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 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 con-
sider identifying Ethereum accounts from a graph classification
perspective, and propose an end-to-end graph neural network
framework named EthIdent, to characterize the behavior pat-
terns 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 con-
trastive 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

I. INTRODUCTION

The past few years have witnessed the application of
blockchain technology in new technological and indus-
trial revolutions, such as cryptocurrency [1], financial ser-
ices [2], supply chain management [3], healthcare [4], etc.
As a distributed data storage technology, blockchain is decen-
tralized, traceable and tamper-proof, which guarantees the
fidelity and security of data recording and generates trust
without a third-party notarization. Benefiting from these char-
acteristics, 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 CoinMarketCap, 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 transac-
tions, 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 (DAapps) 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.

1https://coincapmarket.com/
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 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], [7], [8], [9] and graph analytics [10], [11], [12], [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


| Symbol | Definition |
|--------|------------|
| $G, g$ | Graph, subgraph. |
| $v, V, E$ | Node (account), node set, edge set. |
| $N(v_i)$ | 1-hop neighbor set of node $v_i$. |
| $x, X$ | Node feature, node feature matrix. |
| $e$ | Edge feature. |
| $y, Y$ | Account identity label, label set. |
| $h$ | Hidden representation of node feature. |
| $s, g$ | Subgraph representation before/after attentive pooling. |
| $z$ | Projection representation of subgraph. |
| $J_{\theta}, f_{\psi}, f_{\phi}$ | Encoder, prediction head, projection head. |
| $\alpha, \beta$ | Normalized attention scores. |
| $a$ | Unnormalized attention scores. |
| $\Theta$ | 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. |
| $\lambda$ | Parameter: trade-off hyper-parameter in loss functions. |

**TABLE I
MAIN SYMBOLS USED IN THIS PAPER**

- **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.

### 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], [15], [16], [17], [18], [19], [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. Androulaki 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] and [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

\(^2\)Data and code are available at https://github.com/jjzhou012/EthIdent
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 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 $G = (V, E, X, Y)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes or accounts, $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V\}$ is the set of edges, $X \in \mathbb{R}^{n \times f_1}$ is the node feature matrix, and $E \in \mathbb{R}^{m \times f_2}$ is the edge feature matrix (we assume, $|E| = m$). We use $Y = \{(v_i, y) \mid v_i \in V\}$ 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.

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 $G = (V_{eo}, V_{ca}, E_t, E_c, Y)$, where $V_{eo}$ and $V_{ca}$ are the set of EOA and CA respectively, $E_t = \{(v_i, v_j, d, w) \mid v_i, v_j \in V_{eo}\}$ is the directed edge set constructed from transaction information, and $E_c = \{(v_i, v_j, f) \mid v_i \in V_{eo} \cup V_{ca}, v_j \in V_{ca}\}$ 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 \( Y = \{ (v_i, y_i) \mid v_i \in V_{eo} \} \).

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 \( G = (V_{eoa}, E_t, X, E, Y) \), where \( E_t = \{ (v_i, v_j, t, \tilde{w}) \mid v_i, v_j \in V_{eoa} \} \), \( X \) is the node feature matrix constructed from contract call information and \( E \) 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 \( v_j \).

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 \( e_{ij} = [t, \tilde{w}] \).

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 \( X \in \mathbb{R}^{n \times F} \) to represent the preference for contract call, as formulated below:

\[
X = [x_1; \cdots; x_i; \cdots; x_n]^T,
\]

\[
x_i = [t_1, \cdots, t_j, \ldots, t_F],
\]

where \( t_j = \begin{cases} t & \text{If there are } t \text{ calls to } v_j^{call}; \\
0 & \text{If there is no call to } v_j^{call}; \end{cases} \).

Note that \( x_i \) is the feature of \( v_i^{call} \). 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 \( v_t \), 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 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 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: \( \text{Amount}(\tilde{w}) \), \( \text{Times}(t) \) or average Amount (\( \text{avgAmount}, \tilde{w}/t \)). Specifically, for a target account node \( v_l \), 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:

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

where \( V_k \) is the set of nodes sampled at hop \( k \) and \( V_0 = \{v_l\} \), \( \mathcal{N}_v \) 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 \( \text{topK} \) is the function that returns the top-K most important nodes. After \( h \) iterations, we obtain the top set \( V_h = \bigcup_{k=0}^{h} V_k \) sampled from lw-AIG, and the subgraph \( g_i \) of target account \( v_l \) can be induced by \( V_l \) 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: \( D = \{ (g_i, y_i) \mid (v_l, y_i) \in Y \} \). 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 Ethidient 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 \( f_0 \).

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 \( v_l \) has its neighbor features that concatenate both neighboring account features (\( \mathbf{x}_j \)) and the connecting interaction features (\( \mathbf{e}_{lj} \)), represented as \( [\mathbf{x}_j \parallel \mathbf{e}_{lj}] \). 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 \( \mathbf{x}_j \in \mathbb{R}^F \) and its neighbor features \( [\mathbf{x}_j \parallel \mathbf{e}_{lj}] \in \mathbb{R}^{F+2} \) 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 \( \Theta_a \) as follows, and generates aligned embeddings for neighbors of the target account.

\[
\tilde{\mathbf{x}}_j = \text{LeakyRelu} \left( \Theta_a \cdot [\mathbf{x}_j \parallel \mathbf{e}_{lj}] \right),
\]

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 \( v_l \), if \( v_a \) has many high-volume transactions with \( v_l \) while \( v_b \) has only one low-volume transaction with \( v_l \), or if \( v_a \) has a more similar preference of contract call with \( v_l \) than \( v_b \), then \( v_a \) often plays a more important role in identifying \( v_l \) since it preserves more information associated with the identity of \( v_l \). 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 \( v_l \) in the input subgraph \( g \), the node-level attention mechanism learns the contribution attention scores for its neighbors \( v_j \), as follows:

\[
a_{lj} = \text{LeakyRelu} \left( \Theta'_g \cdot [\mathbf{h}'_j \parallel \mathbf{h}_j] \right),
\]

where a linear transformation parameterized by \( \Theta'_g \) and a nonlinear LeakyRelu activation are performed together to compute the importance of account \( v_j \)'s hidden features to account \( v_l \) 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:

\[
a_{lj} = \text{Softmax}(a_{lj}) = \frac{\exp(a_{lj})}{\sum_{i \in \mathcal{N}(v_l) \cup \{v_l\}} \exp(a_{lj})},
\]

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

$$h_i^{l+1} = \text{Elu} \left( a_i^{l+1} \cdot \Theta_s^{l+1} \cdot h_i^l + \sum_{j \in N(i)} a_{ij}^{l+1} \cdot \Theta_s^{l+1} \cdot h_j^l \right),$$

where a linear transformation parameterized by $\Theta_s^{l+1}$ and a nonlinear $\text{Elu}$ 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 $h_i^0$ 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 $h_i^0 = x_i$, and that of its neighboring accounts $v_j$ is $h_j^0 = x_j$. To better characterize accounts, the stack performs an iterative process of transferring, transforming, aggregating and updating the representation from interactive neighbors. After $k$ iterations, the final output account embeddings $h^k_i$ 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 $s$ by using global max pooling over all account embeddings in the subgraph:

$$s = \text{MaxPooling} \left( h^k \right).$$

Note that the input of the $\text{MaxPooling}$ layer is the final account embeddings $h^k$ generated in Sec. IV-B.2. To better characterize the subgraph pattern, we update $s$ by aggregating features of all accounts with different contributions (attentions). In other words, for the initial subgraph embedding $s$, we use an attention mechanism to learn the contribution attention score for arbitrary account $v_j$ in the subgraph as follows:

$$a_j = \text{LeakyRelu} \left( \Theta_s \cdot \left| s \parallel h_j^k \right\| \right),$$

where a linear transformation parameterized by $\Theta_s$ and a nonlinear $\text{LeakyRelu}$ activation are performed to compute the importance of account $v_j$’s hidden features to the initial subgraph embedding $s$. Same as the node-level attention,
a softmax function is applied to compute the normalized attention scores:
\[
\beta_j = \text{Softmax}(a_j) = \frac{\exp(a_j)}{\sum_{i \in V_g} \exp(a_i)},
\]
where \( V_g \) is the node set of subgraph \( g \), and \( a_i \) is the self-attention score of \( s \). Finally, the attentive pooling performs the update process as follows:
\[
g = \text{Elu} \left( \beta_s \cdot \Theta_\beta \cdot s + \sum_{j \in V_g} \beta_j \cdot \Theta_\beta \cdot h^g_j \right),
\]
where a linear transformation parameterized by \( \Theta_\beta \) and a nonlinear Elu activation are used to compute the final subgraph embedding \( g \) 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 \([35], [36], [37]\). 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 \( p \) to be dropped from subgraph.
  - **Edge Removing**: Each edge has a certain probability \( p \) to be removed from subgraph.

- **Attribute-level Augmentation**
  - **Node Attribute Masking**: Each dimension of node features has a certain probability \( p \) to be set as zero.
  - **Edge Attribute Masking**: Each dimension of edge features has a certain probability \( p \) to be set as zero.

- **Sampling-based Augmentation**

Since each subgraph is sampled from Iw-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 \( \hat{g}_1, \hat{g}_2 \) for each target account subgraph \( g_t \), and assign the identity label of target account to them as a pseudo label:
\[
\begin{align*}
D_{\text{aug}1} &= \left\{ (\hat{g}_1^1, y) \mid \hat{g}_1^1 = T_1(g_t); (o_i, y_i) \in Y \right\}, \\
D_{\text{aug}2} &= \left\{ (\hat{g}_1^2, y) \mid \hat{g}_1^2 = T_2(g_t); (o_i, y_i) \in Y \right\}.
\end{align*}
\]

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 \( g_t \), its two correlated views \( \hat{g}_1^1 \) and \( \hat{g}_1^2 \) are generated by undergoing two augmentation operators \( T_1 \) and \( T_2 \), where \( \hat{g}_1^1 = T_1(g_t) \) and \( \hat{g}_1^2 = T_2(g_t) \). The correlated augmented views are fed into the encoder \( f_\theta \), producing the whole subgraph representations \( \hat{g}_1^1 \) and \( \hat{g}_1^2 \). Then they are mapped into an embedding space for contrast via a projection head \( \phi \), yielding \( \hat{z}_1^1 \) and \( \hat{z}_1^2 \). Note that \( \theta \) and \( \phi \) 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:
\[
\mathcal{L}_\text{self} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i,
\]
where \( N \) is the number of subgraphs in a batch (i.e., batch size). The loss for each subgraph can be computed as:
\[
\mathcal{L}_i = - \log \frac{e^{\cos(z_1^i, \hat{z}_1^i)/\tau}}{\sum_{j=1, j \neq i}^{N} e^{\cos(z_1^i, \hat{z}_1^j)/\tau}},
\]
where \( \cos(\cdot, \cdot) \) is the cosine similarity function with \( \cos(z_1^i, \hat{z}_1^j) = z_1^i \top \hat{z}_1^j / \|z_1^i\| \|\hat{z}_1^j\| \) and \( \tau \) is the temperature parameter. The two correlated views \( z_1^i \) and \( \hat{z}_1^j \) of account subgraph \( g_t \) 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 \( f_\psi \), which maps the subgraph representations to labels reflecting
account identity, yielding a classification loss:

\[ \mathcal{L}_{\text{pred}} = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log (f_{\psi} (g_i)), \]

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 \( \text{HGATE} \) 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 formalized below:

\[ \mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda \cdot \mathcal{L}_{\text{self}}, \]

where \( \lambda \) is a trade-off hyper-parameter controls the contribution of the self-supervision term.

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 website\(^3\) [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 browser,\(^4\) 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: I\(^2\)BGGN, which achieves account identification based on different edge information, yielding two variants: I\(^2\)BGGN-A and I\(^2\)BGGN-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 graph augmentation, we set the probability \( P \) to 10%. For the encoder \( \text{HGATE} \), 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 \( f_{\psi} \) is a two-layer perceptron with Relu activation and linear skip connection, and the prediction head \( f_{\gamma} \) is a two-layer perceptron with Relu and Softmax activation. We set the temperature parameter \( r \) and trade-off coefficient \( \lambda \) to 0.2 and 0.01, respectively.

For GCN, GAT, GIN and I\(^2\)BGGN, 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
### Table IV

| Method                  | Dataset (with different sampling strategy) | Outperform Manual Feature Engineering | Outperform GNN-Based Methods | E. Pattern Analysis in Micro Interaction Subgraphs |
|-------------------------|-------------------------------------------|--------------------------------------|-------------------------------|---------------------------------------------------|
|                        |                                            |                                      |                               |                                                   |
|                        |                                            |                                      |                               |                                                   |
| To comparison methods.  |                                           |                                      |                               |                                                   |
| Specifically, our Ethident significantly outperforms manual feature engineering and graph embedding methods across all datasets, and yields 2.09% \sim 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% \sim 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 1BGNN 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. 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 $\{2, 4, \ldots, 20\}$. 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 interesting 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
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.

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 \( L_{\text{pred}} \), more obvious inter-class separability and intra-class compactness are achieved after applying the contrastive constraint \( L_{\text{self}} \), which illustrates its effectiveness on learning the behavior pattern differences. Moreover, Ethident separates different patterns with relatively clearer boundaries, suggesting that contrastive self-supervision...
fig. 9. The UMAP visualization of the subgraph embeddings learned by Ethident with and without subgraph contract.

### TABLE V

| Dataset      | Method            | Amount | Time  | Sample Strategy | Encoding | Sampling Strategy | Encoding |
|--------------|-------------------|--------|-------|-----------------|----------|-------------------|----------|
| ETH-ICO      | HGate (w/o AttPooling) | 90.31±0.003 | 90.60±0.003 | ETH-ICO      | HGate    | ETH-ICO            | HGate    |
| ETH-Mining   | HGate (w/o AttPooling) | 90.31±0.003 | 90.60±0.003 | ETH-Mining   | HGate    | ETH-Mining         | HGate    |
| ETH-Exchange | HGate (w/o AttPooling) | 90.31±0.003 | 90.60±0.003 | ETH-Exchange | HGate    | ETH-Exchange       | HGate    |
| ETH-PHilHerK | HGate (w/o AttPooling) | 90.31±0.003 | 90.60±0.003 | ETH-PHilHerK | HGate    | ETH-PHilHerK       | HGate    |

### TABLE VI

| Method      | h = 1, K = 10 | h = 3, K = 100 | h = 1, K = 10 | h = 3, K = 100 |
|-------------|---------------|----------------|---------------|----------------|
| GCN         | 99.59±0.003   | 99.47±0.003   | 99.30±0.004   | 99.20±0.005   |
| GAT         | 99.52±0.002   | 99.68±0.002   | 99.12±0.004   | 99.01±0.005   |
| GIN         | 99.52±0.002   | 99.55±0.001   | 99.40±0.002   | 99.31±0.003   |
| P2BGNN-A    | 99.47±0.002   | 99.53±0.002   | 99.12±0.005   | 99.10±0.006   |
| P2BGNN-T    | 99.26±0.002   | 99.62±0.002   | 99.17±0.005   | 99.15±0.006   |
| Ethident (w/o GC) | 99.58±0.003 | 99.70±0.002 | 99.20±0.005 | 99.20±0.005 |

| Method      | h = 1, K = 10 | h = 3, K = 100 | h = 1, K = 10 | h = 3, K = 100 |
|-------------|---------------|----------------|---------------|----------------|
| Ethident    | 99.75±0.001   | 99.75±0.001   | 99.47±0.002   | 99.47±0.002   |

### TABLE VII

| Method      | Test with Dval or Dtest | Ethident | edgeRemove = 0.005 | edgeRemove = 0.01 | edgeRemove = 0.02 |
|-------------|-------------------------|----------|--------------------|--------------------|--------------------|
| Original    | 98.34±0.002             | 98.34±0.002 | 98.34±0.002 | 98.34±0.002 | 98.34±0.002 |
| Malfunctioning | 98.34±0.002             | 98.34±0.002 | 98.34±0.002 | 98.34±0.002 | 98.34±0.002 |
| Loss        | 0.00%                   | 0.00%     | 0.00%              | 0.00%              | 0.00%              |
| Loss        | 0.00%                   | 0.00%     | 0.00%              | 0.00%              | 0.00%              |
| Loss        | 0.00%                   | 0.00%     | 0.00%              | 0.00%              | 0.00%              |

1. AttPooling: subgraph-level attentive pooling operation.

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 $P$ in data augmentation. We use a view for data augmentation (in the form of “Identity & DA”) and vary $P$ in $\{0.1, 0.2, \ldots, 0.5\}$, 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 $\text{Avg.} |N|$ and $\text{Avg.} |V|$) 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 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 $\lambda$ which controls the contribution of subgraph contrast. As we can see from Fig. 11, our Ethident achieves relatively better performance when $\lambda$ 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 Cryptocurrencies:** 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 ($X \in \mathbb{R}^{n \times 1216}$) 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-C.1. 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 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 $D_{ori}$ 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 $D_{mal}$ 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 $D_{ori}$ and $D_{mal}$ 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 $D_{ori}$, our Ethident show -0.63% $\sim$ 0.83% performance fluctuations in malfunctioning testing set $D_{mal}$, 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.

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.

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

| Manual Features | Phish-Hack | Exchange | Mining | IGO-Wallets | Common | Definition |
|----------------|-----------|----------|--------|-------------|--------|------------|
| active_days    | 76.94     | 703.35   | 595.98 | 547.84      | 14.49  | the number of active days of the account. |
| total_received | 110.84    | 1551629.77 | 5470.67 | 6642.11   | 245.38 | the total amount of Ether received by the account. |
| num_transactions | 27     | 88495.39 | 68.55 | 279.86 | 4.13 | the number of transactions with Ether received by the account. |
| inter_activity  | 23.29    | 29985.26 | 13.6  | 218.77 | 0.4  | the number of accounts sending Ether to the target account. |
| total_output   | 124.03   | 2309107.24 | 367339.66 | 33824.02 | 370.21 | the amount of Ether spent by the account. |
| num_transactions | 29.91   | 88285.11 | 81877.92 | 62.84    | 4.56  | the number of transactions that the account has spent Ether. |
| inter_activity  | 8.18     | 46692.69 | 16825.85 | 37.53 | 1.48 | the number of accounts receive Ether from the target account. |
| avg_revenue_day | 29.52   | 2002.53 | 49.79 | 854.35 | 7.06 | the average amount of Ether received by the account. |
| avg_revenue_day | 11.66   | 1748.88 | 6.35 | 11.36 | 4.93 | the average amount of Ether received by the account per day. |
| avg_revenue_day | 1.51    | 113.5   | 0.16  | 0.49 | 0.03 | the number of transactions with Ether received by the account per day. |
| avg_output     | 30.58    | 14535.75 | 249.95 | 4399.06 | 9.46 | the average amount of Ether spent by the account. |
| avg_output_day | 14.66    | 2233.77 | 389.08 | 80.3 | 5.39 | the average amount of Ether spent by the account per day. |
| avg_output_day | 0.66     | 101.26  | 633.21 | 0.2  | 0.03 | the average amount of transactions with Ether spent by the account per day. |
| times_contacted | 11.68   | 66882.01 | 61002.03 | 690.93 | 1.52 | the number of times the account calls the smart contract. |
| times_contacted | 0.5      | 82.66   | 41.84 | 1.07 | 0.02 | the number of times the account calls the smart contract per day. |
| times_contacted | 3.29     | 2031.6  | 1256.31 | 3.26 | 0.13 | the number of contracts called by the account. |

REFERENCES

[1] M. H. Miraz and M. Ali, “Applications of blockchain technology beyond cryptography,” Ann. Emerg. Technol. Comput., vol. 2, no. 1, pp. 1–6, Jan. 2018.
[2] K. Fanning and D. P. Centers, “Blockchain and its coming impact on financial services,” J. 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 Proc. Annul. Hawaii Int. Conf. Syst. Sci., 2019, pp. 1–9.
[4] T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He, “Blockchain in healthcare applications: Research challenges and opportunities,” J. Netw. Comput. Appl., vol. 135, pp. 62–75, Jun. 2019.
[5] R. Kusman and T. Dijkhuizen, “Deanonimisation in Ethereum using existing methods for Bitcoin,” Syst. Netw. Lab. (SNE-Lab), Univ. Amsterdam, Amsterdam, The Netherlands, Tech. Rep. 61, 2018.
[6] K. Toyoda, T. Ohtsuki, and M. Spagnuolo, “Bitiodine: Extracting intelligence from the Bitcoin network,” in Proc. Fast unfolding of communities in large networks,” in Proc. Int. Conf. Financial Cryptogr. Data Secur. Berlin, Germany: Springer, 2014, pp. 457–468.
[7] F. Reid and M. Harrigan, “An analysis of anonymity in the Bitcoin system,” in Security and Privacy in Social Networks. New York, NY: USA: Springer, 2013, pp. 197–223.
[8] E. Androulaki, G. O. Karama, M. Roeschlin, T. Scherer, and S. Capkun, “Evaluating user privacy in Bitcoin,” in Proc. Int. Conf. Financial Cryptogr. Data Secur. Berlin, Germany: Springer, 2013, pp. 34–51.
[9] M. Harrigan and C. Fetter, “The unreasonable effectiveness of address clustering,” in Proc. Intl IEEE Conf. Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People, Smart World Congr. (UIC/ATC/ScalCom/CBDCom/IoT/SmartWorld), Jul. 2016, pp. 368–373.
[10] C. Remy, B. Rym, and L. Matthieu, “Tracking Bitcoin users activity using community detection on a network of weak signals,” in Proc. Int. Conf. Complex Netw. Their Appl. Cham, Switzerland: Springer, 2017, pp. 166–177.
[11] M. Lischke and B. Fabian, “Analyzing the Bitcoin network: The first four years,” Future Internet, vol. 8, no. 4, p. 7, Mar. 2016.
[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” J. Stat. Mech., Theory Exp., vol. 2008, no. 10, Oct. 2008, Art. no. P10008.
[13] F. Victor, “Address clustering heuristics for Ethereum,” in Proc. Int. Conf. Financial Cryptogr. Data Secur. Cham, Switzerland: Springer, 2020, pp. 617–632.
[14] S. Linoy, N. Stakanova, and S. Ray, “De-anonymizing Ethereum blockchain smart contracts through code attribution,” Int. J. Netw. Manage., vol. 31, no. 1, p. e2130, Jan. 2021.
[15] M. Jourdan, S. Blandin, L. Wnyter, and P. Deshpande, “Characterizing entities in the Bitcoin blockchain,” in Proc. IEEE Intl. Conf. Data Mining Workshops (ICDMW), Nov. 2014, pp. 55–62.
[16] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017, pp. 3146–3154.
[17] B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online learning of social representations,” in Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2014, pp. 701–710.
[18] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2016, pp. 855–864.
[28] J. Wu et al., “Who are the phishers? Phishing scam detection on Ethereum via network embedding,” IEEE Trans. Syst., Man, Cybern. Syst., vol. 52, no. 2, pp. 1156–1166, Feb. 2020.

[29] Z. Yuan, Q. Yuan, and J. Wu, “Phishing detection on Ethereum via learning representation of transaction subgraphs,” in Proc. Int. Conf. Blockchain Trustworthy Syst. Singapore: Springer, 2020, pp. 178–191.

[30] N. Annamalai, C. Mahinthan, V. Rajaeekar, C. Lihui, L. Yang, and J. Shantana, “graph2vec: Learning distributed representations of graphs,” in Proc. 13th Int. Workshop Mining Learn. Graphs (MLG), 2017, pp. 1–8.

[31] Q. Xuan et al., “Subgraph networks with application to structural feature space expansion,” IEEE Trans. Knowl. Data Eng., vol. 33, no. 6, pp. 2776–2789, Jun. 2019.

[32] S. Pfsisouvans, F. Ogjier, and A. Datta, “EGRET: Extortion graph exploration techniques in the Bitcoin network,” in Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW), Nov. 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,” J. Syst. Archit., vol. 110, Nov. 2020, Art. no. 101829.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in Proc. 6th Int. Conf. Learn. Represent., 2018, pp. 1–12.

[35] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” in Proc. 34nd Int. Conf. Neural Inf. Process. Syst., 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 Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2020, pp. 207–217.

[37] J. Zhou, J. Shen, and Q. Xuan, “Data augmentation for graph classification,” in Proc. 29th ACM Int. Conf. Inf. Knowl. Manage., Oct. 2020, pp. 2341–2344.

[38] P. Zheng, Z. Zheng, J. Wu, and H.-N. Dai, “XBlock-EHT: Extracting and exploring blockchain data from Ethereum,” IEEE Open J. Comput. Soc., vol. 1, pp. 95–106, 2020.

[39] L. F. Ribeiro, P. H. P. Saverese, and D. R. Figueiredo, “struc2vec: Learning node representations from structural identity,” in Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2017, pp. 385–394.

[40] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. 5th Int. Conf. Learn. Represent., 2017, pp. 1–14.

[41] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in Proc. 6th Int. Conf. Learn. Represent., 2018, pp. 1–17.

[42] L. McInnes, J. Healy, N. Saul, and L. Großberger, “UMAP: Uniform manifold approximation and projection,” J. Open Source Softw., vol. 3, no. 29, p. 861, Sep. 2018.

Jiajun Zhou received the B.S. 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.

Jianlei Chi received the B.S. degree in computer science and technology from Harbin Engineering University, China, in 2014, and the Ph.D. degree in computer science and technology from Xi’an Jiaotong University, China, in 2022. He is currently an Assistant Professor with the Hangzhou Research Institute, 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. She 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.

Qi Xuan (Senior Member, IEEE) received the B.S. and Ph.D. 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, 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 of the Department of Computer Science, University of California at Davis, CA, USA. He is currently a Professor with the Institute of Cyberspace Security, College of Information Engineering, Zhejiang University of Technology, Hangzhou. His current research interests include network science, graph data mining, cyberspace security, machine learning, and computer vision.