International Journal of Multimedia Information Retrieval (2022) 11:539–551
https://doi.org/10.1007/s13735-022-00264-3

REGULAR PAPER

Who is gambling? Finding cryptocurrency gamblers using multi-modal retrieval methods

Zhengjie Huang[1] · Zhenguang Liu[1] · Jianhai Chen[1] · Qinming He[1] · Shuang Wu[2] · Lei Zhu[3] · Meng Wang[4]

Received: 18 June 2022 / Revised: 17 September 2022 / Accepted: 7 October 2022 / Published online: 15 November 2022

Abstract
With the popularity of cryptocurrencies and the remarkable development of blockchain technology, decentralized applications emerged as a revolutionary force for the Internet. Meanwhile, decentralized applications have also attracted intense attention from the online gambling community, with more and more decentralized gambling platforms created through the help of smart contracts. Compared with conventional gambling platforms, decentralized gambling has *transparent rules* and a *low participation threshold*, attracting a substantial number of gamblers. In order to discover gambling behaviors and identify the contracts and addresses involved in gambling, we propose a tool termed *ETHGamDet*. The tool is able to automatically detect the smart contracts and addresses involved in gambling by scrutinizing the smart contract code and address transaction records. Interestingly, we present a novel LightGBM model with memory components, which possesses the ability to learn from its own misclassifications. As a side contribution, we construct and release a large-scale gambling dataset at https://github.com/AwesomeHuang/Bitcoin-Gambling-Dataset to facilitate future research in this field. Empirically, *ETHGamDet* achieves a F1-score of 0.72 and 0.89 in *address classification* and *contract classification* respectively, and offers novel and interesting insights.

Keywords Gambling detection · Multi-modal · Blockchain · Smart contract · Ethereum

✉ Zhenguang Liu
liuzhenguang2008@gmail.com

Zhengjie Huang
zj.h@zju.edu.cn

Jianhai Chen
chenjh919@zju.edu.cn

Qinming He
hqm@zju.edu.cn

Shuang Wu
wushuang@outlook.sg

Lei Zhu
leizhu0608@gmail.com

Meng Wang
eric.mengwang@gmail.com

1 College of Computer Science and Technology, Zhejiang University, Hangzhou, China

2 Nanyang Technological University, Singapore, Singapore

3 School of Information Science and Engineering, Shandong Normal University, Jinan, China

4 School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China

1 Introduction

Blockchain is essentially a distributed ledger that is shared among the nodes in a peer-to-peer network. The nodes, known as *miners*, record all the transactions that occurred in the network following a consensus protocol. The duplicate ledgers stored in the worldwide nodes ensure that transactions are immutable once recorded, endowing blockchain with *tamper-proof* and *decentralization* nature [52]. Fundamentally, the key feature of blockchain is that it maintains a secure and immutable transaction ledger among the peers that do not trust one another.

A **smart contract** is simply a program stored on the *blockchain*, which will be executed when its predetermined conditions are satisfied [43]. Smart contracts encode predefined contract terms into runnable code. The execution of a smart contract cannot be terminated by any participant, so that the terms are strictly followed. Smart contracts make the automatic execution of contract terms possible, ensuring fairness for all participants. Remarkably, smart contracts have enabled a wide range of applications in various domains, including distributed exchanges, wallets, crowdfunding, and decentralized gambling [7,23,39,40,55].

Conventional gambling applications have long suffered from notorious issues such as opaque guessing processes, fictitious prize pools, and refusal to pay the winners. *Smart contracts for gambling*, in contrast, enforce the gambling funds transferred strictly following the predefined rules, making the entire process completely transparent. This resolves the key concerns in the traditional gambling. Consequently, the number of decentralized gambling has seen a skyrocketing growth in the past years.

Discovering *gambling contracts* and *addresses* among millions of entities in the blockchain network not only enables us to perceive the security risks, but also offers a grand picture of the whole ecosystem. However, automatically identifying gambling contracts and addresses is by no means easy, and to the best of our knowledge there is still no such framework available.

Upon scrutinizing and experimenting with the multi-modal and complex data in Ethereum, we observe that there are three key challenges to be addressed in the gambling detection task, as shown in Fig. 1. **(1)** The data, such as *transactions, accounts*, and *smart contracts*, in blockchain are heterogeneous and multi-modal, and it is still unclear how to formulate the data for gambling-related knowledge extraction. **(2)** Smart contracts are compiled to bytecode before their deployment on Ethereum. The bytecode, which is in the form of assembly language, is extremely difficult to understand and process. **(3)** The scales of transaction data of different addresses vary greatly, making it inherently challenging to construct unified and effective features for all addresses.

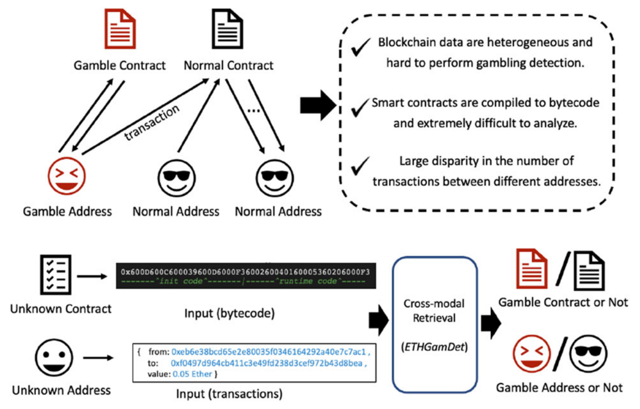
Gamble Contract
Normal Contract
transaction
...
Gamble Address
Normal Address
Normal Address
Blockchain data are heterogeneous and hard to perform gambling detection.
Smart contracts are compiled to bytecode and extremely difficult to analyze.
Large disparity in the number of transactions between different addresses.
Unknown Contract
0x600D600C600039600D6000F3600260040160005360206000F3
-------^init code^-------|------^runtime code^-----
Input (bytecode)
Unknown Address
{ from: 0xeb6e38bcd65e2e80035f0346164292a40e7c7ac1 ,
to: 0xf0497d964cb411c3e49fd238d3cef972b43d8bea ,
value: 0.05 Ether }
Input (transactions)
Cross-modal Retrieval
(ETHGamDet)
Gamble Contract or Not
Gamble Address or Not

Fig. 1 The key challenges of gambling behavior analysis. There are mainly two types of entities in the blockchain platform, addresses and contracts, which are connected by transactions. Our goal is to identify the entities involved in gambling

In this paper, we seek to formulate the gambling detection problem and present the first framework for it. We also contribute a large-scale multi-modal dataset for this task, hoping to inspire future researchers. We try to tackle the challenges in this problem by advocating three key designs: **(1) Contract Classification.** We utilize the smart contract bytecode to identify gambling-related smart contracts $\{\mathbf{S}_i\}_{i=1}^{k}$. Specifically, we disassemble the bytecode to obtain the contract operation code. We then devise principled features for the operation code and present a novel classifier, consisting of a machine learning model and a memory component, to identify gambling contracts. **(2) Address Classification.** We collect the transaction records of the gambling-related contracts $\{\mathbf{S}_i\}_{i=1}^{k}$. Sifting through these records, we obtain all the addresses $\{\mathbf{A}_i\}_{i=1}^{r}$ that have transactions with $\{\mathbf{S}_i\}_{i=1}^{k}$. To detect the addresses that are truly involved in gambling, we construct a transaction graph for each address $\mathbf{A}_i$. The transaction graph models the money transfer activities between $\mathbf{A}_i$ and other addresses, revealing the behavior pattern of $\mathbf{A}_i$. We then propose a cutting-edge approach for extracting features from the graphs and perform classification with our proposed classifier. **(3) Correction.** Finally, we engage in a feedback correction method for improving the overall accuracy of the system. Particularly, we leverage the results of the address classification to refine the classification of smart contracts.

This paper makes the following contributions:

- We propose a new problem of detecting gambling contracts and addresses from multi-modal data and present the first strong baseline for the problem. We have constructed and released a large-scale benchmark for evaluating gambling detection approaches, and publicized our code to engage the community.
- We provide a novel framework *ETHGamDet* to tackle the problem. Within the framework, we present a LightGBM model with memories to boost the classification accuracy, a feature extraction paradigm for smart contract bytecode, and a feature extraction paradigm for Ethereum addresses to realize the identification of addresses participating in gambling. One key highlight of *ETHGamDet* is the ability to directly handle smart contracts through bytecode (without reliance on access to source code), improving the utility of the system.
- *ETHGamDet* achieves a 0.72 and 0.89 F1-score on gambling contract and address classification, respectively. Through an empirical study, we also gain interesting insights and findings in this problem. We believe our work is an important step toward more intelligent and insightful gambling detection.

2 Multi-modal retrieval method

2.1 Problem formulation

Formally, presented with sets of smart contracts $\{S_i\}_{i=1}^{m}$ and account addresses $\{A_i\}_{i=1}^{n}$, we are interested in detecting all smart contracts and account addresses involved in gambling. We propose to jointly identify the smart contracts and addresses by scrutinizing two modalities, namely (1) smart contract code and (2) the transaction records of the account addresses.

2.2 Method overview

The proposed method *ETHGamDet* (Ethereum Gamble Detector) is outlined in Fig. 2, consisting of three key components: (1) smart contract classification, (2) account address classification, (3) feedback correction for reducing the false positive rate of smart contract classification. Next, we will introduce the three components, respectively.

2.3 Smart contract classification

Presented with millions of smart contracts, we must tackle the following challenges to detect gambling ones. (1) Within the contracts, only a small fraction of them are involved in gambling. (2) Most contracts are not open sourced, which means we have only their bytecode in hand.

Naively, we may directly train a classification model that takes bytecode as input and the label as output. However, the bytecode is a binary code sequence, and it is extremely difficult for a classification model to directly reason and understand the underlying semantics from the bytecode, leading to low accuracy. Another solution is to decompile the bytecode into source code, and adopt off-the-shelf approaches [4,22,42,53] for contract classification. However, it is well known that reverting the bytecode back to the source code is extremely difficult and will inevitably introduce wrong translations [30].

Therefore, we propose to disassemble the bytecode into EVM opcodes and cast EVM opcodes of different lengths into unified features. As demonstrated in Fig. 2, our smart contract classification component consists of three modules: EVM Disassembler, Opcode Feature Extractor and Contract Classifier.

2.3.1 EVM disassembler

Since most smart contracts exist in the form of bytecode, we implement a EVM disassembler to extract EVM opcodes from them.

Specifically, bytecode is an executable program, consisting of a sequence of $\langle opcode,\ data\rangle$ pairs. The Ethereum Virtual Machine (EVM) is the runtime environment for bytecode on Ethereum. In order to analyze the execution logic of the smart contract, we need to convert bytecode to assembly language using the opcode rules of EVM.

The EVM opcode is divided into specific instruction sets such as arithmetic operations, logical and comparison operations, control flow, system calls, stack operations, and memory operations [49]. In addition to typical bytecode operations, the EVM must manage account information (*e.g.,* addresses and balances), current gas prices, and block information [24]. Table 1 lists several common Ethereum opcodes. Each opcode corresponds to a specific assembly language operation.

2.3.2 Opcode feature extractor

After disassemble the bytecode of a smart contract, we obtain its corresponding opcode sequence. Since opcode sequence is unstructured data and different opcode sequences of different contracts have distinct lengths, we need to further encode the opcode sequence into a fixed-length feature vector.

We investigated all the 136 opcode operations in EVM one by one and count the number of occurrences of each opcode in each smart contract. In the experiments, we obtain two empirical insights. (1) Different from other smart contracts, contracts for gambling frequently involve *random number generation* operations and *gambling fund collection* operations. (2) We further observe that some opcodes appear rarely in contracts (e.g., PUSH5-PUSH32, DUP5-DUP16, SWAP5-SWAP16, etc.). Based on these observations, we discard the opcode operations that are both rarely used and unrelated to gambling. Finally, we settle down to 80 opcode operations and utilize the number of occurrence of each opcode operation in the contract as the features.

2.3.3 Contract classifier

Following the EVM Disassembler and Opcode Feature Extractor, we map each smart contract to a feature vector. We then engage a classification model to distinguish feature vectors for gambling contracts from that of other contracts. After scrutinizing the existing methods, we select LightGBM [27] as our contract classifier model. We would like to highlight that in order to make the model more intelligent, we innovatively augment LightGBM with an additional memory component.

Specifically, LightGBM uses the negative gradient of the loss function as the residual approximation of the current decision tree and then uses the residual approximation to fit the new decision tree [12,19].

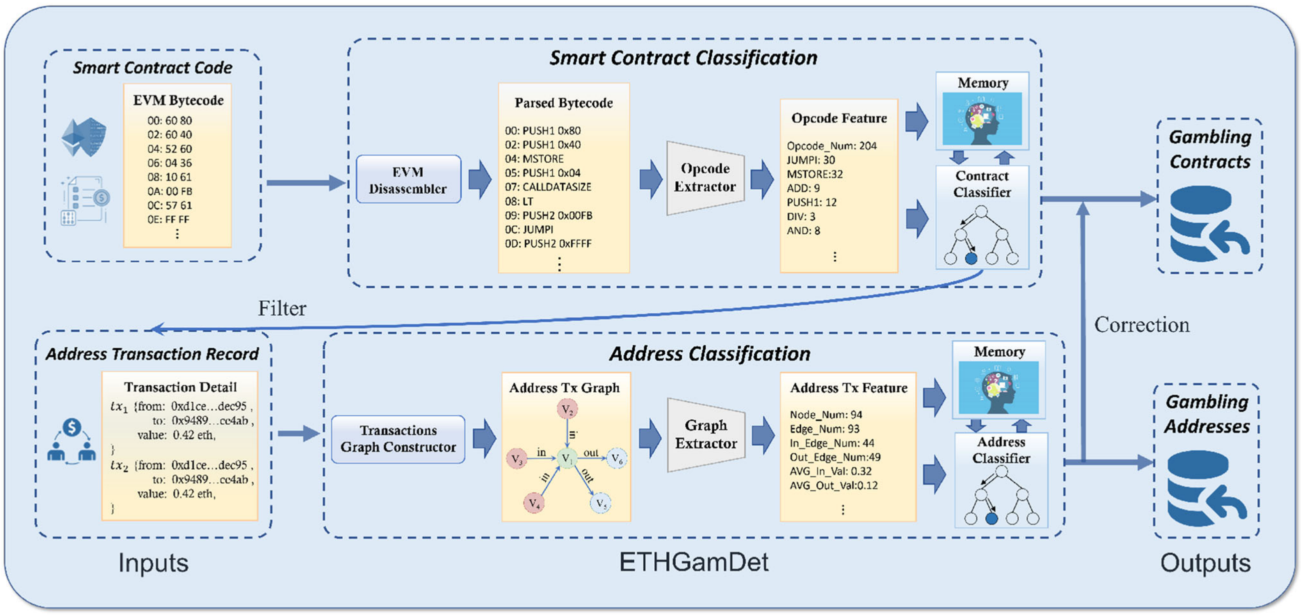

Fig. 2 The overview of the *ETHGamDet* system. There are three layers in this systems: **a** The inputs layer is used to provide input data consisting of Smart Contract Codes and Address Transaction Records to *ETHGamDet*. **b** The retrieval layer is where *ETHGamDet* processes the input data and outputs the result. This layer uses a multi-modal method to process contract codes and transaction records. This layer is mainly divided into two parts, *Smart Contract Classification* and *Address Classification*. After completing the data processing of these two parts, the system will also use the address classification results to refine the contract classification results, for reducing false positives and improving the accuracy of the system. **c** The outputs layer receives the classification results of *ETHGamDet*. After being processed by the ETHGamDet, we can get the contract data and address data involved in gambling

Table 1 EVM opcode examples and their corresponding operations

Opcode	Assembly	Description
0x00	stop	End command
0x01	add	Pop two values, add them, and push the result to the stack
0x02	mul	Pop two values, multiply them, and push the result to the stack
0x03	sub	Pop two values, subtract the two and push the result to the stack
0x10	lt	If the first popped value is less than the second, push 1, otherwise push 0
0x11	gt	If the first popped value is greater than the second, push 1, otherwise push 0
0x14	eq	Pop two values, if the two values are equal, push 1, otherwise push 0
0x15	iszero	Pop one value, if the value is 0, push 1 to the stack, otherwise push 0
0x34	callvalue	Get the transfer amount in the transaction
0x35	calldataload	Get the value of the input field in the transaction
0x36	calldatasize	Get the length of the input field in the transaction
0x50	pop	Pop the top value from the stack
0x52	mstore	Pop two values arg0 and arg1 in turn, and store arg1 at arg0 in memory
0x54	sload	Pop the value as the storage index, and load the value to the stack

Mathematically, The LightGBM can be viewed as an additive model consisting of K trees:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i),\ f_k \in F \tag{1}$$

The model is optimized using the *Boosting* algorithm. Specifically, it starts with a constant prediction and learns a new function each time, which is given by:

$$\hat{y}_i^0 = 0 \tag{2}$$

$$\hat{y}_i^1 = f_1(x_i) = \hat{y}_i^0 + f_1(x_i) \tag{3}$$

$$\hat{y}_i^2 = f_1(x_i) + f_2(x_i) = \hat{y}_i^1 + f_2(x_i) \tag{4}$$

$$\ldots$$

$$\hat{y}_i^t = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{t-1} + f_t(x_i) \tag{5}$$

At step t, the objective function can be formulated as:

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^t) + \sum_{i=i}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned} \tag{6}$$

The *residual* at step $t-1$ is defined as:

$$res = \hat{y}_i^{t-1} - y_i, \tag{7}$$

For each step of LightGBM, we only need to fit the residual of the previous step when generating the decision tree.

Empirically, we found that LightGBM is nearly 10 times faster than XGBoost, requires only one-sixth of the memory used by XGBoost, and more notably, has improved accuracy. LightGBM can speed up the training of the GBDT model without compromising the accuracy due to the following improvements made [27]: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

Gradient-based one-side sampling (GOSS) is an algorithm that balances reducing the amount of training data and ensuring accuracy. GOSS achieves the purpose of improving efficiency by distinguishing instances with different gradients, retaining instances with larger gradients and randomly sampling instances with smaller gradients.

Exclusive feature bundling (EFB) reduces feature dimension by feature bundling. EFB has designed an indicator termed conflict rate, which is used to measure the degree of mutually exclusiveness of features. When this metric is small, we can bundle the two features without affecting the final accuracy since they are not completely mutual exclusive.

Memory component Conventionally, LightGBM is composed of purely a classification model with a set of trained parameters. Inspired by the structure of human brain, our LightGBM system is designed to possess two components, *a classification model* and *a memory*. The memory remembers the misclassified samples. We periodically replay the misclassified samples to update the model parameters so that the classification model constantly learns from its own mistakes. This is consistent with our brain mechanism that occasionally replays important events registered in our memory. In particular, our LightGBM system is formulated in Algorithm 1. Briefly, the process can be sketched as:

(1) In the first-round training, we train the classification model of our LightGBM using all the training samples.
(2) We pick all the wrongly classified samples of the trained classification model and their corresponding labels, putting them into the memory component.
(3) In the second-round training, we re-train the classification model using all the training samples. However, after every k epochs, we add an additional epoch for replaying wrongly classified samples in the memory. More specifically, we (a) pick all the samples in the memory and (b) randomly select an equal number of samples from the training set to form the *training subset* for the replay epoch.
(4) Iterate (2) and (3) till converge.

Through replaying misclassified data, our LightGBM is endowed with the ability to intelligently learn and remember its own mistakes, ultimately improving the classification accuracy.

2.4 Address classification

Up to this point, we are able to classify smart contracts into *gambling contracts* and *others*. For the addresses that have transactions with the contracts classified as gambling ones, we further analyze their behaviors to decide whether they are indeed gamblers.

Toward this aim, we first construct a transaction graph for each address. Then, we perform graph feature extraction from the transaction graph and classify the features using a classification model. On the whole, our proposed address classification consists of three modules: Transaction Graph Constructor, Graph Feature Extractor and Address Classifier.

2.4.1 Transaction graph constructor

Unlike traditional transactions, transactions on Ethereum are not that complicated. We extract the payment address, recip-

Algorithm 1 LightGBM model with memory component

```
function train():
Input: gambling train dataset : train_data
Output: predict model : model
1: let memory = [ ]
2: let model = LightGBM.initialize()
3: while pred_loss < loss_threshold do
4:     input = train_data + (each $k$ epochs) memory
5:     model, pred_loss, err_data = model(input)
6:     memory = memory + err_data
7: end while
8: return model
function predict():
Input: gambling test dataset : test_data
Input: predict model : model
Output: Predict Results : results
1: let results = [ ]
2: for data in test_data do
3:     pred = model.predict(data)
4:     results.append(pred)
5: end for
6: return results
```

ient address, and transaction amount as the features of the transaction. For each address A_i, we model all the addresses that have transactions with A_i as the nodes and cast the transactions as edges. Particularly, if A_i is the payment address, an output edge is constructed. Conversely, if A_i is the recipient address, an input edge is added. The weight of the edge is set as the transaction amount.

Constructing such a transaction graph for each address have the following benefits. (1) Different from other addresses, gambling addresses often have multiple transactions target to the same recipient address, i.e., repeatedly adding funds to the identical gambling pool. This bizarre behavior is often not easily captured by conventional feature vectors but can be easily revealed by this graph structure. (2) The activeness of different addresses varies much, leading to a large variation in the number of transactions for each address. This graph enables us to devise fix-length features for all addresses.

Figure 3 shows an exemplar transaction graph. For the address located in the center of the figure, its transaction graph contains a total of seven edges. Two input edges represent two money incoming transactions, and five output edges stand for five money outgoing transactions. In total, the address has transactions with 5 other addresses. It is worth mentioning that there might be multiple transactions between two addresses and each transaction is modeled as an edge.

In order to construct and process large-scale graph data, we choose Neo4j as the graph database. Neo4j is a high-performance NoSQL database capable of storing structured data on the graph network [36,48].

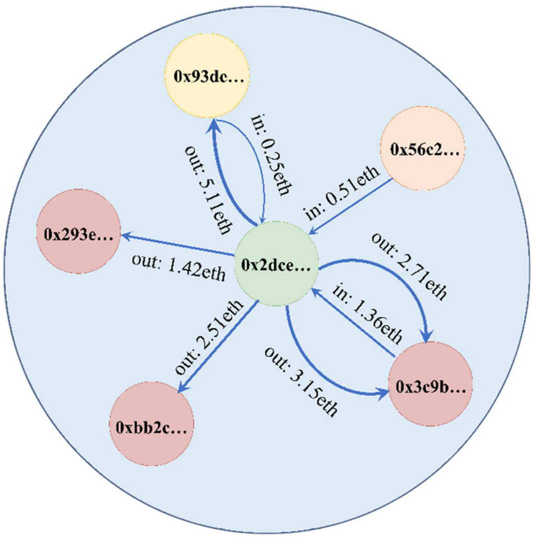

Fig. 3 An example of an address transaction graph. Each address has a transaction graph, which records all addresses that have transactions with that address. Nodes in the graph represent addresses, the edges between nodes represent transactions, and the direction of the edges represents the flow of the money

2.4.2 Graph feature extractor

To extract features from the transaction graph, we engage in three kinds of metrics, namely *basic metrics*, *degree metrics*, and *amount metrics*, which are demonstrated in Table 2.

Basic metrics mainly focus on the overall structure of the graph, which capture the number of nodes, edges, and edge types in the transaction graph.

Degree metrics concern the nodes of the graph. The degree metrics mainly model the average degree, average in-degree, and average out-degree of nodes.

Amount metrics concentrate on the edges of the graph. Amount metrics mainly include the sum of transaction amounts, average transaction amount, and transaction amount variance for the edges.

2.4.3 Address classifier

After obtaining the transaction graph features for each address, we use a machine learning model to classify the addresses. For the machine learning model, we adopt the LightGBM model with memory component introduced before in Subsect. 2.3. Put differently, we reuse the structure of the smart contract classifier but train it with address data.

2.5 Feedback correction

Due to the huge number of contracts on Ethereum, gambling contracts only account for a tiny portion of them. Therefore, even a small false positive rate of the smart contract classifier

Table 2 Metrics used for graph feature extraction

Metrics type	Metric name	Description
Basic metrics	vertex_number	The total number of vertices in the entire graph
	edge_number	The total number of edges in the entire graph
	in_edge_number	The total number of input edges in the entire graph
	out_edge_number	The total number of output edges in the entire graph
Degree Metrics	vertex_degree	The average degree of the nodes in the graph
	vertex_in_degree	The average in-degree of the nodes in the graph
	vertex_out_degree	The average out-degree of the nodes in the graph
Amount Metrics	total_amount	The sum of the amount of all transactions
	total_in_amount	The sum of the amount of all input transactions
	total_out_amount	The sum of the amount of all output transactions
	avg_amount	Average amount of all transactions
	avg_in_amount	Average amount of all input transactions
	avg_out_amount	Average amount of all output transactions
	amount_variance	Amount variance for all transactions
	in_amount_variance	Amount variance for all input transactions
	out_amount_variance	Amount variance for all output transactions

would lead to a large number of non-gambling addresses classified as gambling ones. Motivated by this, we design a feedback correction mechanism to further reduce the false positive rate of the smart contract classifier.

Empirically, if most of the addresses associated with a contract are not gambling addresses, then the contract is probably not a gambling contract. After horizontal comparison in experiments, we finally select the threshold 80%. If less than 80% of the addresses associated with the contract are gambling addresses, then the contract is likely to be a false positive sample. We revise its label to non-gambling. After adopting this method, the false positive rate of the smart contract classifier is greatly reduced.

3 Experiments

In this section, we evaluate the performance of the model. We seek to answer the following research questions:

- **RQ1:** How to build the benchmark dataset for evaluating gambling classification models?
- **RQ2:** Is the proposed method effective in identifying gambling smart contracts and addresses? How does it compare to other classification models.
- **RQ3:** How is the importance of different features? Can we gain new insights from the empirical results?

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

3.1 Experimental settings

3.1.1 Implementation details

All experiments are done on a server equipped with an Intel E5-2630 v4 2.20GHz CPU and 128 G of memory. The *ETHGamDet* tool was developed using Python as the programming language. For the transaction graph of addresses, we use Neo4j as the database to store and manage the data.

3.1.2 Evaluation metrics

ETHGamDet mainly classifies whether a contract is providing gambling service or an address is a gambler, which concerns a binary classification problem. Therefore, we adopt metrics commonly used in binary classification tasks: accuracy, precision, recall, and F1-score.

3.2 Datasets (RQ1)

As Fig. 2 shows, we need to collect the smart contract code and address transaction records. We observe that the Ethereum browser website *etherscan.io* [44] provides relevant APIs to obtain contract data and transaction data, so we obtain data through the website's APIs.

It is worth noting that gambling is only one of the various applications on blockchain. Therefore, gambling contracts and gambling addresses only account for a small portion of contracts and addresses in Ethereum. To make the classification models aware of this phenomenon, we constructed an

Table 3 The quantitative information of dataset

Dataset	Gamble	Non-gamble	Unlabel	Total
Contract	260	1,040	2,585	3,885
Address	10,423	51,004	28,919	90,346

imbalanced dataset, where gambling contracts and addresses only accounts for around 20%.

For model evaluation and training, we constructed two datasets, gambling contract dataset and gambling address dataset [25], respectively. We manually labeled the addresses in our dataset. In total, four annotators participated in the data collection and labeling. We explored a large number of decentralized gambling sites and recorded the contracts published on the sites. By analyzing these contracts and associated transactions, we were able to obtain the contracts and addresses involved in gambling. Table 3 shows the statistics of the datasets. Interestingly, we would like to point out we also collected 2,585 unlabeled contracts and 28,919 unlabeled addresses in the datasets, leaving a room for exploring semi-supervised classification models on this open problem.

3.2.1 Gambling contract dataset

We collected a total of 260 gambling smart contracts from decentralized gambling websites, such as Dicether [2], Degens [1]. At the same time, in order to construct the negative samples required for training, we selected 1,040 smart contracts that are not involved in gambling (e.g., erc20 [46], erc721 [16], mixer [18], etc.). In the dataset, we use accounts to refer to contracts (e.g., 0x3fe2b...f8a33f), where 1, 0, and −1 to represent the gamble, non-gamble, and other types, respectively.

3.2.2 Gambling address dataset

We collected 10,423 gambling addresses that have transactions with gambling contracts. Moreover, we also selected 51,004 non-gambling addresses (such as exchanges [47], wallet addresses [17], etc.), making the gambling address dataset more complete. In the dataset, we use accounts to refer to addresses (e.g., 0xd1ce...edec95), where 1, 0, and −1 represent the gamble, non-gamble, and other types, respectively.

3.3 Classification results (RQ2)

In this section, we evaluate the proposed *ETHGamDet* by conducting comparative experiments. Our experiments start from two perspectives: address classification results and contract classification results. By combining these two types of classification results, we have an intuitive understanding of the effect of *ETHGamDet*.

We selected seven commonly used models, LR, SVM, KNN, Bernoulli NB, Gaussian NB, Decision Tree, and Random Forest, for comparison. These models cover the current mainstream machine learning methods. As shown in Table 4, experiments demonstrate that the model LightGBM with memory component achieves remarkable results in both classification tasks. In the experiment of identifying gambling addresses, the F1-score of *ETHGamDet* reached 0.72, while in the experiment of identifying gambling contracts, the F1-score of *ETHGamDet* reached 0.89. This suggests that *ETHGamDet* is able to accurately identify the addresses and contracts involved in gambling.

To allow for more intuitive understanding, we also visualized the comparison results in Fig. 4. The detailed discussions on the experimental results are given below.

3.3.1 Address classification results

We conducted comparative experiments on a total of 61,427 addresses. The experimental results show that our model consistently outperforms other classification models and achieves 0.90, 0.67, 0.77, 0.72 in accuracy, precision, recall, and F1-score, respectively. The F1-score of our model is 0.03 higher than the LightGBM without memory component, which ranks the second.

Interestingly, we observe that all methods tend to have a relatively low precision in the experiments. We speculate the reasons are two-folds. On one hand, by analyzing the experimental data, we found that there are a large number of addresses having only a few transactions. The behavior of such addresses is extremely difficult to analyze due to the lack of data, affecting the precision of address classification. On the other hand, gambling transactions are similar to legal frequent transfer transactions, which makes it difficult for the models to distinguish them.

3.3.2 Contract classification results

We conducted comparative experiments on a total of 1,300 labeled contracts. We first compared across different classification models. It can be seen from the experimental results in Table 4 that our LightGBM algorithm surpasses LR, SVM, Naive Bayes, Decision Tree and Random Forest in various indicators. The empirical evidences reveal that our model is effective in the gambling contract detection task.

Next, we studied the effect of our designed *Correction Feedback* on the performance. By default, we used the *Correction Feedback* component to reduce false positives in contract classification. To study its effect, we removed it from the method. After removing *Correction Feedback*, the accuracy, precision, and F1-score of the model all decreased.

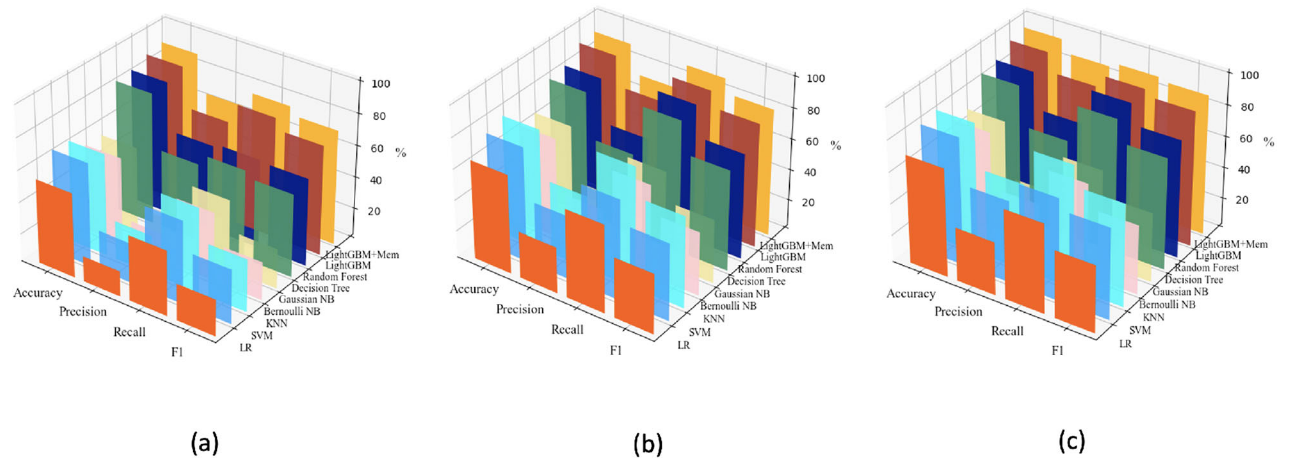

Fig. 4 Visually comparison: **a** shows the experimental comparison results of address classification on the gambling address dataset. **b** & **c** shows the experimental comparison results of contract classification on the gambling contract dataset, while (**b**) has no feedback correction and (**c**) has feedback correction. In each graph, the 9 rows from front to back denote the LR, SVM, KNN, Bernoulli NB, Gaussian NB, Decision Tree, Random Forest, LightGBM and LightGBM with memory methods, respectively. For each column in the figures, accuracy, recall, precision, and F1-score are, respectively, demonstrated from left to right

Table 4 *ETHGamDet* experimental results

Classification task	Model selection	Accuracy	Precision	Recall	F1-score
Address classification	LR	0.53	0.16	0.41	0.23
	SVM	0.65	0.25	0.52	0.34
	KNN	0.64	0.24	0.53	0.33
	Bernoulli NB	0.55	0.17	0.43	0.24
	Gaussian NB	0.55	0.17	0.44	0.25
	Decision tree	0.83	0.49	0.56	0.52
	Random forest	0.84	0.54	0.56	0.55
	LightGBM	0.88	0.63	0.76	0.69
	ETHGamDet	**0.90**	**0.67**	**0.77**	**0.72**
Contract Classification (without correction)	LR	0.64	0.30	0.57	0.39
	SVM	0.74	0.41	0.65	0.50
	KNN	0.78	0.47	0.78	0.59
	Bernoulli NB	0.65	0.31	0.59	0.41
	Gaussian NB	0.66	0.31	0.60	0.41
	Decision tree	0.81	0.52	0.85	0.65
	Random forest	0.83	0.55	0.88	0.68
	LightGBM	0.91	0.72	0.90	0.80
	ETHGamDet	**0.92**	**0.74**	**0.91**	**0.82**
Contract classification (with correction)	LR	0.69	0.34	0.57	0.43
	SVM	0.81	0.52	0.65	0.58
	KNN	0.83	0.55	0.78	0.65
	Bernoulli NB	0.70	0.35	0.59	0.44
	Gaussian NB	0.70	0.35	0.60	0.44
	Decision tree	0.86	0.61	0.85	0.71
	Random forest	0.88	0.65	0.88	0.75
	LightGBM	0.94	0.82	0.90	0.86
	ETHGamDet	**0.96**	**0.88**	**0.91**	**0.89**

Bold values indicate the experimental results of our method, which outperforms all other methods

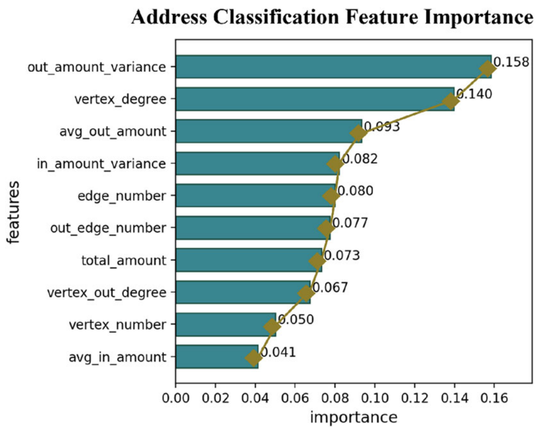

Fig. 5 This figure shows the feature importance of address classification. We show the top ten most important features. It can be seen that features *out_amount_variance* and *vertex_degree* are ranked the highest in importance

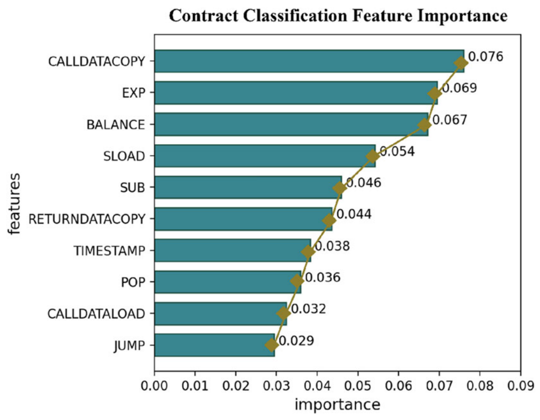

Fig. 6 This figure shows the feature importance of contract classification. We show the top ten most important features. It can be seen that features *CALLDATACOPY*, *EXP*, and *BALANCE* are ranked the highest in importance

More specifically, the precision of contract classification decreased from 0.96 to 0.92, and the F1-score of the classification decreased from 0.89 to 0.82. This reveals the effectiveness of the proposed *Correction Feedback* component. It is worth noting that since the correction feedback mechanism does not affect the false negative rate of the model, the recall does not change with or without correction feedback.

Finally, we horizontally compare the results of contract classification and address classification. From Table 4, we observe that the F1-score of contract classification is generally better than that of address classification. This is because the code in the contract is more semantically rich, and a substantial number of addresses have only a few transactions. This also shows the rationality of the design of *ETHGamDet*. *ETHGamDet* processes the contract code first and then handles the address transactions. Through this processing pipeline, the better classification results of the contract code can be utilized to the greatest extent.

3.4 Feature analysis (RQ3)

To understand the interpretability of *ETHGamDet*, we perform feature importance analysis in the address classification and the contract classification separately. We selected the contract classification and address classification models trained in Sect. 3.3 and used the built-in feature importance function of LightGBM to print out the weights of each input feature and draw them into a graph.

Figure 5 shows feature importance for address classification. From the figure, we have the following observations. (1) Feature *out_amount_variance* has the highest weight. The feature *out_amount_variance* represents the variance of the amount paid by the gambler to the gambling contract. Unlike behaviors such as transfers, gamblers usually pay a similar amount each time, so gambling addresses are similar in this feature. (2) Another feature with high importance is *vertex_degree*. This feature can reflect the frequency of gamblers' participation in a particular gamble. We found that gamblers often participate in a certain gambling contract, which is reflected in the graph that there are multiple edges between two nodes.

Figure 6 shows feature importance for contract classification. As can be seen from the figure, CALLDATACOPY, EXP and BALANCE have the highest weights. CALLDATACOPY copies multiple bytes of transaction data into memory. Gambling contracts need to complete bets and withdrawals based on transaction data, so CALLDATACOPY often appears in gambling contracts. Gambling contracts need to generate random numbers based on some data, so the exponentiation modulo operation EXP is often used to generate random numbers. The role of BALANCE is to query the balance in the contract account. The gambling contract needs to judge the balance to calculate the amount of the gambling lottery.

4 Related work

With the continuous development of blockchain technology, the application ecology based on blockchain is constantly changing the Internet. While these applications gradually expand the use boundaries of the blockchain, they also bring many problems, such as gambling [41], money laundering [11], darknet transactions [10], and hacker attacks [35]. In recent years, more and more researchers paid attention to these problems and strived to solve them using novel techniques.

Unlike traditional problems, all contract code and transaction records are deployed on the blockchain, which makes it easier for us to obtain all data for analysis. However, the data on the blockchain are multi-modal and lack connections to real-world entities.

At present, most blockchain research is based on two types of data, namely the code of smart contracts and the transaction records of addresses.

By analyzing the code of the smart contract, we may have a better understanding of the purpose or security issues of the contract [6,8,9,13,20,21,31,32,34,37,54]. Chen et al. [14] used machine learning methods to analyze the contract code, identifying more than 400 contracts that may be related to the Ponzi scheme. Moreover, analyzing the contract code can also help us find out the loopholes in the contract, so as to avoid being exploited by hackers. Luu et al. [33] designed a tool called Oyente. The tool can use the method of symbolic execution to find out four types of vulnerabilities: Transaction-ordering Dependence, Timestamp Dependence, Mishandled Exceptions, and Reentrancy Vulnerability. Kalra et al. [26] designed a tool called Zeus by borrowing ideas from the Oyente tool. The tool utilizes both abstract interpretation and symbolic model checking. A more advanced detection tool Securify was proposed by Tsankov et al. [45]. The tool first extracts semantic information accurately by analyzing the dependency graph of the contract, and then captures compliance and violation patterns to analyze the security of the contract.

By analyzing the transaction records of the address, we can realize the behavior pattern of the address. Wu et al. [50] proposed a method to detect phishing scams on Ethereum. This method designs a new network embedding algorithm *trans2vec* to perform data mining on transaction records. Through this method, Wu et al. realized the classification of addresses in Ethereum, so as to determine whether an address is a phishing one. Chen et al. [15] analyzed the leaked transaction records of the Mt.Gox exchange. They model transaction information as a graph, so as to dig out the abnormal transaction behaviors. Chen et al. found that there was serious market manipulation in the Mt.Gox exchange, illustrating that the regulation of cryptocurrency exchanges needs to be strengthened. In addition, address transaction records are also used for abnormal behavior monitoring [3,5,28,29,38,51], such as money flow detection and identity tracing.

Overall, analyzing the smart contract code enables us to comprehend the semantic characteristics of the contract, while analyzing the address transaction records allows us to understand the behavioral characteristics of the address. By analyzing both data sources simultaneously, we are able to capture the vast majority of events on the blockchain. As a result, we resort to a multi-modal retrieval method to process these two types of data to identify the contracts and accounts involved in gambling on Ethereum.

5 Conclusion and future work

In this paper, we propose *ETHGamDet*, a novel Ethereum gambling detection system. The tool employs a multi-modal approach to analyze real data on Ethereum. *ETHGamDet* attempts to address the difficulties of gambling detection from three perspectives: (1) *Problem* Divides the issue into two subtasks: classification of gambling contracts and classification of gambling addresses. (2) *Dataset* Constructs smart contract and address transaction datasets so that gambling detection algorithms can be evaluated. (2) *Method* Presents a unique framework for jointly detecting gambling contracts and gambling addresses. Within the framework, it develops paradigms for smart contract feature extraction and address feature extraction, and introduces a new LightGBM model enhanced with memories. Extensive experiments are conducted to assess the proposed method, which demonstrates its efficacy and yields novel findings.

Although *ETHGamDet* has shown exceptional success in identifying gambling-related activities, there are still limitations. In future work, we will enhance *ETHGamDet* in terms of performance and application breadth. (1) *Performance* On the one hand, we will improve the opcode extractor in the smart contract classification phase. We will attempt to conduct feature understanding on smart contracts using program analysis techniques such as control flow graphs, abstract syntax trees. On the other hand, at the phase of address classification, we will enhance the graph extractor. We will try to utilize a graph neural network to perform feature extraction on the constructed graph, in order to mine the features of the transaction graph at a deeper level. (2) *Scope of application* We believe that *ETHGamDet*, as the most advanced gambling identification tool, should not only assist the identification of gambling contracts and gambling addresses, but also distinguish between different types of contracts and addresses. Concerning the gambling contract, we will examine the code of the contract in further detail to evaluate if it is a “fair contract”. Regarding the gambling address, we will study the gambling behavior of the address to establish its gambling preference that whether it is risk-averse or greedy.

With *ETHGamDet*, it is possible to detect and investigate gambling behaviors on Ethereum. By doing further research on these gaming addresses and contracts, we were able to determine the design rules of the gambling game and the preferences of players. We believe our work is a significant advancement in the field of cryptocurrency gambling detection, and we hope it will could inspire the community.

Acknowledgements This work was supported in part by the National Key R&D Program of China (2021YFB2700500); in part by the National Natural Science Foundation of China (No. 61902348); and in part by the Key R&D Program of Zhejiang Province (No. 2021C01104).

Data Availability The datasets analyzed during the current study are available in the GitHub repository [25], https://github.com/AwesomeHuang/Bitcoin-Gambling-Dataset.

Declarations

Conflict of interest All the authors have checked the manuscript and have agreed to the submission in International Journal of Multimedia Information Retrieval. There is no conflict of interest.

References

1. (2022) Degens - the ethereum betting exchange. Website, https://degens.com/
2. (2022) Dicether. Website, https://dicether.com/
3. Akcora CG, Li Y, Gel YR, et al (2020) Bitcoinheist: topological data analysis for ransomware prediction on the bitcoin blockchain. In: Proceedings of the twenty-ninth international joint conference on artificial intelligence
4. Albert E, Gordillo P, Livshits B, et al (2018) Ethir: a framework for high-level analysis of ethereum bytecode. In: International symposium on automated technology for verification and analysis, Springer, pp 513–520
5. Ante L, Fiedler I, Strehle E (2021) The impact of transparent money flows: Effects of stablecoin transfers on the returns and trading volume of bitcoin. Technological Forecasting and Social Change 170(120):851
6. Atzei N, Bartoletti M, Cimoli T (2017) A survey of attacks on ethereum smart contracts (sok). In: International conference on principles of security and trust, Springer, pp 164–186
7. Ayed AB (2017) A conceptual secure blockchain-based electronic voting system. Int J Network Sec Appl 9(3):01–09
8. Bhargavan K, Delignat-Lavaud A, Fournet C, et al (2016) Formal verification of smart contracts: Short paper. In: Proceedings of the 2016 ACM workshop on programming languages and analysis for security, pp 91–96
9. Brent L, Jurisevic A, Kong M, et al (2018) Vandal: a scalable security analysis framework for smart contracts. arXiv preprint arXiv:1809.03981
10. Broadhurst R, Lord D, Maxim D, et al (2018) Malware trends on 'darknet' crypto-markets: research review. Available at SSRN 3226758
11. Campbell-Verduyn M (2018) Bitcoin, crypto-coins, and global anti-money laundering governance. Crime, Law and Social Change 69(2):283–305
12. Chen T, He T, Benesty M et al (2015) Xgboost: extreme gradient boosting. R package version 04-2 1(4):1–4
13. Chen T, Li X, Luo X, et al (2017) Under-optimized smart contracts devour your money. In: 2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER), IEEE, pp 442–446
14. Chen W, Zheng Z, Cui J, et al (2018) Detecting ponzi schemes on ethereum: towards healthier blockchain technology. In: Proceedings of the 2018 world wide web conference, pp 1409–1418
15. Chen W, Wu J, Zheng Z, et al (2019) Market manipulation of bitcoin: Evidence from mining the mt. gox transaction network. In: IEEE INFOCOM 2019-IEEE conference on computer communications, IEEE, pp 964–972
16. Chirtoaca D, Ellul J, Azzopardi G (2020) A framework for creating deployable smart contracts for non-fungible tokens on the ethereum blockchain. In: 2020 IEEE international conference on decentralized applications and infrastructures (DAPPS), IEEE, pp 100–105
17. Er-Rajy L, El Kiram My A, El Ghazouani M et al (2017) Blockchain: Bitcoin wallet cryptography security, challenges and countermeasures. Journal of Internet Banking and Commerce 22(3):1–29
18. Feng Q, He D, Zeadally S et al (2019) A survey on privacy protection in blockchain system. Journal of Network and Computer Applications 126:45–58
19. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat pp 1189–1232
20. Fu Y, Ren M, Ma F, et al (2019) Evmfuzzer: detect evm vulnerabilities via fuzz testing. In: Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, pp 1110–1114
21. Grech N, Kong M, Jurisevic A et al (2018) Madmax: surviving out-of-gas conditions in ethereum smart contracts. In: Proceedings of the ACM on programming languages 2(OOPSLA):1–27
22. Grech N, Brent L, Scholz B, et al (2019) Gigahorse: thorough, declarative decompilation of smart contracts. In: 2019 IEEE/ACM 41st international conference on software engineering (ICSE), IEEE, pp 1176–1186
23. Guo Y, Liang C (2016) Blockchain application and outlook in the banking industry. Financial innovation 2(1):1–12
24. Hildenbrandt E, Saxena M, Rodrigues N, et al (2018) Kevm: a complete formal semantics of the ethereum virtual machine. In: 2018 IEEE 31st computer security foundations symposium (CSF), IEEE, pp 204–217
25. Huang Z (2022) Bitcoin gambling dataset. Website, https://github.com/AwesomeHuang/Bitcoin-Gambling-Dataset
26. Kalra S, Goel S, Dhawan M, et al (2018) Zeus: analyzing safety of smart contracts. In: Ndss, pp 1–12
27. Ke G, Meng Q, Finley T, et al (2017) Lightgbm: a highly efficient gradient boosting decision tree. Adv Neural Inf Process Syst 30
28. Lee C, Maharjan S, Ko K, et al (2019) Toward detecting illegal transactions on bitcoin using machine-learning methods. In: International conference on blockchain and trustworthy systems, Springer, pp 520–533
29. Li P, Xu H, Ma T (2021) An efficient identity tracing scheme for blockchain-based systems. Information Sciences 561:130–140
30. Liu J, Liu Z (2019) A survey on security verification of blockchain smart contracts. IEEE Access 7:77894–77904
31. Liu Z, Qian P, Wang X, et al (2021) Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion. arXiv preprint arXiv:2106.09282
32. Liu Z, Qian P, Wang X, et al (2021) Combining graph neural networks with expert knowledge for smart contract vulnerability detection. IEEE Trans Knowl Data Eng
33. Luu L, Chu DH, Olickel H, et al (2016) Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp 254–269
34. Macrinici D, Cartofeanu C, Gao S (2018) Smart contract applications within blockchain technology: A systematic mapping study. Telematics and Informatics 35(8):2337–2354
35. Mehar MI, Shier CL, Giambattista A et al (2019) Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack. Journal of Cases on Information Technology (JCIT) 21(1):19–32
36. Miller JJ (2013) Graph database applications and concepts with neo4j. In: Proceedings of the southern association for information systems conference, Atlanta, GA, USA
37. Mohanta BK, Panda SS, Jena D (2018) An overview of smart contract and use cases in blockchain technology. In: 2018 9th international conference on computing, communication and networking technologies (ICCCNT), IEEE, pp 1–4
38. Morishima S (2021) Scalable anomaly detection in blockchain using graphics processing unit. Computers & Electrical Engineering 92(107):087

39. Norta A (2016) Designing a smart-contract application layer for transacting decentralized autonomous organizations. In: International conference on advances in computing and data sciences, Springer, pp 595–604
40. Qian P, Liu Z, Wang X, et al (2019) Digital resource rights confirmation and infringement tracking based on smart contracts. In: 2019 IEEE 6th international conference on cloud computing and intelligence systems (CCIS), IEEE, pp 62–67
41. Scholten OJ, Zendle D, Walker JA (2020) Inside the decentralised casino: A longitudinal study of actual cryptocurrency gambling transactions. PloS one 15(10):e0240,693
42. Suiche M (2017) Porosity: a decompiler for blockchain-based smart contracts bytecode. DEF con 25(11)
43. Szabo N, et al (1994) Smart contracts
44. Team E (2017) Etherscan: The ethereum block explorer. https://etherscan.io
45. Tsankov P, Dan A, Drachsler-Cohen D, et al (2018) Securify: practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp 67–82
46. Victor F, Lüders BK (2019) Measuring ethereum-based erc20 token networks. In: International conference on financial cryptography and data security, Springer, pp 113–129
47. Warren W, Bandeali A (2017) 0x: An open protocol for decentralized exchange on the ethereum blockchain. https://githubcom/0xProject/whitepaper, pp 04–18
48. Webber J (2012) A programmatic introduction to neo4j. In: Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, pp 217–218
49. Wood G (2014) Ethereum yellow paper. Internet: https://githubcom/ethereum/yellowpaper, [Oct 30, 2018] p 30
50. Wu J, Yuan Q, Lin D, et al (2020) Who are the phishers? phishing scam detection on ethereum via network embedding. IEEE Trans Syst Man Cybern Syst
51. Yan C, Zhang C, Lu Z et al (2022) Blockchain abnormal behavior awareness methods: a survey. Cybersecurity 5(1):1–27
52. Zheng Z, Xie S, Dai HN et al (2018) Blockchain challenges and opportunities: A survey. Int J Web Grid Serv 14(4):352–375
53. Zhou Y, Kumar D, Bakshi S, et al (2018) Erays: reverse engineering ethereum's opaque smart contracts. In: 27th USENIX security symposium (USENIX Security 18), pp 1371–1385
54. Zhuang Y, Liu Z, Qian P, et al (2020) Smart contract vulnerability detection using graph neural network. In: IJCAI, pp 3283–3290
55. Zichichi M, Contu M, Ferretti S, et al (2019) Likestarter: a smart-contract based social dao for crowdfunding. In: IEEE INFOCOM 2019-IEEE conference on computer communications workshops (INFOCOM WKSHPS), IEEE, pp 313–318