities. Ferreira Torres, Baden, Norvill, and Jonker (2019) introduce a tool that shields smart contracts and users on the blockchain from being exploited. Luu, Chu, Olickel, Saxena, and Hobor (2016) propose ways to enhance the operational semantics of Ethereum to make contracts less vulnerable and build a symbolic execution tool called Oyente to find potential security bugs. Ma et al. (2019) propose a method to reinforce the EVM when the smart contract contains vulnerabilities, and this method can stop dangerous transactions in real-time. Kolluri, Nikolic, Sergey, Hobor, and Saxena (2019) build an automatic tool called ETHRACER, which is effective at detecting a subtle yet dangerous class of bugs that existing tools miss. Torres, Schütte, and State (2018) focus on vulnerabilities related to integer bugs and introduce Osiris, a framework that combines symbolic execution and taints analysis, in order to accurately find integer bugs in Ethereum smart contracts. Rodler, Li, Karame, and Davi (2018) propose a novel smart contract security technology named Sereum, which protects existing deployed contracts against reentrancy attacks in a backward-compatible way based on runtime monitoring and validation.

5.4. Prediction and classification study

There are also some researchers find the behavior pattern, model the features of smart contract or transaction by data mining, they use machine learning or other algorithms to make prediction or classification research. Singh and Hafid (2019) present a novel approach to estimate the time it would take for a mining node to accept and confirm a transaction to a block using machine learning. Norvill, Pontiveros, State, Awan, and Cullen (2017) propose a framework to group together similar contracts within the Ethereum network using only the publicly available compiled code of contracts. He, Wu, Wang, Guo, and Jiang (2020) study to characterize the code reuse practice in the Ethereum smart contract ecosystem and to cluster contracts based on the similarity of bytecodes. Chen, Narwal, and Schultz (2019a) use various classification methods to predict the price sign of Ethereum, and all methods achieved above 50% accuracy.

5.5. Summary

Our work has covered all four aspects of research on Ethereum mentioned above.

1. **Data Acquisition Study.** The premise of our work is to collect complete contract and transaction data, and thus we have combed through the research on how to obtain Ethereum data in Section 5.1. The transactions we need to collect include both external and internal transactions (as we mentioned in Section 2.3), so it is not possible to get the complete data directly from parsing the blocks. Many works use the API provided by Etherscan to download transaction data, but direct use this data collection scheme can only get the last 10,000 transactions. We found that it is possible to get all the transactions of a contract by modifying the block ranges. This method can flexibly obtain the transactions of a single contract. However, due to the limitation of the server, it is not possible to

download transactions for a large number of contracts quickly. Thus, we also use the Parity client and set it to tracing-on mode so that it can compute and store tracing data. The data we obtain in these two ways can be cross-validated to ensure that they are correct. Compared to other works, although it is not possible to obtain customized data, our method of obtaining transactions is simpler and more convenient, while also ensuring the integrity of the data.

2. **Empirical Study.** In Section 5.2, the empirical study on Ethereum is summarized. Our work has also contributed to empirical study. Based on our analysis of these contracts in Section 3.2.1–3.2.4, we find that their transaction behavior has independent characteristics among different types of contracts. These peculiar patterns of behavior are consistent with actual phenomena. For examples, in a game-type contract usually shows the behavior of their Balance cliff-like reduction. That is because there is a winner in the game, and part of the Balance sends to the winner as a bonus. In a gambling-type contract, the Ether flow in and flow out show a fixed ratio relationship. This is consistent with the fixed odds situation in gambling. In a Ponzi contract, the sooner the account participates in the contract, the more times they are rewarded, which is significantly higher than that of the account participants later. This is consistent with the phenomenon of Ponzi schemes, in which early participants benefit from the investments of late participants, while late participants are less likely to make money.
3. **Performance and Security Study.** Blockchain security and performance have been the focus of research, which we review in Section 5.3. We believe that the classification and detection approach in this paper is important from the aspects of enhancing security, performance, and management of smart contracts. The reasons are as follows: 1) Classifying the contracts is the first step of malicious contract detection. Different contracts (eg., the detection of game contracts and finance contracts) have different behavior characters, thus the corresponding detection focus is different. The classify scheme and features extracted in this paper may provide a reference to studies of malicious smart contracts detection. 2) The deployer of a contract may conceal their true purpose, for example, some Ponzi contracts are masked by investment plan. The classification and detection approach in this paper can help users to verify the probability of a contract belonging to a certain type, thus contribute to discovering contracts with ambiguous purposes. 3) Some contracts are flawed in their design and may consume too much GAS, thus causing financial loss to users, which simultaneously affects the performance of the blockchain. Our approach can help users to discover these contracts and thus avoid losses. 4) The detection of malicious smart contracts can mitigate the security problems in the blockchain. Our detection approach can be extended to anomaly detection, malicious contract detection, and unknown smart contract identification, which can contribute to alleviating the security issues caused by smart contracts.
4. **Prediction and Classification Study.** Researches in this area are relatively insufficient, and the existing research is mainly focused on how to predict the price trends of cryptocurrencies in the blockchain. In Section 5.4, we review some related researches. Our work also contributes to this area. We extracted 14 features from the contract transactions, which were subsequently used to train the model through the LSTM network. Then, the trained model is used to classify smart contracts according to their scope of application, as well as to detect anomalous behavior of contracts and detect malicious contracts. In addition, classifying and examining the most common application cases can help those new smart contract developers in specific fields to understand which areas are worth more investment.

6. Conclusion

To address the security issues of smart contracts and to yield some useful insights into blockchain technology, we propose the transaction-based classification and detection approach for Ethereum smart contracts. We find out four patterns that can help us to have a better understanding of the contracts' transaction behavior, and they could be used to distinguish the difference between different types of contracts. Then 14 basic features of a smart contract are constructed. After that, we propose a data slicing algorithm to construct the experimental dataset. And we use an LSTM network to train and test our smart contract datasets. We demonstrate its usefulness through three experiments, including the evaluation of differences between different types of contracts, anomaly detection, and malicious contract identification.

Declaration of Competing Interest

The authors declare no conflicts of interest in this work.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. U19A2066, No. 61902262) and the Institute of Computer Application, China Academy of Engineering Physics (SJ2019A05).

References

Bai, Q., Zhang, C., Xu, Y., Chen, X., & Wang, X. (2020). Evolution of ethereum: A temporal graph perspective. arXiv preprint arXiv:2001.05251.
Baniata, H., Anaqreh, A., & Kertesz, A. (2021). PF-BTS: A privacy-aware fog-enhanced blockchain-assisted task scheduling. *Information Processing & Management, 58* (1), 102393.
Bartoletti, M., Carta, S., Cimoli, T., & Saia, R. (2020). Dissecting Ponzi schemes on ethereum: Identification, analysis, and impact. *Future Generation Computer Systems, 102*, 259–277.
Bartoletti, M., Lande, S., Pompianu, L., & Bracciali, A. (2017). A general framework for blockchain analytics. *Proceedings of the 1st workshop on scalable and resilient infrastructures for distributed ledgers* (pp. 1–6).

Berdik, D., Otoum, S., Schmidt, N., Porter, D., & Jararweh, Y. (2012). A survey on blockchain for information systems management and security. *Information Processing & Management, 58*(1), 102397.

Buterin, V., et al. (2014). A next-generation smart contract and decentralized application platform. *White Paper, 3*(37).

Chen, M., Narwal, N., & Schultz, M. (2019a). Predicting price changes in ethereum. *International Journal on Computer Science and Engineering (IJCSE) ISSN*, 0975–3397.

Chen, Q., Srivastava, G., Parizi, R. M., Aloqaily, M., & Al Ridhawi, I. (2020a). An incentive-aware blockchain-based solution for internet of fake media things. *Information Processing & Management, 57*(6), 102370.

Chen, T., Cao, R., Li, T., Luo, X., Gu, G., Zhang, Y., , … He, Z., et al. (2020b). SODA: A generic online detection framework for smart contracts. *Proceedings of the 27th network and distributed system security symposium*.

Chen, T., Li, X., Luo, X., & Zhang, X. (2017a). Under-optimized smart contracts devour your money. *2017 IEEE 24th international conference on software analysis, evolution and reengineering (saner)* (pp. 442–446). IEEE.

Chen, T., Li, X., Wang, Y., Chen, J., Li, Z., Luo, X., … Zhang, X. (2017b). An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks. *International conference on information security practice and experience* (pp. 3–24). Springer.

Chen, T., Li, Z., Zhang, Y., Luo, X., Chen, A., Yang, K., , … Hu, T., et al. (2019b). DataEther: Data exploration framework for ethereum. *2019 IEEE 39th international conference on distributed computing systems (ICDCS)* (pp. 1369–1380). IEEE.

Chen, T., Zhang, Y., Li, Z., Luo, X., Wang, T., Cao, R., … Zhang, X. (2019c). TokenScope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum. *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security* (pp. 1503–1520).

Chen, T., Zhu, Y., Li, Z., Chen, J., Li, X., Luo, X., … Zhange, X. (2018a). Understanding ethereum via graph analysis. *IEEE INFOCOM 2018-IEEE conference on computer communications* (pp. 1484–1492). IEEE.

Chen, W., Zheng, Z., Cui, J., Ngai, E., Zheng, P., & Zhou, Y. (2018b). Detecting Ponzi schemes on ethereum: Towards healthier blockchain technology. *Proceedings of the 2018 world wide web conference* (pp. 1409–1418).

Chen, W., Zheng, Z., Ngai, E. C.-H., Zheng, P., & Zhou, Y. (2019d). Exploiting blockchain data to detect smart Ponzi schemes on ethereum. *IEEE Access, 7*, 37575–37586.

Cheng, Z., Hou, X., Li, R., Zhou, Y., Luo, X., Li, J., & Ren, K. (2019). Towards a first step to understand the cryptocurrency stealing attack on ethereum. *22nd International symposium on research in attacks, intrusions and defenses ({RAID} 2019)* (pp. 47–60).

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Dappradar, (2019). Available: https://www.dappradar.com/.

Dapp, (2020). Available: https://www.dapp.com/.

Dappreview (2020). Available: https://www.dapp.review/.

Desjardins, J. (2016). Its official: Bitcoin was the top performing currency of 2015. Retrieved from The Money Project Website: http://money. visualcapitalist. com/ its-official-bitcoin-was-the-topperforming-currency-of-2015,.

Fenu, G., Marchesi, L., Marchesi, M., & Tonelli, R. (2018). The ICO phenomenon and its relationships with ethereum smart contract environment. *2018 International workshop on blockchain oriented software engineering (IWBOSE)* (pp. 26–32). IEEE.

Ferreira Torres, C., Baden, M., Norvill, R., & Jonker, H. (2019). Ægis: Smart shielding of smart contracts. *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security* (pp. 2589–2591).

Ferretti, S., & D'Angelo, G. (2020). On the ethereum blockchain structure: A complex networks theory perspective. *Concurrency and Computation: Practice and Experience, 32*(12), e5493.

Ghosh, A., Gupta, S., Dua, A., & Kumar, N. (2020). Security of cryptocurrencies in blockchain technology: State-of-art, challenges and future prospects. *Journal of Network and Computer Applications*, 102635.

Grossman, S., Abraham, I., Golan-Gueta, G., Michalevsky, Y., Rinetzky, N., Sagiv, M., & Zohar, Y. (2017). Online detection of effectively callback free objects with applications to smart contracts. *Proceedings of the ACM on Programming Languages, 2*(POPL), 1–28.

He, N., Wu, L., Wang, H., Guo, Y., & Jiang, X. (2020). Characterizing code clones in the ethereum smart contract ecosystem. *International conference on financial cryptography and data security* (pp. 654–675). Springer.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.

Jiang, B., Liu, Y., & Chan, W. (2018). ContractFuzzer: Fuzzing smart contracts for vulnerability detection. *2018 33rd IEEE/ACM International conference on automated software engineering (ASE)* (pp. 259–269). IEEE.

Kalra, S., Goel, S., Dhawan, M., & Sharma, S. (2018). Zeus: Analyzing safety of smart contracts.. *Ndss*.

Kiffer, L., Levin, D., & Mislove, A. (2017). Stick a fork in it: Analyzing the ethereum network partition. *Proceedings of the 16th ACM workshop on hot topics in networks* (pp. 94–100).

Kolluri, A., Nikolic, I., Sergey, I., Hobor, A., & Saxena, P. (2019). Exploiting the laws of order in smart contracts. *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis* (pp. 363–373).

Krupp, J., & Rossow, C. (2018). teEther: Gnawing at ethereum to automatically exploit smart contracts. *27th {USENIX} security symposium ({USENIX} security 18)* (pp. 1317–1333).

Li, J., Wu, J., Jiang, G., & Srikanthan, T. (2020a). Blockchain-based public auditing for big data in cloud storage. *Information Processing & Management, 57*(6), 102382.

Li, X., Jiang, P., Chen, T., Luo, X., & Wen, Q. (2020b). A survey on the security of blockchain systems. *Future Generation Computer Systems, 107*, 841–853.

Liang, J., Li, L., & Zeng, D. (2018). Evolutionary dynamics of cryptocurrency transaction networks: An empirical study. *PloS one, 13*(8), e0202202.

Liu, H., Yang, Z., Liu, C., Jiang, Y., Zhao, W., & Sun, J. (2018). EClone: Detect semantic clones in ethereum via symbolic transaction sketch. *Proceedings of the 2018 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering* (pp. 900–903).

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (pp. 254–269).

Ma, F., Fu, Y., Ren, M., Wang, M., Jiang, Y., Zhang, K., … Shi, X. (2019). EVM*: From offline detection to online reinforcement for ethereum virtual machine. *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)* (pp. 554–558). IEEE.

Nakamoto, S. (2019). Bitcoin: A peer-to-peer electronic cash system. *Technical Report*. Manubot.

Nikolić, I., Kolluri, A., Sergey, I., Saxena, P., & Hobor, A. (2018). Finding the greedy, prodigal, and suicidal contracts at scale. *Proceedings of the 34th annual computer security applications conference* (pp. 653–663).

Norvill, R., Pontiveros, B. B. F., State, R., Awan, I., & Cullen, A. (2017). Automated labeling of unknown contracts in ethereum. *2017 26th International conference on computer communication and networks (ICCCN)* (pp. 1–6). IEEE.

Oham, C., Michelin, R. A., Jurdak, R., Kanhere, S. S., & Jha, S. (2020). B-FERL: Blockchain based framework for securing smart vehicles. *Information Processing & Management, 58*(1), 102426.

Pierro, G. A., & Rocha, H. (2019). The influence factors on ethereum transaction fees. *2019 IEEE/ACM 2nd international workshop on emerging trends in software engineering for blockchain (WETSEB)* (pp. 24–31). IEEE.

Rodler, M., Li, W., Karame, G. O., & Davi, L. (2018). Sereum: Protecting existing smart contracts against re-entrancy attacks. arXiv preprint arXiv:1812.05934.

Rouhani, S., & Deters, R. (2017). Performance analysis of ethereum transactions in private blockchain. *2017 8th IEEE international conference on software engineering and service science (ICSESS)* (pp. 70–74). IEEE.

Singh, H. J., & Hafid, A. S. (2019). Transaction confirmation time prediction in ethereum blockchain using machine learning. arXiv preprint arXiv:1911.11592.

Solidity, E. (2019). Solidity documentation. Available:,https://solidity.readthedocs.io/en/v0.5.11/index.html.

Somin, S., Gordon, G., & Altshuler, Y. (2018). Social signals in the ethereum trading network. arXiv preprint arXiv:1805.12097.

Sovbetov, Y. (2018). Factors influencing cryptocurrency prices: Evidence from bitcoin, ethereum, dash, litcoin, and monero. *Journal of Economics and Financial Analysis, 2*(2), 1–27.

Spain, M., Foley, S., & Gramoli, V. (2020). The impact of ethereum throughput and fees on transaction latency during ICOs. *International conference on blockchain economics, security and protocols (tokenomics 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

State of the dapps (2020). Available:,https://www.stateofthedapps.com/.

Sun, H., Ruan, N., & Liu, H. (2019). Ethereum analysis via node clustering. *International conference on network and system security* (pp. 114–129). Springer.

Szabo, N. (1997). The idea of smart contracts, nick Szabo's papers and concise tutorials (1997). URL http://www. fon. hum. uva. nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinte rschool2006/szabo. best. vwh. net/smart_ contracts_2. html,.

Team, E. (2017). Etherscan: The ethereum block explorer. Available:,https://etherscan.io/.

Torres, C. F., Schütte, J., & State, R. (2018). Osiris: Hunting for integer bugs in ethereum smart contracts. *Proceedings of the 34th annual computer security applications conference* (pp. 664–676).

Torres, C. F., Steichen, M., et al. (2019). The art of the scam: Demystifying honeypots in ethereum smart contracts. *28th {USENIX} security symposium ({USENIX} security 19)* (pp. 1591–1607).

Wood, G., et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper, 151*(2014), 1–32.

Zhao, Q., Chen, S., Liu, Z., Baker, T., & Zhang, Y. (2020). Blockchain-based privacy-preserving remote data integrity checking scheme for IoT information systems. *Information Processing & Management, 57*(6), 102355.

Zheng, W., Zheng, Z., Dai, H.-N., Chen, X., & Zheng, P. (2020). XBlock-EOS: Extracting and exploring blockchain data from EOSIO. arXiv preprint arXiv:2003.11967.

Zheng, Z., Xie, S., Dai, H.-N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services, 14*(4), 352–375.