

Behavior-aware Account De-anonymization on Ethereum Interaction Graph

Jiajun Zhou, Chenkai Hu, Jianlei Chi, Jiajing Wu, *Senior Member, IEEE*,
Meng Shen, *Member, IEEE*, and Qi Xuan, *Senior Member, IEEE*

Abstract—Blockchain technology has the characteristics of decentralization, traceability and tamper-proof, which creates a reliable decentralized trust mechanism, further accelerating the development of blockchain finance. However, the anonymization of blockchain hinders market regulation, resulting in increasing illegal activities such as money laundering, gambling and phishing fraud on blockchain financial platforms. Thus, financial security has become a top priority in the blockchain ecosystem, calling for effective market regulation. In this paper, we consider identifying Ethereum accounts from a graph classification perspective, and propose an end-to-end graph neural network framework named *Ethident*, to characterize the behavior patterns of accounts and further achieve account de-anonymization. Specifically, we first construct an Account Interaction Graph (AIG) using raw Ethereum data. Then we design a hierarchical graph attention encoder named *HGATE* as the backbone of our framework, which can effectively characterize the node-level account features and subgraph-level behavior patterns. For alleviating account label scarcity, we further introduce contrastive self-supervision mechanism as regularization to jointly train our framework. Comprehensive experiments on Ethereum datasets demonstrate that our framework achieves superior performance in account identification, yielding 1.13% ~ 4.93% relative improvement over previous state-of-the-art. Furthermore, detailed analyses illustrate the effectiveness of *Ethident* in identifying and understanding the behavior of known participants in Ethereum (e.g. exchanges, miners, etc.), as well as that of the lawbreakers (e.g. phishing scammers, hackers, etc.), which may aid in risk assessment and market regulation.

Index Terms—Blockchain, de-anonymization, behavior pattern, graph neural network, hierarchical graph attention, contrastive learning.

I. INTRODUCTION

THE past few years have witnessed the application of blockchain technology in new technological and industrial revolutions, such as cryptocurrency [1], financial services [2],

supply chain management [3], healthcare [4], etc. As a distributed data storage technology, blockchain is decentralized, traceable and tamper-proof, which guarantees the fidelity and security of data recording and generates trust without a third-party notarization. Benefiting from these characteristics, blockchain has attracted considerable attention and is best known for its crucial role in the field of digital cryptocurrencies, such as Bitcoin and Ethereum. According to statistics from market analysis sites such as CoinMarketCap¹, as of August 2021, about 11,000 types of cryptocurrencies existed, with a total market value of up to 1.9 trillion dollars.

However, blockchain has become a tempting target for hackers and other cybercriminals due to its huge economic value and anonymization. Each individual has a virtual identity on blockchain unrelated to the real one, called pseudonym. For instance, in the Ethereum system, the last 20 bytes of the public key hash are used as the account address (i.e., pseudonym). However, while pseudonymous accounts protect users' privacy, it also provides shelter for illegal transactions, making it difficult for regulators to identify the culprit. At present, the weak regulation of blockchain platforms has led to endless financial crimes such as money laundering, gambling and phishing scams. In 2018, a statistical report published by Kaspersky Lab showed that Ether is the most popular digital asset for criminals, and the loss caused by illegal activities on decentralized applications (DApps) has reached 900 million dollars. Therefore, financial security has become a top priority in the blockchain ecosystem, and it is of great significance to study security strategies for public blockchain in application scenarios such as risk assessment and market regulation.

A. Account Identification vs. Address Clustering

Fortunately, the openness and transparency of blockchain make access to block information without barriers. Recently, existing related work has focused on using the public transaction information to analyze the behavior patterns of accounts and mine the identity information behind them, such as exchanges, phishing scammers, miners and Ponzi schemes, deriving several typical de-anonymization tasks, especially for address clustering and account identification. Fig. 1 shows an illustrative example to explain the difference between address clustering and account identification. From the definition perspective, address clustering aims to partition the address set observed in Bitcoin transactions into maximal address subsets likely controlled by the same entity, i.e., re-identifying multiple

J. Zhou and C. Hu are with the Institute of Cyberspace Security, College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China. E-mail: jjzhou@zjut.edu.cn, ckhu0122@gmail.com.

J. Chi is with the Hangzhou Research Institute of Xidian University, Hangzhou 311231, China. E-mail: chijianlei@gmail.com.

J. Wu is with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China. E-mail: wujiajing@mail.sysu.edu.cn.

M. Shen is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China, and also with Peng Cheng Laboratory (PCL), Shenzhen 518066, China. E-mail: shenmeng@bit.edu.cn.

Q. Xuan is with the Institute of Cyberspace Security, College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China, with the PCL Research Center of Networks and Communications, Peng Cheng Laboratory, Shenzhen 518000, China, and also with the Utron Technology Co., Ltd. (as Hangzhou Qianjiang Distinguished Expert), Hangzhou 310056, China. E-mail: xuanqi@zjut.edu.cn.

Corresponding author: Qi Xuan.

¹<https://coinmarketcap.com/>

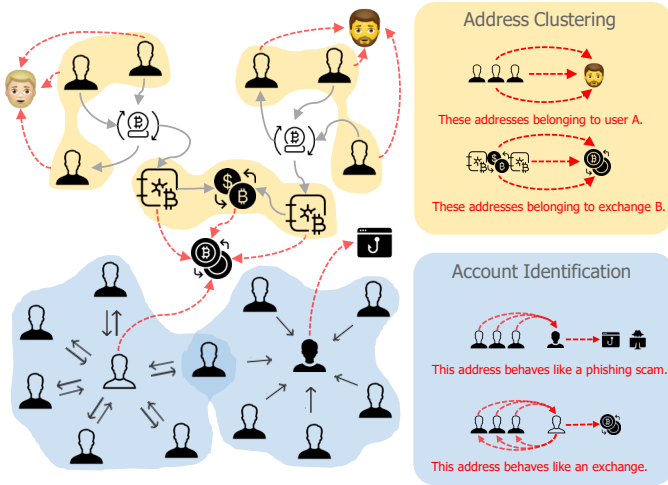


Fig. 1. Illustration of the difference between account identification and address clustering.

addresses belonging to the same entity. Account identification aims to determine the identity type of the account by mining the attributes and behavioral characteristics, i.e., attributing the accounts to specific types. From the task paradigm, the former can be regarded as an unsupervised clustering task, while the latter is generally the supervised classification task. From the application perspective, a large number of existing address clustering methods are usually designed according to the characteristics of the Bitcoin system, and are usually applied to Bitcoin rather than Ethereum due to their technical differences [5]. While account identification methods only rely on general information such as transaction records on the blockchain, as well as external technologies such as machine learning and network science, thus showing better universality.

B. Challenges

In this paper, we focus on de-anonymizing Ethereum accounts through account identification. Existing account identification methods mainly concentrate on manual feature engineering [6]–[9] and graph analytics [10]–[13], which are effective but suffer from several shortcomings and challenges. First, manual feature engineering relies on the prior knowledge of feature designers and is incapable of capturing the underlying information in blockchain data, such as transaction patterns, resulting in low feature utilization and unsound expressiveness. In addition, manual features have weak reusability across different blockchain platforms due to technical differences. For example, Ethereum data has features associated with smart contracts that Bitcoin does not, which greatly limits the reusability of manual features. Second, graph analytics relies on large-scale transaction graphs constructed from mass blockchain data, resulting in high computational consumption and time cost when applying graph random walks or graph neural networks (GNNs). Meanwhile, the growing number of transactions on the blockchain drives frequent updates in the transaction graphs in terms of nodes and edges, which is not conducive to full-graph learning. Lastly, the annotated information of account identities published in the third-party

sites is relatively scarce, resulting in a poor generalization of supervised models.

C. Our Contributions

To tackle these challenges, we design a behavior-aware Ethereum account identification framework (*Ethident*) — an end-to-end graph neural network model, to characterize the behavior patterns of accounts and further achieve account de-anonymization on Ethereum. Specifically, we first collect and collate large amounts of data involving transaction, smart contract and public annotation of account identity from the Ethereum-related platforms, and then construct an Account Interaction Graph (AIG) and its lightweight version. Since the large-scale account interaction graph is not feasible for full-batch training of GNNs, we consider account identification as a subgraph-level classification task, and extract neighborhood subgraphs of target accounts from the complete interaction graph, yielding micro interaction subgraphs, which allows for mini-batch training of GNNs. To better capture the account behavior patterns, we design a Hierarchical Graph ATtention Encoder named *HGATE* as the backbone of our framework, which can effectively characterize the node-level account features and subgraph-level behavior patterns. Furthermore, we introduce data augmentation and contrastive self-supervision mechanism for account identification to alleviate the label scarcity that may lead to poor model generalization during supervised learning. In this way, our framework jointly trains the subgraph contrast and classification tasks, achieving state-of-the-art performance in account identification. The main contributions of this work are summarized as follows:

- **Data collection:** We construct the Account Interaction Graph (AIG) using collected Ethereum data, and further publish the subgraph datasets for account identification research on Ethereum.
- **Scalability:** We consider identifying Ethereum accounts from a graph classification perspective, and design subgraph sampling strategies to achieve scalable account identification.
- **Powerful feature characterization:** We propose a hierarchical graph attention encoder named *HGATE* to effectively characterize the node-level account features and subgraph-level behavior patterns.
- **Generalization:** We establish a behavior-aware Ethereum account identification framework named *Ethident*² which integrates graph augmentation and self-supervision mechanisms, to alleviate the label scarcity and learn highly-expressive behavior pattern representations.
- **State-of-the-art performance:** Extensive experiments on Ethereum datasets demonstrate that our framework can achieve state-of-the-art performance in account identification. We further analyze the behavior patterns of different accounts and illustrate the superiority of our framework in terms of performance, scalability and generalization.

²Data and code are available at <https://github.com/jjzhou012/Ethident>

TABLE I
MAIN SYMBOLS USED IN THIS PAPER.

Symbol	Definition
G, g	Graph, subgraph.
v, V, E	Node (account), node set, edge set.
$\mathcal{N}(i)$	1-hop neighbor set of node v_i .
\mathbf{x}, \mathbf{X}	Node feature, node feature matrix.
\mathbf{e}	Edge feature.
y, Y	Account identity label, label set.
\mathbf{h}	Hidden representation of node feature.
\mathbf{s}, \mathbf{g}	Subgraph representation before/after attentive pooling.
\mathbf{z}	Projection representation of subgraph.
f_θ, f_ψ, f_ϕ	Encoder, prediction head, projection head.
α, β	Normalized attention scores.
a	Unnormalized attention scores.
Θ	Weight parameters.
T	Graph augmentation method.
\mathcal{L}	Loss function.
N	Parameter: batch size.
h	Parameter: hop in subgraph sampling.
K	Parameter: number of sampled neighbors per hop.
λ	Parameter: trade-off hyper-parameter in loss functions.

II. RELATED WORK

De-anonymization in blockchain has received considerable attention for market analysis, abnormal behavior detection, and law enforcement, deriving several mainstream techniques, such as address clustering and account identification.

A. Address Clustering

Early studies [14]–[20] mainly focus on address clustering, also known as user re-identification or entity recognition. Reid et al. [16] proposed the first heuristic for re-identification, named multi-input heuristic, which assumes that the input addresses of a particular transaction are possessed by the same entity. This heuristic is based on the fact that all private keys associated with addresses must be used conjointly to sign a transaction. Androuraki et al. [17] proposed the change address heuristic, which assumes that a new “change” address created by a transaction is likely controlled by the same entity that created the transaction, and has also been applied in [14], [15]. This heuristic stems from the change characteristics of Bitcoin that serves as a mechanism for enhancing user privacy. Martin et al. [18] explored the reasons behind the effectiveness of using the multi-input heuristic for address clustering. Cazabet et al. [19] proposed to construct an identity hint network and applied the Louvain algorithm [21] to detect communities representing the sets of addresses belonging to the same entities.

The aforementioned address clustering methods are widely used in Bitcoin. Robin [5] analyzed the feasibility of two Bitcoin de-anonymization methods of IP linking and address clustering on Ethereum, and concluded that these two methods meet difficulties when applied to Ethereum due to technical differences. Friedhelm [22] proposed three heuristics that exploit patterns related to deposit addresses, multiple participation in airdrops and token authorization mechanisms, and quantified

the feasibility of each heuristic over the first four years of the Ethereum. Shlomi et al. [23] assumed that the smart contract code written by the same author has a unique style, and further linked contract addresses with similar code styles together, thinking that these addresses are generated by the same author.

B. Account Identification

Thanks to the openness of blockchain transactions, as well as the development of machine learning and network science, a new class of de-anonymization strategies — account identification, has been proposed and comprehensively developed. Existing account identification methods mainly concentrate on manual feature engineering and graph analytics.

1) *Manual Feature Engineering*: Manual feature engineering extremely relies on the prior knowledge of feature designers. Normally, the more expert experience involved, the more reliable the manual features are. Toyoda et al. [6] extracted seven statistical features such as the rate of bitcoin coinbase transactions to infer account identities. Lin et al. [7] designed various features associated with transaction timestamps and analyzed the importance of each one. Bartoletti et al. [8] designed the Gini coefficient and the characteristics of possible abnormal behavior patterns to infer the Ponzi accounts in the transaction network. Marc et al. [24] designed a large number of manual features associated with addresses, entities and graph motifs in Bitcoin transaction networks, and classified different Bitcoin entities via LightGBM [25]. In addition, some emerging public blockchains contain smart contracts, providing new features. Huang et al. [9] considered the calling information of smart contracts to expand the feature space, and realized the identification of bot accounts in EOSIO.

2) *Graph Analytics*: Massive transaction data can be modeled as graphs, and a considerable part of existing methods regards account identification as a classification task from a graph perspective. Li et al. [10] considered the topological features of accounts and found the difference in topological structure between the Ponzi accounts and the normal ones. Yuan et al. [11] applied graph random walks such as DeepWalk [26] and Node2vec [27] to learn account features in the transaction graph. Wu et al. [28] performed graph random walks by considering both the transaction amount and timestamp information, proposing a novel embedding method named Trans2Vec to extract the address feature for phishing detection. Yuan et al. [29] extracted the subgraphs for each target account and embedded their transaction topology via Graph2Vec [30]. Moreover, they introduced the SGN mechanism [31] to further enhance the transaction structure embedding. Chen et al. [13] also extracted transaction subgraphs and got the embeddings by a graph convolution layer combining graph auto-encoder in an unsupervised manner, finally achieving phishing detection by LightGBM. Shen et al. [12] constructed the account interaction graphs using Ethereum and EOSIO data, and proposed an end-to-end graph convolution network model to identify different categories of accounts or bots.

Besides the aforementioned methods, there are other frameworks to achieve identity identification. Phetsouvanh et al. [32] proposed a graph mining technology to detect suspicious

TABLE II
INFORMATION OF RAW ETHEREUM BLOCK DATA.

Data Field	Custom Symbol	Definition
blockNumber		The block ID where the transaction is located.
timestamp	d	The timestamp of a transaction.
from	v	The account that initiates the transaction.
to	v	The account that receives the transaction.
fromIsContract		Whether the transaction is sent by a CA.
toIsContract		Whether the transaction is received by a CA.
callingFunction	f	The name of function called if there is a contract call.
value	w	The transaction amount.

bitcoin flows and accounts by analyzing the path length and confluence account of the directed subgraph. Zhang et al. [33] introduced the concept of meta-path from the heterogeneous network. This method deals with the bitcoin network from both static and dynamic perspectives and can effectively detect abnormal accounts and transactions.

III. ACCOUNT INTERACTION GRAPH MODEL

A. Problem Description

In this paper, we mainly focus on identifying accounts in Ethereum via deep graph analytics, especially from a graph classification perspective. A transaction graph constructed from blockchain transaction data is typically represented by a graph $G = (V, E, \mathbf{X}, \mathbf{E}, Y)$, where $V = \{v_1, v_2, \dots, v_n\}$ is the set of account nodes, $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V\}$ is the set of interaction edges, $\mathbf{X} \in \mathbb{R}^{n \times F_1}$ is the node feature matrix, and $\mathbf{E} \in \mathbb{R}^{m \times F_2}$ is the edge feature matrix (we assume, $|E| = m$). We use $Y = \{(v_i, y_i) \mid v_i \in V\}$ to represent the label set of partial account nodes. The subgraph of an account node v can be represented as $g_v \subset G$. For the given transaction graph G , subgraph-level account identification is to learn a function $f(g_v) \mapsto y$ mapping the pattern of account subgraph g_v to the identity label y .

B. Ethereum and Block Data

Ethereum is the second-largest blockchain platform after Bitcoin, and it allows users to conduct complex transactions based on *smart contracts*, which are applications that run on Ethereum virtual machines. An *account* in Ethereum is an entity that owns Ether and can be divided into two categories: Externally Owned Account (EOA) and Contract Account (CA). EOA is controlled by a user who owns the private key of the account, and can initiate transactions. CA is controlled by smart contract code, which cannot initiate transactions actively and can only be executed according to the pre-written smart contract code after being triggered. Between Ethereum accounts, there are usually two types of interactions: *transaction* and *contract call*. The *transaction* must be initiated by EOA, and can be received by EOA or CA. The *contract call* refers to the process of triggering the smart contract code in CA to perform different operations. The Ethereum blockchain is a succession of blocks, and each block contains a set of transactions and contract calls. The raw block data of Ethereum is structural and provides a wealth of information, as listed in Table II.

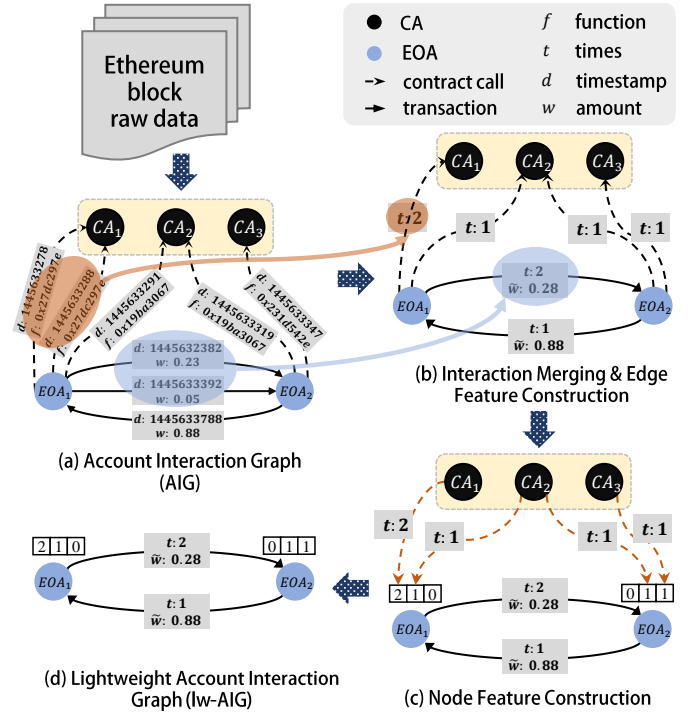


Fig. 2. Constructing Account Interaction Graph and its lightweight version.

C. Account Interaction Graph

The raw block data is informative and provides the details of transactions and contract calls, by which we can construct an Account Interaction Graph (AIG), as defined below.

Definition 1. (Account Interaction Graph, AIG): a directed, weighted and heterogeneous multigraph $G = (V_{eoa}, V_{ca}, E_t, E_c, Y)$, where V_{eoa} and V_{ca} are the set of EOA and CA respectively, $E_t = \{(v_i, v_j, d, w) \mid v_i, v_j \in V_{eoa}\}$ is the directed edge set constructed from transaction information, and $E_c = \{(v_i, v_j, d, f) \mid v_i \in V_{eoa} \cup V_{ca}, v_j \in V_{ca}\}$ is the directed edge set constructed from contract call information. The three edge attributes d, w, f represent *timestamp*, *value* and *callingFunction* respectively in Table II. The AIG is partially labeled, i.e., a few EOA have identity labels y and can compose the labeled node set $Y = \{(v_i, y_i) \mid v_i \in V_{eoa}\}$.

The original AIG is a heterogeneous multigraph that has dense connections as well as different types of information attached to nodes and edges, as shown in Fig. 2(a). The heterogeneity and multiple edges significantly increase the complexity of information mining. So we further simplify the AIG into a homogeneous and more sparse graph by **interaction merging** and **feature construction**.

Definition 2. (Lightweight Account Interaction Graph, lw-AIG): a directed, weighted and homogeneous graph $G = (V_{eoa}, \tilde{E}_t, \mathbf{X}, \mathbf{E}, Y)$, where $\tilde{E}_t = \{(v_i, v_j, t, \tilde{w}) \mid v_i, v_j \in V_{eoa}\}$, \mathbf{X} is the node feature matrix constructed from contract call information and \mathbf{E} is the edge feature matrix. The edge attribute t denotes the number of directed interactions from v_i to v_j , and the edge attribute \tilde{w} denotes total transaction amount from v_i to v_j .

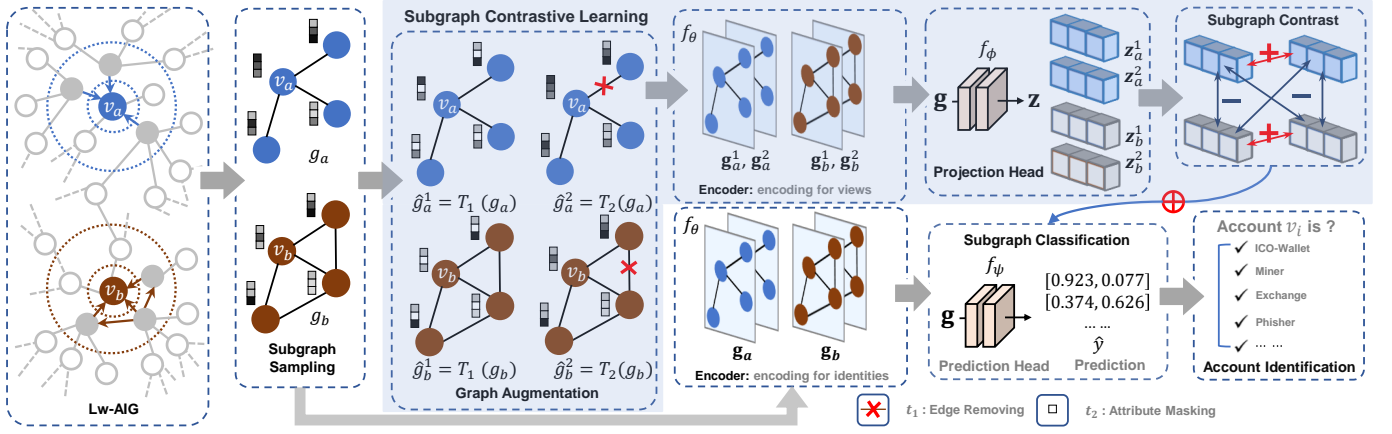


Fig. 3. The architecture of Ethident. The complete workflow proceeds as follows: (1) sampling subgraphs centered on target accounts from lw-AIG; (2) applying two augmentation operators on each subgraph to generate two correlated views; (3) encoding subgraphs and corresponding augmented views; (4) optimizing the GNN encoder by jointly training subgraph contrast and classification tasks.

1) Interaction Merging and Edge Feature Construction:

During interaction merging, as shown in Fig. 2(b), multiple directed interactions (transactions or contract calls) from the source account v_i to the target account v_j will be merged into a single edge with a newly added edge attribute t representing the number of merged interactions. For transactions, another new edge attribute \tilde{w} represents the total transaction amount of merged interactions. In addition, a feature pruning operation will take effect, removing the two raw edge attributes of *timestamp* d and *callingFunction* f . Finally, we represent the edge feature vector for arbitrary transaction edge $(v_i, v_j) \in E_t$ as $\mathbf{e}_{ij} = [t, \tilde{w}]$.

2) *Node Feature Construction*: The behavior characteristics of an account are not only related to its transaction objects, amount and frequency, but also to the smart contracts it calls. Accounts with different behavior patterns have different calling preferences for smart contracts. Therefore, we can construct account features using the information on contract call, as shown in Fig. 2(c). Specifically, let n and F be the number of EOA and CA respectively in AIG, we can construct an account feature matrix $\mathbf{X} \in \mathbb{R}^{n \times F}$ to represent the preference for contract call, as formulated below:

$$\begin{aligned} \mathbf{X} &= [\mathbf{x}_1; \dots; \mathbf{x}_i; \dots; \mathbf{x}_n]^\top, \\ \mathbf{x}_i &= [t_1, \dots, t_j, \dots, t_F], \\ \text{where } t_j &= \begin{cases} t & \text{If there are } t \text{ calls to } v_j^{ca}; \\ 0 & \text{If there is no call to } v_j^{ca}; \end{cases} \end{aligned} \quad (1)$$

Note that \mathbf{x}_i is the feature of v_i^{EOA} . During feature construction, we convert the AIG to a homogeneous lw-AIG.

In summary, the node features of the lw-AIG reflect the contract call information, and the edge features reflect the transaction information.

IV. METHODOLOGY

In this section, we provide the details of the proposed framework *Ethident*, as schematically depicted in Fig. 3. For a target account v_i , the input of *Ethident* is the account interaction subgraph g_i sampled from lw-AIG, and the output is the predictive identity label \hat{y}_i . Our framework is mainly composed of the

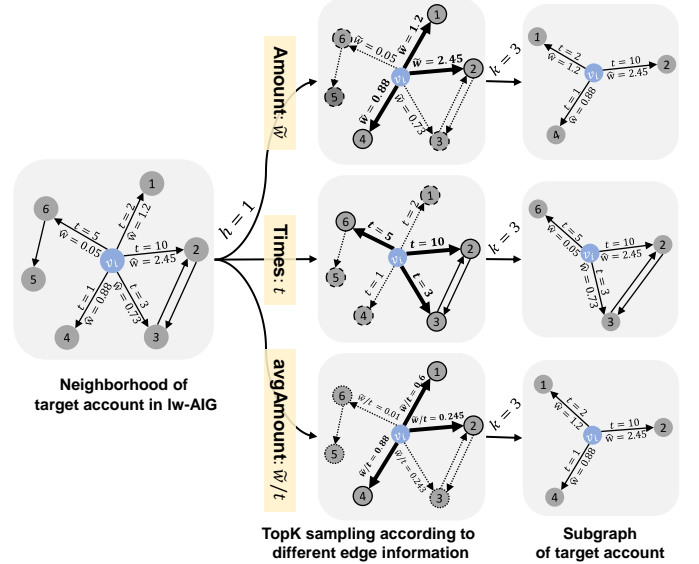


Fig. 4. Subgraph sampling according to different edge information.

following components: (1) a subgraph extractor that captures the micro interaction subgraphs centered on target accounts from the lw-AIG topology; (2) a subgraph augmentation module that generates a series of variant graph views using various transformations on subgraphs; (3) a GNN encoder that encodes the subgraphs as expressive representations via hierarchical graph attention mechanism; (4) a training module that jointly trains the subgraph contrast and classification tasks. Next, we describe the details of each component.

A. Subgraph Sampling

The raw data contains tens of millions of blocks, making AIG a large-scale graph and not feasible for full-batch training of GNNs. Even though the lightweight process greatly simplifies AIG, it still maintains a large number of EOA nodes. On the other hand, existing account identification methods based on graph embedding or GNNs generally rely on full-batch training, which restricts their scalability on large-scale graphs

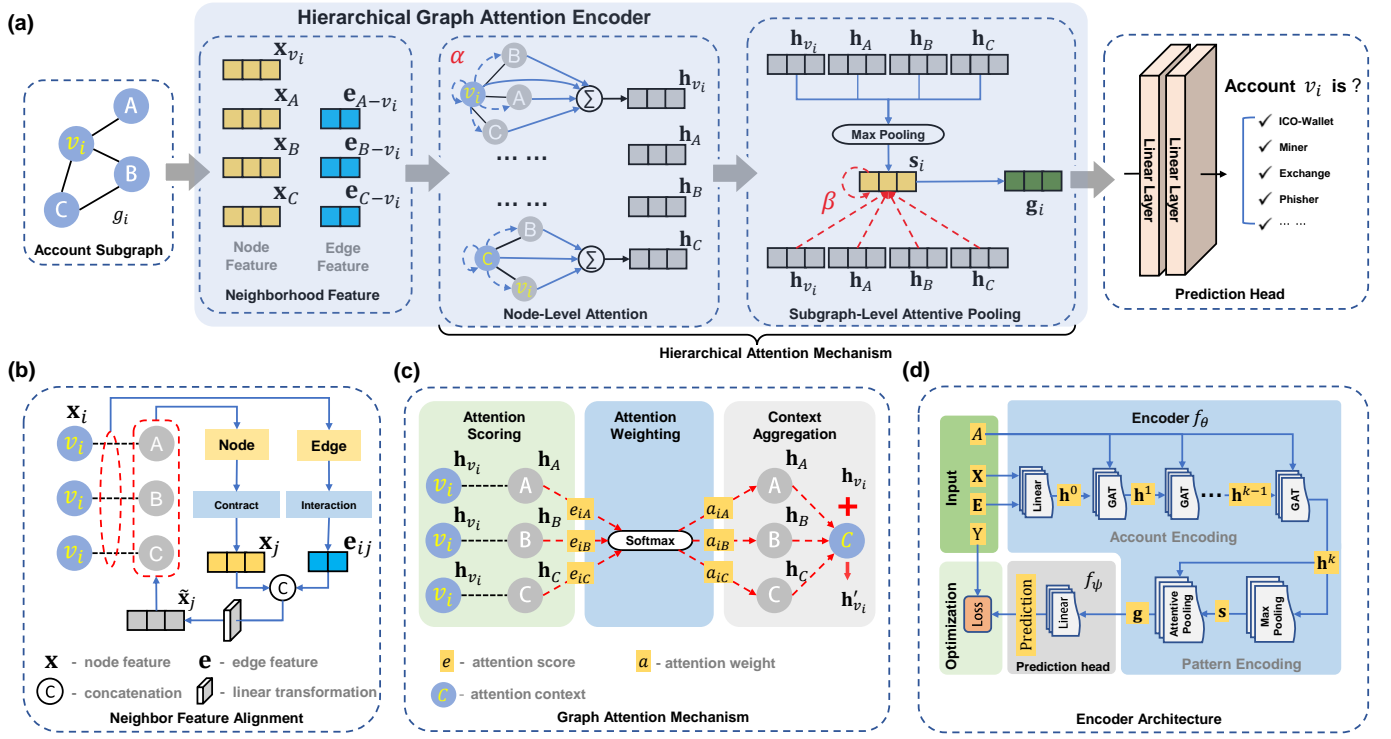


Fig. 5. Schematic depiction of the hierarchical graph attention encoder (HGATE): (a) the pipeline of HGATE on account identification; (b) the process of neighbor feature alignment; (c) the illustration of graph attention mechanism; (d) the model architecture of HGATE.

for account representation learning. Thus, we consider account identification as a subgraph-level classification task based on the following facts: (1) different types of accounts have different behavior patterns, implicit in their local structure; (2) subgraph consisting of the target account and its local neighborhood information (neighbors and their interactions) is informative and plays a critical role in providing behavior patterns for account identification; (3) subgraph is the receptive field of the center target node, which is much smaller than the whole graph and allows for mini-batch training.

In this work, we consider subgraph sampling that allows for mini-batch training of GNNs on large-scale graphs. We perform TopK sampling to obtain the h -hop interaction subgraphs according to different edge information: **Amount** (\tilde{w}), **Times** (t) or average **Amount** (**avgAmount**, \tilde{w}/t). Specifically, for a target account node v_i , we sample top- K most important neighbors based on one of the edge attributes, and again sample top- K most important neighbors for each account sampled at the previous hop, and recursive ones in the downstream hops. The recursive sampling can be formulated as follows:

$$V_k = \bigcup_{v \in V_{k-1}} \text{topK}(\mathcal{N}_v, K, \mathbf{E}[v, \mathcal{N}_v, i]), \quad i \in \{0, 1, 2\}, \quad (2)$$

where V_k is the set of nodes sampled at hop k and $V_0 = \{v_i\}$, \mathcal{N}_v is the 1-hop neighbor set of node v , K is the number of sampled neighbors per hop, $\mathbf{E}[v, \mathcal{N}_v, i]$ is the edge attributes of candidate interactions that guides the neighbor sampling, i is an indicator of which edge attribute to use, and topK is the function that returns the top- K most important nodes. After h iterations, we obtain the account set $V_i = \bigcup_{k=0}^h V_k$ sampled

from lw-AIG, and the subgraph g_i of target account v_i can be induced by V_i from the lw-AIG. Fig. 4 illustrates the process of subgraph sampling according to different edge information. For the labeled target account set Y , their corresponding subgraphs form a dataset: $D = \{(g_i, y_i) \mid \forall (v_i, y_i) \in Y\}$. Note that we assign the label of the target account to the subgraph, and aim to learn a function mapping the subgraph patterns to account identity labels.

B. Encoder Architecture

The backbone of *Ethident* is the designed GNN encoder named *HGATE*, which is capable of learning expressive representations for accounts and their behavior patterns, as schematically depicted in Fig. 5. This encoder learns account and pattern embeddings via a hierarchical attention mechanism, and can also implement account identification independently by following a prediction head, as shown in Fig. 5(a). Next, we describe the details of our encoder f_θ .

1) *Neighbor Feature Alignment*: For lw-AIG, its nodes and edges are encoded according to contract call and transaction information. Since our encoder is account-centric, each account v_i has its neighbor features that concatenate both neighboring account features (\mathbf{x}_j) and the connecting interaction features (\mathbf{e}_{ij}), represented as $[\mathbf{x}_j \parallel \mathbf{e}_{ij}]$. Here we need to perform a column normalization on neighbor features to eliminate the dimensional differences between different attributes. Note that the target account feature $\mathbf{x}_i \in \mathbb{R}^F$ and its neighbor features $[\mathbf{x}_j \parallel \mathbf{e}_{ij}] \in \mathbb{R}^{F+2}$ do not have the same dimension, so a linear transformation and a nonlinear activation are performed to align the feature dimension, as shown in Fig. 5(b). This procedure

can be achieved via a fully connected layer parameterized by Θ_x as follows, and generates aligned embeddings for neighbors of the target account.

$$\tilde{\mathbf{x}}_j = \text{LeakyRelu}(\Theta_x \cdot [\mathbf{x}_j \parallel \mathbf{e}_{ij}]). \quad (3)$$

2) *Node-level Attention for Account Embedding*: This module aims to preserve the relevance of interactive accounts in the input subgraph, and learns account representation by focusing on the most relevant parts of the neighborhood. When identifying a target account in the interaction subgraph, different neighboring accounts generally contribute differently to it. For example, both accounts v_a and v_b have transactions with account v_i , if v_a has many high-volume transactions with v_i while v_b has only one low-volume transaction with v_i , or if v_a has a more similar preference of contract call with v_i than v_b , then v_a often plays a more important role in identifying v_i since it preserves more information associated with the identity of v_i . Based on the above understanding and inspired by previous work [34], we utilize the node-level attention mechanism, as illustrated in Fig. 5(c), to learn the hidden representation of each account in the input subgraph by composing its neighbor features with different contributions (attentions).

Specifically, for arbitrary account v_i in the input subgraph g , the node-level attention mechanism learns the contribution attention scores for its neighbors v_j , as follows:

$$a_{ij}^l = \text{LeakyRelu}(\Theta_n^l \cdot [\mathbf{h}_i^l \parallel \mathbf{h}_j^l]), \quad (4)$$

where a linear transformation parameterized by Θ_n^l and a nonlinear *LeakyRelu* activation are performed together to compute the importance of account v_j 's hidden features to account v_i in l -th layer. Subsequently, to make attention scores easily comparable across different accounts, the attention scores a are further normalized using the softmax function over the neighbor accounts:

$$\alpha_{ij}^l = \text{Softmax}(a_{ij}^l) = \frac{\exp(a_{ij}^l)}{\sum_{x \in \mathcal{N}(i) \cup \{i\}} \exp(a_{ix}^l)}, \quad (5)$$

where $\mathcal{N}(i)$ is the 1-hop neighbor set of account v_i . Once obtained, the normalized attention scores are used to update the features of target account via neighborhood context aggregation:

$$\mathbf{h}_i^{l+1} = \text{Elu} \left(\alpha_{ii}^l \cdot \Theta_\alpha^l \cdot \mathbf{h}_i^l + \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^l \cdot \Theta_\alpha^l \cdot \mathbf{h}_j^l \right), \quad (6)$$

where a linear transformation parameterized by Θ_α^l and a nonlinear *Elu* activation are used to compute the final output features.

The node-level attention mechanism serves for account embedding. Specifically, we use a stack of k graph attention layers to capture the account features, as illustrated in Fig. 5(d). The input of this stack is the initial account embedding \mathbf{h}^0 generated by a fully connected layer that accepts the account and interaction features (Eq. (3)). Notably, for a target account v_i , its initial embedding is $\mathbf{h}_i^0 = \mathbf{x}_i$, and that of its neighboring accounts v_j is $\mathbf{h}_j^0 = \tilde{\mathbf{x}}_j$. To better characterize accounts, the stack performs an iterative process of transferring, transforming, aggregating and updating the representation from interactive

neighbors. And after k iterations, the final output account embeddings \mathbf{h}^k contain the interaction influence within k -hops.

3) *Subgraph-level Attentive Pooling for Pattern Embedding*: This module aims to characterize the behavior patterns of target accounts in the input subgraphs by extracting expressive subgraph-level features. Actually, the behavior patterns of accounts are associated with their identities, i.e., accounts of different identities usually behave differently and have different subgraph patterns. For ‘‘Exchange’’ subgraphs, the center node generally has an extremely high centrality and frequently interacts with surrounding neighbors, indicating high-volume transaction orders. For ‘‘Ponzi’’ or ‘‘Gambling’’ subgraphs, there exist two explicit characteristics indicating high investment and low return: (1) bi-directional edges (mutual transactions) between the center node and surrounding neighbors are rare, and the center node has high in-degree and low out-degree; (2) the incoming edges (investment) of the center node contain larger feature values associated with the digital currency than the outgoing edges (return). Therefore, different accounts contribute differently to characterize the subgraph pattern reflecting the behavior of the target account. Meanwhile, traditional practice usually captures the graph-level features using sum, mean or max pooling, resulting in feature smoothing and poor expressiveness. Based on the above understanding, we design a novel subgraph-level attentive pooling module, as illustrated in Fig. 5(a), to learn the expressive representation of account subgraphs.

Specifically, for a subgraph g , we first obtain the initial subgraph-level embedding \mathbf{s} by using global max pooling over all account embeddings in the subgraph:

$$\mathbf{s} = \text{MaxPooling}(\mathbf{h}^k). \quad (7)$$

Note that the input of the *MaxPooling* layer is the final account embeddings \mathbf{h}^k generated in Sec. IV-B2. To better characterize the subgraph pattern, we update \mathbf{s} by aggregating features of all accounts with different contributions (attentions). In other words, for the initial subgraph embedding \mathbf{s} , we use an attention mechanism to learn the contribution attention score for arbitrary account v_j in the subgraph as follows:

$$a_j = \text{LeakyRelu}(\Theta_s \cdot [\mathbf{s} \parallel \mathbf{h}_j^k]), \quad (8)$$

where a linear transformation parameterized by Θ_s and a nonlinear *LeakyRelu* activation are performed to compute the importance of account v_j 's hidden features to the initial subgraph embedding \mathbf{s} . Same as the node-level attention, a softmax function is applied to compute the normalized attention scores:

$$\beta_j = \text{Softmax}(a_j) = \frac{\exp(a_j)}{\sum_{x \in V_g \cup \{s\}} \exp(a_x)}, \quad (9)$$

where V_g is the node set of subgraph g , and a_s is the self-attention score of \mathbf{s} . Finally, the attentive pooling performs the update process as follows:

$$\mathbf{g} = \text{Elu} \left(\beta_s \cdot \Theta_\beta \cdot \mathbf{s} + \sum_{j \in V_g} \beta_j \cdot \Theta_\beta \cdot \mathbf{h}_j^k \right), \quad (10)$$

where a linear transformation parameterized by Θ_β and a nonlinear Elu activation are used to compute the final subgraph embedding \mathbf{g} which characterizes the behavior pattern of the target account.

C. Subgraph Contrastive Learning

To alleviate the account label scarcity as well as learn highly-expressive pattern embeddings, our *Ethident* introduce the contrastive self-supervision learning as a regularization to jointly train the GNN encoder.

1) *Graph Augmentation*: Contrastive learning relies heavily on well-designed data augmentation strategies for view generation. So far, widely used techniques concentrate on structure-level and attribute-level augmentation [35]–[37]. In this paper, we use three categories of graph augmentation methods to generate the augmented views of subgraphs.

- **Structure-level Augmentation**

- **Node Dropping**: Each node has a certain probability \mathcal{P} to be dropped from subgraph.
- **Edge Removing**: Each edge has a certain probability \mathcal{P} to be removed from subgraph.

- **Attribute-level Augmentation**

- **Node Attribute Masking**: Each dimension of node features has a certain probability \mathcal{P} to be set as zero.
- **Edge Attribute Masking**: Each dimension of edge features has a certain probability \mathcal{P} to be set as zero.

- **Sampling-based Augmentation**

Since each subgraph is sampled from lw-AIG via one of the three sampling strategies mentioned in Sec. IV-A, we can use the other two sampling methods to generate the sampling-based augmented views for this subgraph.

During graph augmentation, we generate two augmented views \hat{g}_i^1, \hat{g}_i^2 for each target account subgraph g_i , and assign the identity label of target account to them as a pseudo label:

$$\begin{aligned} D_{\text{aug1}} &= \{(\hat{g}_i^1, y_i) \mid \hat{g}_i^1 = T_1(g_i); (v_i, y_i) \in Y\}, \\ D_{\text{aug2}} &= \{(\hat{g}_i^2, y_i) \mid \hat{g}_i^2 = T_2(g_i); (v_i, y_i) \in Y\}. \end{aligned} \quad (11)$$

In this way, we can scale up the training data and alleviate label scarcity. The raw and augmented datasets will be used together to train the encoder.

2) *Subgraph Contrast*: In our contrastive learning setting, for each account subgraph g_i , its two correlated views \hat{g}_i^1 and \hat{g}_i^2 are generated by undergoing two augmentation operators T_1 and T_2 , where $\hat{g}_i^1 = T_1(g_i)$ and $\hat{g}_i^2 = T_2(g_i)$. The correlated augmented views are fed into the encoder f_θ , producing the whole subgraph representations \mathbf{g}_i^1 and \mathbf{g}_i^2 . Then they are mapped into an embedding space for contrast via a projection head f_ϕ , yielding \mathbf{z}_i^1 and \mathbf{z}_i^2 . Note that θ and ϕ are the parameters of graph encoder and projection head respectively. Finally, the goal of subgraph-level contrast is to maximize the consistency between two correlated augmented views of subgraphs in the contrastive space via minimizing the contrastive loss:

$$\mathcal{L}_{\text{self}} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}_i, \quad (12)$$

TABLE III

STATISTICS OF SUBGRAPH DATASETS SAMPLED FROM ACCOUNT INTERACTION GRAPH OF ETHEREUM. $|G|$ IS THE NUMBER OF SUBGRAPHS, Avg. $|V|$ AND Avg. $|E|$ ARE THE AVERAGE NUMBER OF NODES AND EDGES IN SUBGRAPHS RESPECTIVELY, $|\mathbf{x}|$ AND $|\mathbf{e}|$ ARE THE NUMBER OF NODE AND EDGE FEATURES IN SUBGRAPHS.

Dataset	$ G $	Avg. $ N $	Avg. $ E $	$ \mathbf{x} $	$ \mathbf{e} $
Eth-ICO-A	146	42.5	141.3	14885	2
Eth-ICO-T	146	52.2	152.6	14885	2
Eth-ICO-aA	146	42.4	140.7	14885	2
Eth-Mining-A	130	23.7	72.9	14885	2
Eth-Mining-T	130	24.7	67.2	14885	2
Eth-Mining-aA	130	29.0	91.9	14885	2
Eth-Exchange-A	386	33.6	123.7	14885	2
Eth-Exchange-T	386	38.0	113.4	14885	2
Eth-Exchange-aA	386	38.6	148.6	14885	2
Eth-Phish&Hack-A	5070	37.3	110.8	14885	2
Eth-Phish&Hack-T	5070	37.8	101.6	14885	2
Eth-Phish&Hack-aA	5070	37.8	111.3	14885	2

where N is the number of subgraphs in a batch (i.e., batch size). The loss for each subgraph can be computed as:

$$\mathcal{L}_i = -\log \frac{e^{\cos(\mathbf{z}_i^1, \mathbf{z}_i^2)/\tau}}{\sum_{j=1, j \neq i}^N e^{\cos(\mathbf{z}_i^1, \mathbf{z}_j^2)/\tau}}, \quad (13)$$

where $\cos(\cdot, \cdot)$ is the cosine similarity function with $\cos(\mathbf{z}_i^1, \mathbf{z}_j^2) = \mathbf{z}_i^{1\top} \mathbf{z}_j^2 / \|\mathbf{z}_i^1\| \|\mathbf{z}_j^2\|$, and τ is the temperature parameter. The two correlated views \mathbf{z}_i^1 and \mathbf{z}_i^2 of account subgraph g_i are treated as a positive pair while the rest view pairs in the batch are treated as negative pairs. The objective aims to maximize the consistency of positive pairs as opposed to negative ones, i.e., contrastive learning allows accounts of the same type to have more consistent representations, and makes accounts of different types have more obvious differences.

D. Model Training

We achieve account identification by a prediction head f_ψ , which maps the subgraph representations to labels reflecting account identity, yielding a classification loss:

$$\mathcal{L}_{\text{pred}} = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(f_\psi(\mathbf{g}_i)), \quad (14)$$

where $\mathcal{L}_{\text{pred}}$ is the cross entropy loss.

The self-supervised subgraph contrast is a pretext task that serves as a regularization of the subgraph classification task. The encoder *HGATE* is jointly trained with the pretext and subgraph classification tasks. The loss function consists of both the self-supervision and classification task loss functions, as formularized below:

$$\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda \cdot \mathcal{L}_{\text{self}}, \quad (15)$$

where λ is a trade-off hyper-parameter controls the contribution of the self-supervision term.

TABLE IV

SUMMARY OF PERFORMANCE ON ACCOUNT IDENTIFICATION IN TERMS OF F1-SCORE IN PERCENTAGE WITH STANDARD DEVIATION. THE HIGHEST PERFORMANCE IS MARKED WITH BOLDFACE; THE HIGHEST PERFORMANCE OF DIFFERENT CATEGORIES OF BASELINES IS UNDERLINED. THE OPS. VALUE STANDS FOR THE OPTIMAL PERFORMANCE STATISTICS OF ALL GRAPH EMBEDDING AND GNN-BASED METHODS UNDER DIFFERENT SUBGRAPH DATASETS.

Method	Dataset (with different sampling strategy)											
	Eth-ICO			Eth-Mining			Eth-Exchange			Eth-Phish&Hack		
	Amount	Times	avgAmount	Amount	Times	avgAmount	Amount	Times	avgAmount	Amount	Times	avgAmount
Manual + LR		76.73 \pm 0.059			77.15 \pm 0.036			87.34 \pm 0.037			80.94 \pm 0.042	
Manual + RF	←	<u>79.52\pm0.045</u>	→	←	<u>81.32\pm0.044</u>	→	←	<u>90.13\pm0.030</u>	→	←	<u>90.10\pm0.007</u>	→
Manual + LGBM		74.71 \pm 0.046			<u>82.16\pm0.051</u>			<u>91.25\pm0.030</u>			<u>90.51\pm0.007</u>	
DeepWalk + LR	56.69 \pm 0.094	58.96 \pm 0.054	59.64 \pm 0.049	56.94 \pm 0.067	60.26 \pm 0.066	62.07 \pm 0.084	59.17 \pm 0.059	63.82 \pm 0.049	62.53 \pm 0.045	58.73 \pm 0.020	67.99 \pm 0.028	64.29 \pm 0.011
DeepWalk + RF	73.24 \pm 0.078	68.93 \pm 0.045	67.12 \pm 0.076	65.13 \pm 0.045	71.58 \pm 0.078	65.40 \pm 0.035	76.31 \pm 0.049	80.53 \pm 0.038	80.19 \pm 0.020	91.14 \pm 0.012	89.77 \pm 0.008	<u>92.71\pm0.008</u>
DeepWalk + LGBM	58.28 \pm 0.085	59.18 \pm 0.076	58.28 \pm 0.061	60.29 \pm 0.077	56.17 \pm 0.078	61.56 \pm 0.062	73.73 \pm 0.035	73.21 \pm 0.062	71.49 \pm 0.040	89.78 \pm 0.006	89.65 \pm 0.009	92.13 \pm 0.007
Node2Vec + LR	80.95 \pm 0.054	62.36 \pm 0.055	79.37 \pm 0.079	64.36 \pm 0.061	72.85 \pm 0.063	79.24 \pm 0.046	66.06 \pm 0.024	82.77 \pm 0.029	81.05 \pm 0.029	66.53 \pm 0.020	82.58 \pm 0.007	79.73 \pm 0.009
Node2Vec + RF	<u>88.21\pm0.048</u>	<u>78.91\pm0.051</u>	<u>89.34\pm0.042</u>	<u>74.37\pm0.041</u>	<u>78.48\pm0.068</u>	78.72 \pm 0.058	<u>83.12\pm0.039</u>	<u>88.98\pm0.033</u>	86.56 \pm 0.012	<u>92.04\pm0.003</u>	<u>94.06\pm0.005</u>	92.17 \pm 0.007
Node2Vec + LGBM	81.41 \pm 0.064	65.53 \pm 0.086	80.95 \pm 0.040	65.45 \pm 0.073	73.36 \pm 0.060	78.72 \pm 0.052	80.28 \pm 0.033	87.68 \pm 0.022	85.79 \pm 0.012	91.67 \pm 0.006	<u>94.00\pm0.005</u>	91.90 \pm 0.005
Struc2Vec + LR	61.00 \pm 0.051	58.96 \pm 0.082	55.10 \pm 0.084	51.82 \pm 0.059	61.56 \pm 0.072	59.80 \pm 0.093	63.48 \pm 0.038	57.02 \pm 0.031	59.09 \pm 0.056	57.02 \pm 0.012	59.47 \pm 0.014	54.89 \pm 0.013
Struc2Vec + RF	61.45 \pm 0.069	60.32 \pm 0.079	60.09 \pm 0.057	63.61 \pm 0.080	69.44 \pm 0.058	60.81 \pm 0.059	74.07 \pm 0.030	71.66 \pm 0.035	69.60 \pm 0.033	66.33 \pm 0.016	68.93 \pm 0.007	65.02 \pm 0.007
Struc2Vec + LGBM	62.36 \pm 0.053	56.01 \pm 0.059	58.05 \pm 0.055	55.12 \pm 0.041	60.25 \pm 0.045	62.80 \pm 0.072	70.97 \pm 0.030	71.49 \pm 0.024	68.13 \pm 0.045	65.25 \pm 0.018	68.53 \pm 0.009	64.52 \pm 0.010
Trans2Vec + LR	73.77 \pm 0.081	61.05 \pm 0.064	59.64 \pm 0.066	71.41 \pm 0.068	53.78 \pm 0.076	75.87 \pm 0.061	57.52 \pm 0.034	76.81 \pm 0.041	79.75 \pm 0.034	68.67 \pm 0.019	63.05 \pm 0.016	58.21 \pm 0.013
Trans2Vec + RF	86.50 \pm 0.051	71.02 \pm 0.060	73.16 \pm 0.067	73.34 \pm 0.072	61.85 \pm 0.084	77.87 \pm 0.065	72.91 \pm 0.046	82.65 \pm 0.042	<u>87.31\pm0.025</u>	89.94 \pm 0.008	89.75 \pm 0.006	89.98 \pm 0.007
Trans2Vec + LGBM	73.91 \pm 0.063	59.88 \pm 0.059	59.30 \pm 0.059	67.94 \pm 0.069	50.81 \pm 0.084	72.60 \pm 0.072	68.19 \pm 0.035	76.30 \pm 0.040	84.67 \pm 0.030	87.90 \pm 0.008	88.46 \pm 0.010	89.65 \pm 0.006
Graph2Vec + LR	65.21 \pm 0.061	68.21 \pm 0.067	63.90 \pm 0.063	53.25 \pm 0.068	48.33 \pm 0.080	56.24 \pm 0.073	66.45 \pm 0.037	61.58 \pm 0.036	66.79 \pm 0.043	80.73 \pm 0.009	78.92 \pm 0.006	79.94 \pm 0.008
Graph2Vec + RF	66.71 \pm 0.072	71.15 \pm 0.057	65.75 \pm 0.049	53.68 \pm 0.080	49.94 \pm 0.072	57.38 \pm 0.081	66.50 \pm 0.044	63.42 \pm 0.033	64.72 \pm 0.034	80.91 \pm 0.008	78.97 \pm 0.008	79.60 \pm 0.008
Graph2Vec + LGBM	61.02 \pm 0.070	63.02 \pm 0.063	58.29 \pm 0.071	52.32 \pm 0.072	46.56 \pm 0.089	57.62 \pm 0.078	64.09 \pm 0.046	60.65 \pm 0.041	61.17 \pm 0.041	82.11 \pm 0.009	80.43 \pm 0.009	81.44 \pm 0.006
GCN	87.57 \pm 0.112	86.73 \pm 0.109	86.89 \pm 0.126	75.25 \pm 0.130	<u>82.91\pm0.098</u>	83.52 \pm 0.073	90.23 \pm 0.021	89.85 \pm 0.026	89.79 \pm 0.026	<u>96.49\pm0.005</u>	<u>96.15\pm0.006</u>	<u>96.49\pm0.006</u>
GAT	88.44 \pm 0.084	90.02 \pm 0.047	86.11 \pm 0.147	80.28 \pm 0.111	78.63 \pm 0.152	82.71 \pm 0.084	<u>91.01\pm0.024</u>	91.42 \pm 0.023	<u>90.70\pm0.021</u>	95.87 \pm 0.006	95.68 \pm 0.005	95.91 \pm 0.005
GIN	75.20 \pm 0.104	78.66 \pm 0.102	75.17 \pm 0.096	59.07 \pm 0.106	57.48 \pm 0.113	63.99 \pm 0.129	81.56 \pm 0.074	84.35 \pm 0.067	85.42 \pm 0.055	95.89 \pm 0.007	95.79 \pm 0.007	95.88 \pm 0.005
I ² BGNN-A	<u>92.13\pm0.039</u>	91.10 \pm 0.042	<u>92.41\pm0.033</u>	82.24 \pm 0.056	79.52 \pm 0.124	80.20 \pm 0.117	89.64 \pm 0.021	91.63 \pm 0.021	88.76 \pm 0.032	95.94 \pm 0.005	95.93 \pm 0.005	96.07 \pm 0.004
I ² BGNN-T	91.02 \pm 0.084	<u>91.72\pm0.034</u>	90.13 \pm 0.085	<u>82.64\pm0.077</u>	82.13 \pm 0.101	<u>83.84\pm0.062</u>	89.28 \pm 0.025	<u>91.87\pm0.022</u>	89.87 \pm 0.028	95.99 \pm 0.004	95.98 \pm 0.005	96.11 \pm 0.004
Ethident (w/o GC)	93.02 \pm 0.029	<u>93.36\pm0.032</u>	<u>94.38\pm0.028</u>	85.62 \pm 0.060	83.68 \pm 0.080	84.91 \pm 0.051	92.28 \pm 0.027	92.77 \pm 0.027	92.39 \pm 0.021	97.79 \pm 0.003	97.37 \pm 0.004	97.80 \pm 0.003
Ethident	94.05\pm0.034	92.76 \pm 0.038	94.05 \pm 0.033	86.38\pm0.049	87.00\pm0.040	85.30\pm0.057	93.16\pm0.021	93.55\pm0.027	93.34\pm0.022	97.93\pm0.002	97.58\pm0.004	97.98\pm0.003
OPS.	11	7	5	2	4	16	6	11	5	6	7	10

V. EXPERIMENTS

A. Data Preparation

We intercept the first 10 million block data (the time interval is between “2015-07-03” to “2020-05-04”) from the Xblock website³ [38]. Within this time interval, we can extract in total 309,010,831 transactions and 175,351,541 contract calls, involving 90,193,755 EOA and 16,221,914 CA. Account identity labels are obtained from Label Word Cloud in Ethereum blockchain browser⁴, including 73 *ICO-wallet*, 65 *Mining*, 193 *Exchange* and 2,535 *Phish/Hack*.

These four types of accounts are prevalent on blockchain platforms, and have received widespread attention. It is of sufficient practical significance to identify whether an account belongs to these types, especially for phish and hack accounts. For each type of identity label (*ICO-wallet*, *Mining*, *Exchange* and *Phish/Hack*), we sample all target account subgraphs with this label as the positive sample, as well as the same number of randomly sampled account subgraphs with other labels as the negative sample. We perform subgraph sampling for each labeled account according to different edge information (**Amount**, **Times** or **avgAmount**), yielding three types of datasets whose names are suffixed with “-A”, “-T”, “-aA”, respectively. Table III shows the specifications of subgraph datasets sampled from lw-AIG with $h = 2$ and $K = 20$.

B. Comparison Methods

To illustrate the effectiveness of our *Ethident* on account identification, we compare with three broad categories of methods: manual feature engineering, graph embedding methods and GNN-based methods.

For manual feature engineering that is the most common and simplest method for account identification, we design 16 manual features for Ethereum accounts according to the prior knowledge and the characteristics of raw Ethereum data, as detailedly described in Appendix A, yielding account embeddings with dimension size of 16. For graph embedding methods, we consider DeepWalk [26], Node2Vec [27], Struc2Vec [39], Trans2Vec [28] and Graph2Vec [30] for account embedding. For the above two categories of methods, we achieve account identification by feeding the generated account embeddings into three kinds of machine learning classifiers: Logistic Regression (LR), Random Forest (RF) and LightGBM (LGBM).

For GNN-based methods, we first compare with three commonly used GNNs: GCN [40], GAT [34], and GIN [41], which are adjusted for subgraph classification by following with a pooling layer and a prediction head. We also compare with previous related work for account identification: I²BGNN, which achieves account identification based on different edge information, yielding two variants: I²BGNN-A and I²BGNN-T.

C. Experimental Settings

For subgraph sampling in *Ethident*, we set the subgraph hop h to 2 and sample $K = 20$ neighbors per hop. For

³http://xblock.pro/

⁴https://etherscan.io/labelcloud

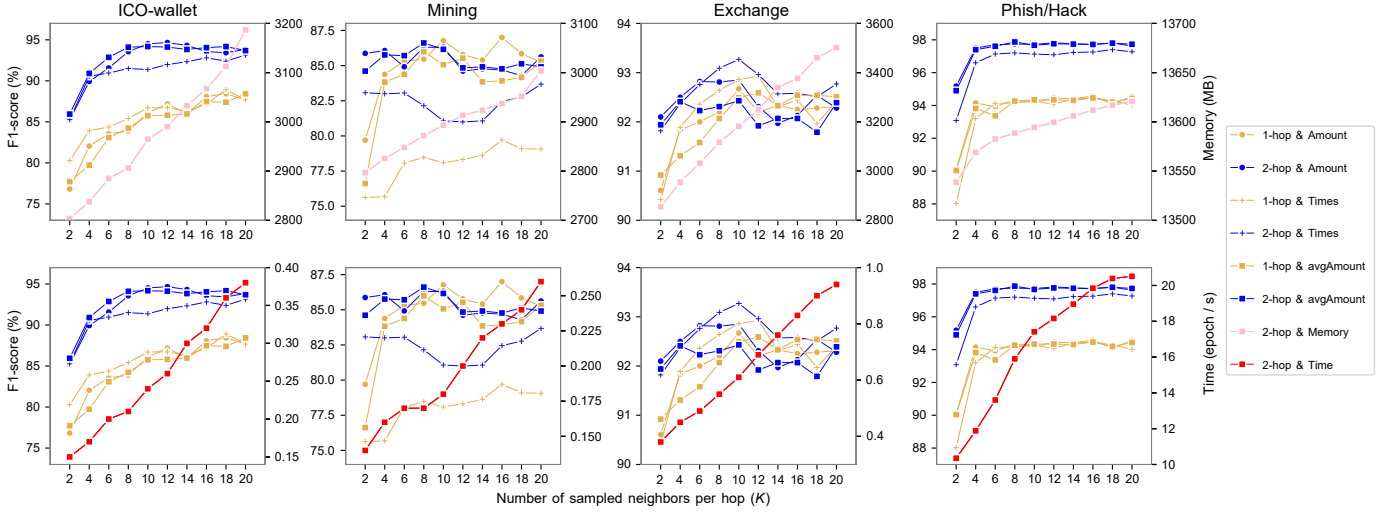


Fig. 6. Impact of sampling scale on performance and consumption (memory and time).

graph augmentation, we set the probability \mathcal{P} to 10%. For the encoder *HGATE*, we stack $k = 2$ graph attention layers with the hidden dimension of 128 for account embedding, and use global max pooling for initial subgraph embedding. In addition, the projection head f_ϕ is a two-layer perceptron with Relu activation and linear skip connection, and the prediction head f_ψ is a two-layer perceptron with Relu and Softmax activation. We set the temperature parameter τ and trade-off coefficient λ to 0.2 and 0.01, respectively.

For GCN, GAT, GIN and I^2 BGNN, the number of the corresponding message passing layers are 2, 2, 5 and 2 respectively. The global max pooling is used for final subgraph embedding. For all GNN-based methods, we set the embedding dimension, batch size N , learning rate, dropout to 128, 32, 0.001, 0.2, respectively. During model training, we use early stopping with patience of 20.

For DeepWalk, Node2Vec, Struc2Vec and Trans2Vec, we set the length of walks to 20, the number of walks to 40, and the context size to 3. For Node2Vec, we set the return parameter p and in-out parameter q to 0.25 and 0.4, respectively. For the above four random walk-based methods which are extremely inefficient on large-scale graphs, we generate a training graph by sampling the connected subgraph containing all target accounts and their partial 2-hop neighbors from lw-AIG. For Graph2Vec, we set the number of Weisfeiler-Lehman iterations to 2, the downsampling frequency to 0.0001, the minimal count of graph feature occurrences to 5, the epoch to 500 and the learning rate to 0.025. For the above graph embedding methods, we set the dimension of output account embedding to 128.

For each subgraph dataset sampled from lw-AIG, we split it into training, validation and testing sets with a proportion of 1:1:1, repeat 3-fold cross validation 10 times and report the average micro-F1 score as well as standard deviation.

D. Evaluation on Account Identification

We evaluate our *Ethident* on account identification and the results are presented in Table IV, from which we can observe

that our *Ethident* achieves state-of-the-art results with respect to comparison methods. Specifically, our *Ethident* significantly outperforms manual feature engineering and graph embedding methods across all datasets, and yields 2.09% ~ 18.27% relative improvement over best baselines in terms of F1-score, indicating that the learned subgraph features are better at capturing the behavior patterns of accounts than manual or shallow topology features. When compared to GNN-based methods, our *Ethident* surpasses strong baselines: we observe 1.13% ~ 4.93% relative improvement over best baselines.

These observations meet our intuition. As we can see, the performance of manual features and graph embedding methods varies largely across different datasets with comparatively lower performance rankings. This is consistent with our assertion that they have limited expressiveness for different kinds of account subgraphs. Because manual features and graph embedding methods cannot learn task-related features in an end-to-end manner, they rely heavily on the choice of classifiers to achieve relatively high performance. Meanwhile, classic GNN baselines normally surpass the manual features and graph embedding methods since they learn simultaneously from both graph topology and latent features. Nevertheless, these baselines like GCN and GAT disregard the important edge information and generate subgraph-level features via naive pooling operations. The two variants of I^2 BGNN only consider one single interaction information and disregard others. Combining with the above analysis, we know that our *Ethident* learns from both node and edge information associated with behavior patterns and identities of accounts, and uses a hierarchical attention mechanism to effectively characterize node-level account features and subgraph-level behavior patterns, reasonably achieving superior performance on account identification.

E. Pattern Analysis in Micro Interaction Subgraphs

After evaluating the overall performance of our method, we investigate the behavior patterns of different accounts using experimental results on micro interaction subgraphs.

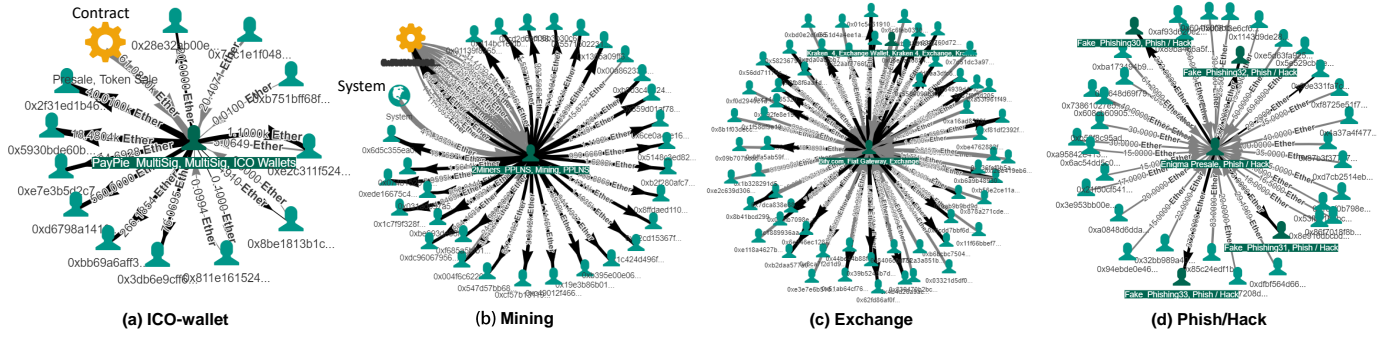


Fig. 7. Different categories of accounts generally have different behavior patterns, as embodied in their micro interaction subgraphs. Here we present some interesting accounts to help explanation.

Furthermore, we list the following **Observations** as well as explainable analysis.

1) **Obs. 1. Larger subgraphs generally contain more critical identity-related pattern information:** We analyze the impact of subgraph scale by evaluating our encoder on subgraph datasets with different scale settings. Specifically, we vary h in $\{1, 2\}$ and K in $\{2, 4, \dots, 20\}$. We observe that it is generally better to infer from 2-hop subgraphs than 1-hop ones, especially for *ICO-wallet* and *Phish/Hack*, judging from Fig. 6. Meanwhile, as the size of subgraphs increases, the performance becomes better first and then remains stable or fluctuates slightly in most cases. The above phenomenon suggests that subgraphs with larger scale benefit account identification more, meeting our intuition that larger subgraphs generally contain more critical pattern information associated with account identities.

2) **Obs. 2. Different subgraph information highlights the behavior patterns of accounts with different contributions:** For each category of accounts, the performance of all methods except manual features varies largely across the datasets with different sampling strategies. And we count the number of optimal performances obtained by all methods under different sampling strategies, yielding the *OPS*. values. Judging from the bottom row in Table IV, we have reasonable explanations for such phenomenon that different sampling strategies benefit differently for account identification.

As we know, different categories of accounts have different behavior patterns that are embodied in their micro interaction subgraphs. Here we present some interaction subgraphs of real accounts to help explain, as shown in Fig. 7. Note that only the cumulative transaction amount of Ether is displayed between any two connected nodes in the interaction subgraphs.

- **ICO-wallet:** Initial Coin Offering (ICO) is a financing method that raises funds for blockchain projects by issuing tokens. ICO projects usually pre-sell tokens in exchange for a large amount of Ether, and after a period, the projects will give supporters a certain return on their investment. The key behavior pattern is represented as a large number of outgoing edges with a certain **Amount** of investment rewards from the center ICO account to the surrounding supporters. Since investment actions generally involve a higher transaction amount, sampling interaction subgraphs according to **Amount** information can maximally preserve

the behavior pattern of ICO accounts.

- **Mining:** Mining pooling is a cooperative mining team that shares computational power to find blocks. The mining pool will receive a large amount of mining rewards issued by the system, and distribute them to subordinate miners according to the proof-of-work (PoW) consensus protocol. The key behavior pattern is represented as a large number of outgoing edges with a certain amount of cumulative rewards from the center mining pool to the surrounding miner nodes. Since the block reward of Ethereum is fixed for a period, miners in the same mining pool generally have a relatively stable average mining income, which inspires us to use the average amount (**avgAmount**) information to guide the sampling of interactive subgraphs.
- **Exchange:** The exchange is a platform that provides users with asset transaction matching and clearing services. Exchange accounts usually interact frequently with their clients to process a large number of transaction orders, and behave as hub nodes with extremely high centrality (i.e., large in-degree and out-degree) in the interaction graphs. So sampling interaction subgraphs according to **Times** information benefits more.
- **Phish/Hack:** Both *Phishers* and *Hackers* engage in illegal fraud activities, in which they usually spread a large number of websites, emails or links containing viruses, Trojans, unwanted software, etc., and trick the recipient into doing remittances directly or providing the sensitive information of system privileges. As shown in Fig. 7(d), the center phish/hack account receives large amounts of Ether through various scams and disperses them to other phish/hack accounts for concealment. The key behavior pattern has one explicit characteristic: bi-directional edges (mutual transactions) between the center node and surrounding ones are rare, and the center node has high in-degree and low out-degree. Since illegal frauds such as fake token exchange or ransomware often set a specific threshold amount or fixed ransom which can be reflected in the **average amount** information, using subgraphs sampled according to **avgAmount** information may benefit more for identifying *Phish/Hack* accounts.

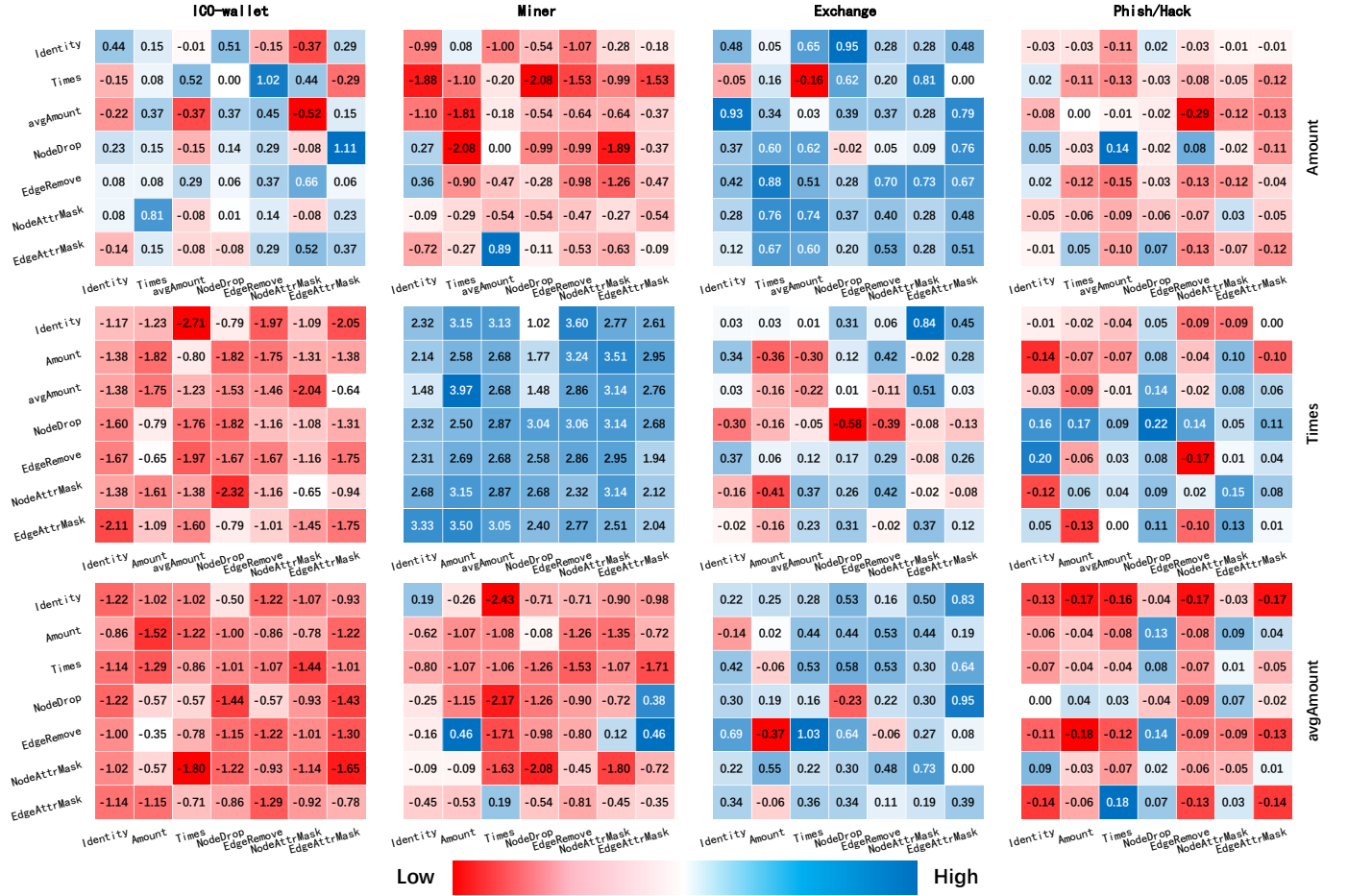


Fig. 8. Account Identification F1-score gain (%) when contrasting different augmentation pairs, compared to Ethident (w/o GC) which stands for a no-augmentation version of our framework, under all datasets. “Identity” represents the original view.

F. Effect of Subgraph Contrastive Learning

We further investigate the effectiveness of subgraph contrast in our *Ethident*, and list several **Observations** as well as explainable analysis.

1) **Obs. 3. Graph augmentation is crucial, and structure-level augmentation seems to benefit more:** We first apply various pairs of augmentation views to all datasets, as illustrated in Fig. 8, and obtain the performance gain of *Ethident* compared with *Ethident* (w/o GC) which stands for a no-augmentation version of our framework (i.e., identifying accounts by using our encoder and a followed prediction head). Overall, it seems more likely to yield positive gain by using either “NodeDrop” or “EdgeRemove” as one of the augmented views, when compared to other augmentation pairs. In addition, for exchange accounts that frequently call various contracts, there will be more non-zero values in their node features, making attribute masking an effective augmentation strategy as well. Finally, we note that the combination of various augmentation views is sensitive to subgraph datasets with different sampling strategies, i.e., the performance gain of our *Ethident* with the same augmentation pairs varies largely across datasets with different sampling strategies, which encourages adaptive selections of sampling strategies and augmentation combinations in future work.

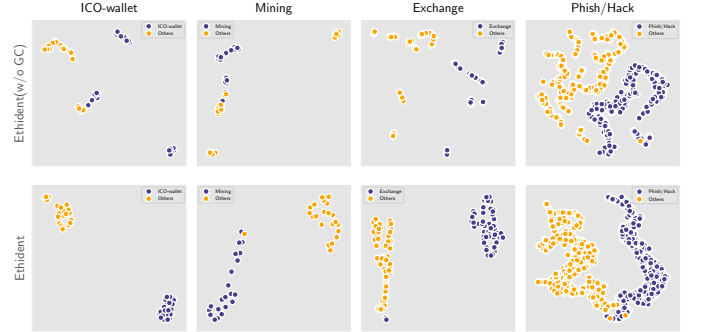


Fig. 9. The UMAP visualization of the subgraph embeddings learned by *Ethident* with and without subgraph contrast.

2) **Obs. 4. Contrastive self-supervision improves the generalization of model in account feature learning:** We utilize the UMAP [42] to visualize the subgraph embeddings learnt by *Ethident* and *Ethident* (w/o GC) in Fig. 9, where different colors mean different labels. Compared with *Ethident* (w/o GC) which only uses prediction loss $\mathcal{L}_{\text{pred}}$, more obvious inter-class separability and intra-class compactness are achieved after applying the contrastive constraint $\mathcal{L}_{\text{self}}$, which illustrates its effectiveness on learning the behavior pattern differences.

TABLE V
COMPARISON OF ENCODER PERFORMANCE WITH AND WITHOUT
SUBGRAPH-LEVEL ATTENTIVE POOLING.

Dataset	Method	Sampling Strategy		
		Amount	Times	averAmount
Eth-ICO	<i>HGATE</i> (w/o AttPooling) ¹	92.81±0.034	93.30±0.039	93.57±0.034
	<i>HGATE</i>	93.02±0.029	93.36±0.032	94.38±0.028
	gain	0.23%	0.06%	0.87%
Eth-Mining	<i>HGATE</i> (w/o AttPooling)	86.21±0.055	84.38±0.072	85.38±0.030
	<i>HGATE</i>	85.62±0.060	83.68±0.080	84.91±0.051
	gain	-0.68%	-0.83%	-0.55%
Eth-Exchange	<i>HGATE</i> (w/o AttPooling)	91.27±0.026	91.12±0.027	90.98±0.023
	<i>HGATE</i>	92.28±0.027	92.77±0.027	92.39±0.021
	gain	1.11%	1.81%	1.55%
Eth-PhishHack	<i>HGATE</i> (w/o AttPooling)	96.81±0.006	96.15±0.009	96.72±0.005
	<i>HGATE</i>	97.79±0.003	97.37±0.004	97.80±0.003
	gain	1.01%	1.27%	1.12%

¹ AttPooling: subgraph-level attentive pooling operation.

TABLE VI
RESULTS OF ACCOUNT IDENTIFICATION ON EOSIO.

Method	EOSIO		
	$h = 1, K = 10$	$h = 1, K = 20$	$h = 2, K = 10$
GCN	99.59±0.003	99.47±0.003	99.30±0.004
GAT	99.52±0.002	99.68±0.002	99.12±0.004
GIN	99.52±0.002	99.55±0.001	99.40±0.002
I ² BGNN-A	99.47±0.002	99.53±0.002	99.12±0.005
I ² BGNN-T	99.25±0.002	99.62±0.002	99.17±0.005
Ethident (w/o GC)	99.58±0.003	99.70±0.002	99.20±0.005
Ethident	99.75±0.001	99.75±0.001	99.47±0.002

Moreover, *Ethident* separates different patterns with relatively clearer boundaries, suggesting that contrastive self-supervision can effectively improve the generalization of models when training with scarce labels.

G. More Analysis

1) *Impact of Subgraph-level Attentive Pooling*: To illustrate the effectiveness of subgraph-level attentive pooling, we compare the performance of our encoder with and without this module, as reported in Table V. We observe that the encoder with *AttPooling* achieves better performance on 3 out of 4 categories of accounts, validating the effectiveness of our proposal. For the exception that *AttPooling* brings a negative gain on *Mining* subgraphs, we speculate that a mining pool organization usually behaves very differently from an individual miner whose transaction behavior has no significant relationship with the mining pool, so aggregating information from neighbors may interfere with the characterization of mining pools' behavior patterns.

2) *Impact of Perturbation Probability*: We continue to analyze the impact of perturbation probability \mathcal{P} in data augmentation. We use a view for data augmentation (in the form of "Identity & DA") and vary \mathcal{P} in $\{0.1, 0.2, \dots, 0.5\}$, the results are shown in Fig. 10. Combined with the statistics in Table III, we have drawn the following conclusions: (1) Datasets with larger amounts of samples are more robust to variation in perturbation probability; (2) Datasets with larger average sample scales (in terms of Avg. $|N|$ and Avg. $|V|$) are less sensitive to variation in perturbation probability; (3) An effective and reasonable selection interval for the perturbation probability could be $[0.1, 0.3]$. Finally, we observe that our

TABLE VII
RESULTS ON THE ORIGINAL AND MALFUNCTIONING EXCHANGE &
PHISHING DATASETS.

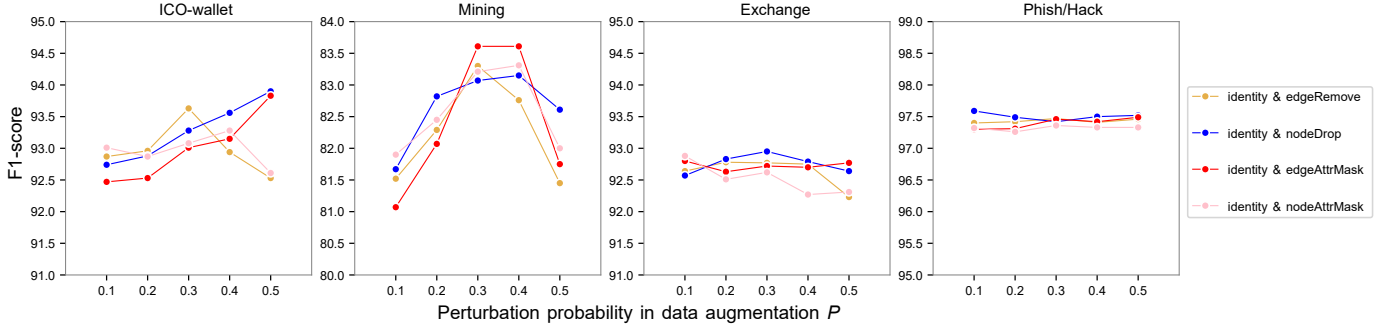
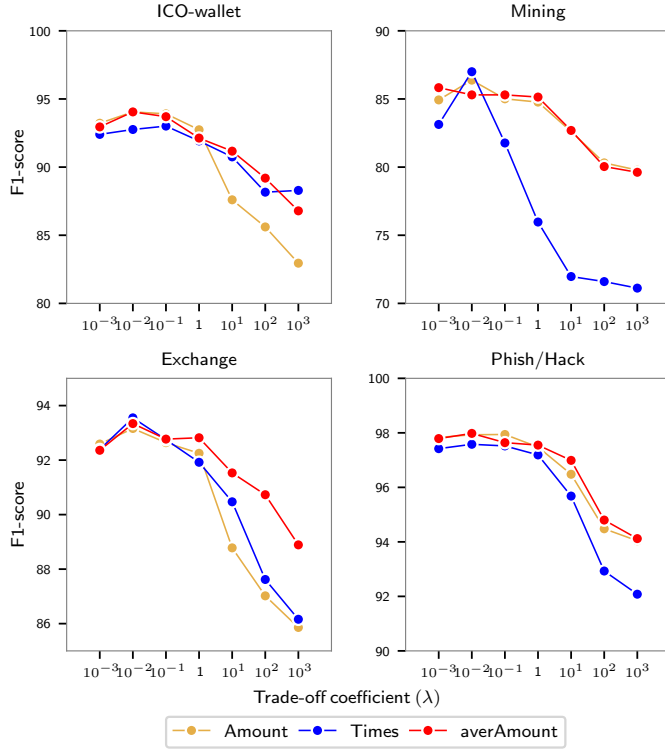
Sample Strategy	Test with D_{test}^{ori} or D_{test}^{mal}	Method			
		Ethident (w/o GC)	edgeRemove & identity	edgeRemove & edgeRemove	edgeRemove & nodeDrop
averAmount	Original	89.74±0.026	90.57±0.031	91.22±0.029	91.74±0.025
	Malfunctioning	89.56±0.038	91.32±0.029	91.24±0.031	91.22±0.030
	Loss	-0.20%	0.83%	0.02%	-0.57%
Amount	Original	89.09±0.035	90.70±0.033	89.39±0.039	90.86±0.024
	Malfunctioning	89.28±0.034	90.13±0.027	89.92±0.030	90.96±0.030
	Loss	0.21%	-0.63%	0.59%	0.11%
Times	Original	86.97±0.029	87.92±0.029	87.51±0.027	88.83±0.023
	Malfunctioning	87.21±0.034	87.40±0.035	87.48±0.028	88.33±0.031
	Loss	0.28%	-0.59%	-0.03%	-0.56%

method still performs well when the perturbation is large, which is likely to benefit from the attentive graph pooling.

3) *Impact of Loss Tradeoff*: Here we analyze the impact of the trade-off coefficient λ which controls the contribution of subgraph contrast. As we can see from Fig. 11, our *Ethident* achieves relatively better performance when λ is less than 1, which meets our intuition. We treat subgraph contrast as a pretext task or a regularization to subgraph classification. When the coefficient of regularization is greater than 1, the classification task cannot be fully optimized, failing to learn the task-related features.

4) *Tradeoffs between Performance and Consumption*: Since the subgraph extraction allows for mini-batch training of our framework, greatly reducing computational consumption and time cost. Here we further investigate the tradeoffs between performance and consumption under different sampling scales, as shown in Fig. 6. Since the performance of 2-hop subgraphs significantly outperforms that of 1-hop subgraphs, we just draw the consumption curves of 2-hop subgraph. We can first observe that the memory and time consumption increase almost linearly with the sample scale, while the performance converges when the sample scale increases to a certain extent. We then use the " $\frac{\text{performance}}{\text{consumption}}$ " metric to roughly analyze the tradeoff between them. Note that a larger " $\frac{\text{performance}}{\text{consumption}}$ " metric generally indicates better performance with less consumption. After observation and calculation, we finally conclude that an appropriate parameter setting of sample scale could be $h = 2$ and $K \in [8, 10]$.

5) *Generalization Application to Other Cryptocurrency*: To verify the generalization of our framework on other cryptocurrencies, we collect transaction data of another on-chain cryptocurrency EOSIO and deploy related experiments. We first construct an account interaction graph including 944,865 nodes and 10,435,037 edges, in which the node features ($\mathbf{X} \in \mathbb{R}^{n \times 1216}$) are constructed by the contract calling information and account name restriction mechanism, and the edge features are constructed in the same way as in Sec. III-C1. We then collect 2000 target accounts, half of which are normal accounts and half are bot accounts, and our goal is to determine the identity of these accounts, that is, normal accounts or bot accounts. We apply our *Ethident* framework to achieve the account identification on the EOSIO dataset, and the experimental settings are similar to that in Sec. V-C. Table VI report the account identification results on EOSIO

Fig. 10. Impact of perturbation probability in data augmentation (P).Fig. 11. Impact of tradeoff coefficient (λ).

dataset. As we can see, our *Ethident* still achieves the state-of-the-art identification performance on the EOSIO dataset, showing a good generalization to other cryptocurrencies.

6) *Generalization Evaluation on Malfunctioning Exchange Accounts*: Accounts in the same broad category share common transaction patterns, but also have their own particularities, so that they can generally still be classified at a more fine-grained level. For example, the malfunctioning exchange accounts which have different transaction patterns from normal ones are common in real Ethereum, and still belong to ‘Exchange’ accounts. However, these malfunctioning exchange accounts which only receive amount from trader accounts but do not send amount back to traders may be more likely to be misclassified as a ‘Phishing’ account rather than an ‘Exchange’ account, as it has a more similar transaction pattern to the former. Here, we conduct experiments to validate whether our *Ethident* model can successfully identify the malfunctioning exchange account

as an ‘Exchange’ account instead of a ‘Phishing’ account.

- Randomly select the same number of ‘Exchange’ accounts and ‘Phishing’ accounts, and extract their transaction subgraphs, yielding a new dataset of 386 account subgraphs. Split the new dataset into training D_{train} , validation D_{val} and testing D_{test}^{ori} sets with a proportion of 1 : 1 : 1.
- Generate the malfunctioning exchange account by removing the target exchange account’s outgoing edges from account subgraph. In this way, we yield the malfunctioning testing set D_{test}^{mal} containing phishing accounts and malfunctioning exchange accounts.
- Train *Ethident* model to determine whether an account is an ‘Exchange’ account or a ‘Phishing’ account using D_{train} and D_{val} .
- Evaluate the performance of the model in determining whether an account is an ‘Exchange’ account or a ‘Phishing’ account using D_{test}^{ori} and D_{test}^{mal} respectively.

As we can see from Table VII, our *Ethident* models still achieve powerful performance in identifying indistinguishable ‘Phishing’ accounts and ‘Malfunctioning Exchange’ accounts. Compared with the original results in D_{test}^{ori} , our *Ethident* show $-0.63\% \sim 0.83\%$ performance fluctuations in malfunctioning testing set D_{test}^{mal} , which is a rational and normal performance jitter. This phenomenon suggests that our model has almost no performance loss in distinguishing between phishing accounts and malfunctioning exchange accounts, showing strong robustness and generalization.

VI. CONCLUSION

Financial security has become a top priority in the blockchain ecosystem. This paper provides a new perspective on account de-anonymization, and proposes a behavior-aware Ethereum account identification framework that integrates hierarchical graph attention and self-supervision mechanism, to effectively characterize the behavior patterns of different accounts. Extensive experiments on Ethereum datasets demonstrate the superiority of our framework in terms of state-of-the-art performance and powerful generalization. Furthermore, our framework also has a good transferability to other blockchain platforms like Bitcoin and EOSIO, which will be discussed in future work.

TABLE VIII
STATISTICS OF THE AVERAGE OF MANUAL FEATURE FOR VARIOUS ACCOUNTS IN ETHEREUM.

Manual Features	Phish-Hack	Exchange	Mining	ICO-Wallets	Common	Definition
<i>active_days</i>	76.94	703.35	595.98	547.84	14.49	the number of active days of the account.
<i>total_received</i>	110.84	1551629.77	5470.67	6642.11	245.38	the total amount of Ether received by the account.
<i>num_received_tx</i>	27	88490.39	68.55	279.86	4.13	the number of transactions with Ether received by the account.
<i>inter_acct_received</i>	23.29	29985.26	13.6	218.77	0.4	the number of accounts sending Ether to the target account.
<i>total_output</i>	124.03	2309107.24	367339.66	33824.02	370.21	the amount of Ether spent by the account.
<i>num_output_tx</i>	29.91	88285.11	818877.92	62.84	4.56	the number of transactions that the account has spent Ether.
<i>inter_acct_output</i>	8.18	46692.69	16825.85	37.53	1.48	the number of accounts receive Ether from the target account.
<i>avg_received</i>	29.52	2002.55	49.79	854.35	7.06	the average amount of Ether received by the account.
<i>avg_received_day</i>	11.66	1748.8	6.55	11.36	4.93	the average amount of Ether received by the account per day.
<i>avg_received_tx_day</i>	1.51	113.5	0.16	0.49	0.03	the number of transactions with Ether received by the account per day.
<i>avg_output</i>	30.58	1453.75	249.95	4399.06	9.46	the average amount of Ether spent by the account.
<i>avg_output_day</i>	14.66	2213.77	389.08	80.3	5.39	the average amount of Ether spent by the account.
<i>avg_output_tx_day</i>	0.66	101.26	633.21	0.2	0.03	the average number of transactions with Ether spent by the account per day.
<i>times_contract_called</i>	11.68	66682.01	61002.03	690.93	1.52	the number of times the account calls the smart contract.
<i>times_contract_called_day</i>	0.5	82.66	41.84	1.07	0.02	the number of times the account calls the smart contract per day.
<i>num_contract_called</i>	3.29	2031.6	1256.31	3.26	0.13	the number of contracts called by the account.

ACKNOWLEDGMENTS

This work was partially supported by the Key R&D Program of Zhejiang under Grant 2022C01018, by the National Key R&D Program of China under Grant 2020YFB1006104, by the National Natural Science Foundation of China under Grant 61973273, by the Zhejiang Provincial Natural Science Foundation of China under Grant LR19F030001, and by the Major Key Project of PCL under Grants PCL2022A03, PCL2021A02 and PCL2021A09.

APPENDIX A MANUAL FEATURE DETAILS

Manual feature engineering is the most common and simplest way for account identification. According to the characteristics of raw Ethereum data and prior knowledge, we design 16 manual features for Ethereum accounts, as shown in Table VIII.

REFERENCES

- [1] M. H. Miraz and M. Ali, "Applications of blockchain technology beyond cryptocurrency," *Annals of Emerging Technologies in Computing (AETiC)*, vol. 2, no. 1, 2018.
- [2] K. Fanning and D. P. Centers, "Blockchain and its coming impact on financial services," *Journal of Corporate Accounting & Finance*, vol. 27, no. 5, pp. 53–57, 2016.
- [3] G. Blossey, J. Eisenhardt, and G. Hahn, "Blockchain technology in supply chain management: An application perspective," in *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.
- [4] T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He, "Blockchain in healthcare applications: Research challenges and opportunities," *Journal of Network and Computer Applications*, vol. 135, pp. 62–75, 2019.
- [5] R. Klusman and T. Dijkhuizen, "Deanonymisation in ethereum using existing methods for bitcoin," 2018.
- [6] K. Toyoda, T. Ohtsuki, and P. T. Mathiopoulos, "Multi-class bitcoin-enabled service identification based on transaction history summarization," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1153–1160.
- [7] Y.-J. Lin, P.-W. Wu, C.-H. Hsu, I.-P. Tu, and S.-w. Liao, "An evaluation of bitcoin address classification based on transaction history summarization," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 302–310.
- [8] M. Bartoletti, B. Pes, and S. Serusi, "Data mining for detecting bitcoin ponzi schemes," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 75–84.
- [9] Y. Huang, H. Wang, L. Wu, G. Tyson, X. Luo, R. Zhang, X. Liu, G. Huang, and X. Jiang, "Understanding (mis) behavior on the cosio blockchain," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 2, pp. 1–28, 2020.
- [10] Y. Li, Y. Cai, H. Tian, G. Xue, and Z. Zheng, "Identifying illicit addresses in bitcoin network," in *International Conference on Blockchain and Trustworthy Systems*. Springer, 2020, pp. 99–111.
- [11] Q. Yuan, B. Huang, J. Zhang, J. Wu, H. Zhang, and X. Zhang, "Detecting phishing scams on ethereum based on transaction records," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.
- [12] J. Shen, J. Zhou, Y. Xie, S. Yu, and Q. Xuan, "Identity inference on blockchain using graph neural network," in *International Conference on Blockchain and Trustworthy Systems*. Springer, 2021, pp. 3–17.
- [13] L. Chen, J. Peng, Y. Liu, J. Li, F. Xie, and Z. Zheng, "Phishing scams detection in ethereum transaction network," *ACM Transactions on Internet Technology (TOIT)*, vol. 21, no. 1, pp. 1–16, 2020.
- [14] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: Characterizing payments among men with no names," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, 2013, pp. 127–140.
- [15] M. Spagnuolo, F. Maggi, and S. Zanero, "Bitiodino: Extracting intelligence from the bitcoin network," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 457–468.
- [16] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and Privacy in Social Networks*. Springer, 2013, pp. 197–223.
- [17] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 34–51.
- [18] M. Harrigan and C. Fretter, "The unreasonable effectiveness of address clustering," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE, 2016, pp. 368–373.
- [19] C. Remy, B. Rym, and L. Matthieu, "Tracking bitcoin users activity using community detection on a network of weak signals," in *International Conference on Complex Networks and Their Applications*. Springer, 2017, pp. 166–177.
- [20] M. Lischke and B. Fabian, "Analyzing the bitcoin network: the first four years," *Future Internet*, vol. 8, no. 1, p. 7, 2016.
- [21] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.
- [22] F. Victor, "Address clustering heuristics for ethereum," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 617–633.
- [23] S. Linoy, N. Stakhanova, and S. Ray, "De-anonymizing ethereum blockchain smart contracts through code attribution," *International Journal of Network Management*, vol. 31, no. 1, p. e2130, 2021.
- [24] M. Jourdan, S. Blandin, L. Wynter, and P. Deshpande, "Characterizing

entities in the bitcoin blockchain,” in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2018, pp. 55–62.

- [25] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” *Advances in Neural Information Processing Systems 10*, vol. 30, pp. 3146–3154, 2017.
- [26] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.
- [27] A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 855–864.
- [28] J. Wu, Q. Yuan, D. Lin, W. You, W. Chen, C. Chen, and Z. Zheng, “Who are the phishers? phishing scam detection on ethereum via network embedding,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.
- [29] Z. Yuan, Q. Yuan, and J. Wu, “Phishing detection on ethereum via learning representation of transaction subgraphs,” in *International Conference on Blockchain and Trustworthy Systems*. Springer, 2020, pp. 178–191.
- [30] N. Annamalai, C. Mahinthan, V. Rajasekar, C. Lihui, L. Yang, and J. Shantanu, “Graph2vec: Learning distributed representations of graphs,” in *Proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG)*, 2017.
- [31] Q. Xuan, J. Wang, M. Zhao, J. Yuan, C. Fu, Z. Ruan, and G. Chen, “Subgraph networks with application to structural feature space expansion,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 6, pp. 2776–2789, 2019.
- [32] S. Phetsouvanh, F. Oggier, and A. Datta, “Egret: Extortion graph exploration techniques in the bitcoin network,” in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2018, pp. 244–251.
- [33] R. Zhang, G. Zhang, L. Liu, C. Wang, and S. Wan, “Anomaly detection in bitcoin information networks with multi-constrained meta path,” *Journal of Systems Architecture*, vol. 110, p. 101829, 2020.
- [34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” *Proceedings of the 6th International Conference on Learning Representations*, 2018.
- [35] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, vol. 33, 2020, pp. 5812–5823.
- [36] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi, “Nodeaug: Semi-supervised node classification with data augmentation,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020, pp. 207–217.
- [37] J. Zhou, J. Shen, and Q. Xuan, “Data augmentation for graph classification,” in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2341–2344.
- [38] P. Zheng, Z. Zheng, J. Wu, and H.-n. Dai, “Xblock-eth: Extracting and exploring blockchain data from ethereum,” *IEEE Open Journal of the Computer Society*, vol. 1, pp. 95–106, 2020.
- [39] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, “Struc2vec: Learning node representations from structural identity,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 385–394.
- [40] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *Proceedings of the 5th International Conference on Learning Representations*, 2017.
- [41] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in *Proceedings of the 6th International Conference on Learning Representations*, 2018.
- [42] L. McInnes, J. Healy, N. Saul, and L. Grossberger, “Umap: Uniform manifold approximation and projection,” *The Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.



Jiajun Zhou received the BS degree in automation from the Zhejiang University of Technology, Hangzhou, China, in 2018, where he is currently pursuing the Ph.D degree in control theory and engineering with the College of Information and Engineering. His current research interests include graph data mining and deep learning, especially for graph self-supervised learning and blockchain data analytics.



Chenkai Hu is currently pursuing the bachelor's degree in automation at Zhejiang University of Technology, Hangzhou, China. His current research interests include data mining in blockchain.



Jianlei Chi received the B.S. degree in computer science and technology from Harbin Engineering University, China, 2014, and the Ph.D. degree in computer science and technology in 2022 from Xi'an Jiaotong University, China. He is currently an assistant professor at Hangzhou Research Institute of Xidian University. His research interests include trustworthy software, software engineering, program analysis and machine learning.



Jiajing Wu (Senior Member, IEEE) received the Ph.D. degree from The Hong Kong Polytechnic University, Hong Kong, in 2014. In 2015, she joined Sun Yat-sen University, Guangzhou, China, where she is currently an Associate Professor. Her research interests include blockchain, graph mining, and network science. Dr. Wu was awarded the Hong Kong Ph.D. Fellowship Scheme during her Ph.D. degree in Hong Kong from 2010 to 2014. She also serves as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS.



Meng Shen (Member, IEEE) received the B.Eng. degree in computer science from Shandong University, Jinan, China, in 2009, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2014. He is currently an Associate Professor with Beijing Institute of Technology, Beijing. He has authored over 50 papers in top-level journals and conferences, such as ACM SIGCOMM, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC), and IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY (TIFS).

His research interests include data privacy and security, blockchain applications, and encrypted traffic classification. He received the Best Paper Award from IEEE/ACM IWQoS 2021. He was selected by the Beijing Nova Program 2020 and the winner of the ACM SIGCOMM China Rising Star Award in 2019. He has guest edited Special Issues on Emerging Technologies for Data Security and Privacy in IEEE Network and IEEE INTERNET OF THINGS JOURNAL.



Qi Xuan (M'18) received the BS and PhD degrees in control theory and engineering from Zhejiang University, Hangzhou, China, in 2003 and 2008, respectively. He was a Post-Doctoral Researcher with the Department of Information Science and Electronic Engineering, Zhejiang University, from 2008 to 2010, respectively, and a Research Assistant with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, in 2010 and 2017. From 2012 to 2014, he was a Post-Doctoral Fellow with the Department of Computer Science,

University of California at Davis, CA, USA. He is a senior member of the IEEE and is currently a Professor with the Institute of Cyberspace Security, College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. His current research interests include network science, graph data mining, cyberspace security, machine learning, and computer vision.