ScamSweeper: Detecting Illegal Accounts in Web3 Scams via Transactions Analysis

Xiaoqi Li, Wenkai Li, Zhijie Liu, Meikang Qiu, Zhiquan Liu, Sen Nie, Zongwei Li, Shi Wu, Yuqing Zhang

***Abstract*—The web3 applications have recently been growing, especially on the Ethereum platform, starting to become the target of scammers. The web3 scams, imitating the services provided by legitimate platforms, mimic regular activity to deceive users. However, previous studies have primarily concentrated on de-anonymization and phishing nodes, neglecting the distinctive features of web3 scams. Moreover, the current phishing account detection tools utilize graph learning or sampling algorithms to obtain graph features. However, large-scale transaction networks with temporal attributes conform to a power-law distribution, posing challenges in detecting web3 scams. To overcome these challenges, we present ScamSweeper, a *novel* framework that emphasizes the dynamic evolution of transaction graphs, to identify web3 scams on Ethereum. ScamSweeper samples the network with a structure temporal random walk, which is an optimized sample walking method that considers both temporal attributes and structural information. Then, the directed graph encoder generates the features of each subgraph during different temporal intervals, sorting as a sequence. Moreover, a variational Transformer is utilized to extract the dynamic evolution in the subgraph sequence. Furthermore, we collect a large-scale transaction dataset consisting of web3 scams, phishing, and normal accounts, which are from the first 18 million block heights on Ethereum. Subsequently, we comprehensively analyze the distinctions in various attributes, including nodes, edges, and degree distribution. Our experiments indicate that ScamSweeper outperforms SIEGE, Ethident, and PDTGA in detecting web3 scams, achieving a weighted F1-score improvement of at least 17.29% with the base value of 0.59. In addition, ScamSweeper in phishing node detection achieves at least a 17.5% improvement over DGTSG and BERT4ETH in F1-score from 0.80.**

***Index Terms*—Web3 scam, Deep learning, Malicious account, Transaction analysis**

I. Introduction

Detecting malicious accounts on the blockchain networks [2], [3] is a significant area of research [4], [5]. Due to its high transaction volume and anonymity, the Ethereum blockchain [2] has been a prime target for fraudulent activities [6], [7], [8]. Numerous studies have focused on identifying phishing accounts to safeguard users from scams [9], [10], [11], [12], [13], [14], [15], [16]. However, with the emergence of Ethereum, web3 scams have also emerged, where certain services engage in covert malicious activities within the blockchain [17], [18]. For example, Venom Drainer offered web3 services that generated over $27 million in profits from about 15,000 victims [19]. These scams attract users and generate profits by concealing specific malicious activities within legitimate services, such as concealing scams in the NFT airdrops [20].

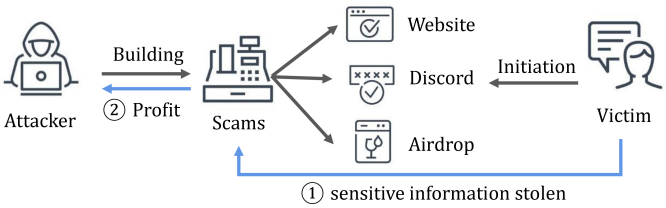


Fig. 1: A Motivating Example of Web3 scams on the Ethereum. The black line represents the activities under the chain, and the blue line indicates the behaviors on the chain.

Web3 scams refer to fraudulent or deceptive practices that exploit the network technologies, platforms, or services in Web3 [21], [22]. Web3 scams can take various forms, such as Ponzi and Phishing. Web3 scams encompass phishing, however, there also exist discernible differences. As depicted in Fig. 1, under the chain, traditional phishing is similar to web3 scams, which trick users into connecting their wallets to fake mediums such as websites and Discord. On the chain, traditional phishing accounts defraud users' funds. However, the web3 scams pretend to offer regular services while allowing the attacker to gain access to the user's funds surreptitiously. Subsequently, attackers could steal tokens without the user's involvement and obscure their traces [1]. Thus, the primary disparity lies in the mimic behavior and track obscuration (e.g., utilizing mixer contracts or multiple transfers).

In this paper, we focus on the multiple transfer behaviors performed by fraudsters to obscure traces. Thus, we analyze the external transactions related to the web3 scams. Previous works [4], [11], [12], [13], [16], [23] have converted the Ethereum transactions into a multi-edge graph, which can intuitively represent the transaction patterns, helping identify malicious patterns. They attempt to learn node feature representations from the power-law distributed and temporal-attribute topology network graph, identifying malicious nodes

Xiaoqi Li, Wenkai Li, and Zongwei Li are with Hainan University, Haikou, 570228, China (e-mail: csxqli@ieee.org, cswkli@hainanu.edu.cn, lizw1017@hainanu.edu.cn)

Zhijie Liu is with Ohio State University, Columbus, OH, USA (e-mail: liu.13062@osu.edu)

Meikang Qiu is with the College of Computer and Cyber Science, Augusta University, Augusta, GA, USA (e-mail: qiumeikang@yahoo.com)

Zhiquan Liu is with the Jinan University, Guangzhou, 510632, China (e-mail: zqliu@jnu.edu.cn)

Sen Nie and Shi Wu are with the Keen Security Lab, Tencent, Shanghai, China (e-mail: {snie, shiwu}@tencent.com)

Yuqing Zhang is with the University of Chinese Academy of Sciences, Beijing, 100049, China (e-mail: zhangyq@nipc.org.cn)

Wenkai Li (cswkli@hainanu.edu.cn) is the corresponding author.

This manuscript is an extended version of our work [1]. It has been extended more than 80% over the ASE conference version, including: (1) Optimization of the contribution and motivation of the paper (Sec. I & Sec. III). (2) Elaboration on the detailed principles of the framework (Sec. IV). (3) Enhancement of the analysis of the experiments (Sec. V). (4) Addition of discussion with the scams detected by our framework (Sec. VI).

or behaviors on the Ethereum network. However, limited research focuses on detecting malicious accounts or behaviors in web3 scams, and there are three significant challenges associated with graph learning methods for web3 scams.

• **Challenge 1 (C1):** ***Large-scale Graph Construction:*** The transactions on Ethereum have grown to 2,762 million as of April 2025 [24], [25], which poses a significant challenge for deep learning methods that require processing the entire data. The majority of graph construction methods employ simple random sampling techniques, disregarding the attributes within the network [26], [12], [16] (e.g., transaction sequence, top-k transaction graphs, random walk).

• **Challenge 2 (C2):** ***Temporal Series:*** The transaction within the network is inherent in time dependency, as evidenced by factors such as fund flow, and invocation. Learning the dynamic evolution in chronological order is crucial, given that interactions between malicious and victim nodes follow a specific sequence. However, most studies [11], [14], [16], [27] only consider the network structures, neglecting the temporal series between the network structures.

• **Challenge 3 (C3):** ***Detection in the Web3 Domain:*** Although numerous detection tools for malicious accounts exist [26], [11], [12], [14], [16], only a few studies have addressed the scams in the web3 domain. Furthermore, there is a lack of research summarizing the current state of scams on web3, distinguishing them from other types of fraud nodes (e.g., phishing nodes), which hampers the development of malicious account detection in the specific domain of web3.

Our Solutions. To address these challenges, we introduce a temporal subgraph learning framework, called *ScamSweeper*, to detect malicious accounts in web3 scams. ScamSweeper considers not only the graph structure but also the dynamic evolution feature between subgraphs.

• **Solution to C1:** We take an optimized sample walking method to extract subgraphs from transactions, constructing subgraphs with temporal distribution. Each subnetwork is expanded from the walking sequences, where each node enriches a structure window-size edges, constructing directed graphs as a sampled dataset (§ IV-C[1]).

• **Solution to C2:** We partition each graph by temporal intervals to obtain multiple subgraphs and construct graphs with timestamps. We sort the subgraphs as a sequence to capture the dynamic evolution, thus gaining insights into the transaction time series (§ IV-E).

• **Solution to C3:** We observe the differences between malicious and normal nodes, particularly in the in- and out-degree distributions. Thus, we create directed graphs to facilitate the graph feature learning and capture the dynamic evolution of subgraphs constructed by the transactions between temporal intervals (§ IV-D).

The main contributions of this paper are as follows:

- To the best of our knowledge, we are the *first* to introduce the temporal subgraph sequence learning framework for identifying web3 scam accounts on Ethereum. We slice subgraphs in sequences via temporal intervals, analyzing the dynamic evolution of malicious nodes (§ IV).
- We enhance the sample walking method by integrating temporal attributes and structural information to extract subnetworks from extensive network data. The sampled subnetworks construct a directed graph, acquiring insights into the patterns of transaction graphs (§ IV-C).
- We collect and analyze a large-scale dataset on web3 scams, phishing, and normal accounts. We evaluate ScamSweeper through the dataset, achieving a weighted F1-score of 17.29% higher than the state-of-the-art in web3 scam accounts and an F1-score of 17.5% higher in phishing accounts (§ V).
- We open-source our codes and experiment data at https://figshare.com/s/65d00a4c50c9d5188c06.

[1]In this paper, we use the symbol § to represent the meaning of Section, for example, § IV-C means the subsection C in Section IV.

II. Related Work

In this section, we mainly introduce previous works on detecting malicious blockchain nodes. There have been various works on phishing scam account detection.

A. Phishing Account Detection

Several classification methods [28] have been used in studies [29], [30], [14], [31], [32], [26], [11], [12], [21], [23], [5], [33], [4] to improve the performance of phishing scam detection. Some works [33], [4] leverage off-chain data to detect phishing scams, providing novel insights into domain information and social networks on social media. Yang et al. [23] analyze NFT phishing for the first time and summarize four phishing patterns and their corresponding behaviors. Chen et al. [29] extracted 219 statistical features from the first- and second-order neighbors in the first 7 million blocks, utilizing the DElightGBM method to discover that the variance of the node's in-degree was the most influential factor in phishing node classification. S. Ratra et al. [5] and Farrugia et al. [30] collect features of the graph network using GNN and statistics, then utilize the XGBoost and AdaBoost methods to classify phishing accounts. Moreover, the time difference in the network and the total transaction amount are ultimately recognized as the two most critical features [30]. Poursafaei et al. [14] utilized the Ri-walk approach, combining node structural features, neighbor features, and transaction network features as node features, and employed logistic regression for classification after integrating these features. Lastly, Yuan et al. [31] employed Node2Vec to sample the network, learn node representations from the neighborhood, and then leverage the SVM algorithm for classification. Chen et al. [32] utilized statistical and structural features extracted from deep learning models to identify phishing nodes. They employed a graph convolutional network to learn the network structures and collected eight types of transaction features. Hu et al. [26] proposed a novel Transformer structure to detect malicious nodes. They embedded address information, location information, node type, degree information, transaction amount, and time information in the node-generated transaction sequences. The Transformer encoder was then used to generate the node embeddings for identification. Li et al. [11] proposed an

incremental self-supervised learning method called SIEGE, which learned the spatial node features by observing the growth in the number of new neighboring nodes. They then focused on learning multiple graph features in temporal order and used both temporal and spatial features as node features. Li et al. [12] used temporal information by using GCN to understand the structural features of the graph and convert edge information into nodes. They trained on transactions and considered structural, statistical, and temporal features to identify phishing nodes. Different from the above studies, ScamSweeper samples the transaction network based on structural and temporal attributes, thereby reducing the computational resource consumption for large-scale transaction graphs.

B. Web3 Scam Detection

Regarding detecting scams on Web3, it can be divided into the smart contract and node levels.

Smart contracts are a logical structure on the blockchain that can deploy logical operating code. web3 scams, such as honeypots [34], [35], Ponzi schemes [36], and rug pull [37], can be implemented through smart contracts, leading to various detection methods. The n-gram model is leveraged to divide the bytecode into several groups, and then the LightGBM model [34] is used to learn the features of the bytecode to detect honeypots. In addition, the attention mechanism [35] is used for focusing the contributions of the groups and making a classification to detect honeypots and Ponzi schemes. Zheng et al. [36] employed a multi-view cascade ensemble model to detect Ponzi schemes. They combined the features of the contract bytecode, opcode, and deployer to achieve classification. Huang et al. [37] detected rug pull [38] by employing multi-dimensional features four days prior, including time series data, token transfer operations, and market trading features.

Node-level scam detection primarily focuses on identifying behavioral patterns in transactions or call operations. Kong et al. [39] implemented taint analysis technology to analyze possible execution flows and observed stored state information in call data to detect vulnerabilities related to price manipulation. Varun et al. [40] extracted features such as the gas price of transactions, the usage of gas tokens, and predicted gas prices from a large amount of transaction data. They used MLP to detect front-running vulnerabilities. Moreover, novel methods [41], [42] are proposed for detecting pump-and-dump events. They processed the data by removing the moving standard deviation of rush orders, number, and volume of trades, etc. Hu et al. [41] combined statistical features (chain IDs and sequential transactions) to detect pump-and-dump events. The above methods concentrate on the static features of graph structure, which makes it difficult to detect accounts that mimic service providers. Different from these works, we not only optimize the sampling algorithm of the network graph, but also focus on the dynamic evolution of the transaction graph structure.

III. Background

In this section, we provide the essential background on the malicious account detection, which aims to identify illicit accounts that behave maliciously in a transaction graph.

A. Sample Walking

The sample walking method [9], [13] is a graph embedding technique that employs random walks to capture the local features of nodes in a graph. The method selects nodes randomly from the graph and generates node sequences by performing random walks. These node sequences are then used to learn the topological structure of the node from the neighbors. They analyze the neighborhood relationships between nodes in a graph by focusing on local features. Several graph embedding methods based on random walks are applied, such as Node2Vec [9] and DeepWalk [13].

Random walk is a classical implementation of sample walking, which does not consider the attributes of edges in the network. Given a graph, $G = (V, E)$, which comprises a set of nodes represented by V and a set of edges represented by E. It traverses a graph $G = (V, E)$ and collects a sequence of nodes $\{n_0, n_1, ..., n_m\}$, where n_x is the x_{th} node, and n_{x+1} is a randomly chosen neighbor of n_x with some probability $p(n_{x+1}|n_x)$. Then, the node sequences are generated by performing random walks from each node. For each walk, the sample method randomly selected a neighbor of the current node and repeated this process until the walk reached a predefined length. After sample walking, node sequences are fed into the word embedding model, such as Word2Vec, to learn the node features. Moreover, DeepWalk assigns equal probabilities to the neighbors of each node and generates node feature vectors in the same dimension. Node2Vec leverages the alias sampling algorithm [9] to sample edges, giving similar preference to all edges.

B. Deep Graph Learning

The deep graph learning method [43] is a technique that aims to represent graphs in a lower-dimensional space to facilitate the analysis of graphs. A common approach in network representation is to utilize graphs as the basic units of representation, which requires two steps: graph extraction and graph encoding [44]. Graph extraction selects relevant subgraphs from the input graph according to various criteria, such as k-core decomposition [45], community detection [46], etc. Graph encoding projects the high-dimensional features of the selected subgraphs to a low-dimensional vector space [47]. These different methods will capture the features of the graph and the relationships between subgraphs. Given a graph $G = (V, E, \mathbb{V}, \mathbb{E}, Y)$, it comprises a set of nodes represented by V and a set of edges represented by E, the $\mathbb{V}, \mathbb{E}$ mean the node feature matrix and edge feature matrix, and the $Y = \{(v, y)|v \in V\}$ is the label set of the nodes, where y is the label of node v. Graph neural networks [48], [49] can learn a function $F = \{f(g_v) \rightarrow y\}$, where g_v is a subgraph derived from node v in G, to get features from non-Euclidean data, such as graphs, networks, and point clouds. By jointly processing the attributes of the network, graph neural networks can capture both the structure and the attribute information of the graph [50]. Subgraph-based statistics utilize the counts of various attributes in the subgraph as features to represent and analyze graphs, such as the motif sampling method [51], [52]. The resulting subgraph counts can capture the graph features.

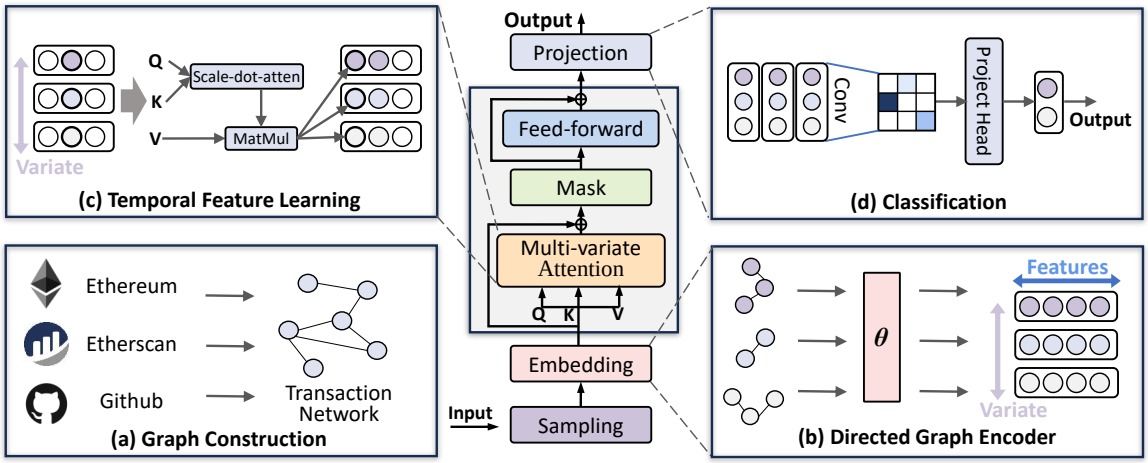


Fig. 2: The Main Example of ScamSweeper. It comprises four components that perform the following steps: (a) transactions are gathered from various public sources to construct a graph; (b) the features of subgraphs with temporal intervals are collected in the multi-directed graph; (c) self-attention is utilized to enhance the feature correlation of subgraphs across different temporal intervals; (d) deep neural network makes a classification with embeddings in local receptive fields.

IV. ScamSweeper

A. Overview

Our work comprises four steps, as illustrated in Fig. 2. Step (a) constructs a graph network by crawling transactions of accounts on Ethereum (§ IV-C). The structure temporal random walk samples a graph network from transactions, building a dataset via time intervals as input (§ IV-D). Step (b) extracts the graph features of the whole directed graph, assigning to each subgraph a directed graph encoder (§ IV-D). Step (c) leverages a transposed Transformer to learn the time series features in the subgraph features (§ IV-E). Step (d) employs a DNN method to make a classification for output (§ IV-E). In the testing phase, ScamSweeper inputs the sampled network generated by the structure temporal random walk. Then, prediction results can be produced by steps (b), (c), and (d).

B. Problem Description

1) ***Definition****:* We model blockchain transactions as a graph, denoted as $G = (V, E)$. In the graph G, the account is expressed as a node[2], and the transaction is expressed as an edge. The set of accounts forms the node set V, while the set of transactions constitutes the edge set E. The direction of the edges is consistent with the direction of the transactions.

2) ***Description****:* Since the graph learning method can intuitively learn the structure features of the node from the network graph, the sequence learning method can address the shortcomings of sequential temporal attributes in dealing with graph learning. Therefore, for a graph $G = (V, E)$ constructed by the blockchain network, we employ the sample walking method to extract the subgraph sequence $S = \{g^0_{v_0}, g^1_{v_0}, ..., g^m_{v_0}\}$, where $g^i_{v_0}$ is the subgraph at the i_{th} temporal interval, v_0 represents the start node, which is the source of the subgraph sequence, and m is the length of the sequence. For the subgraph sequence S of node v_0, we reacquire the node features matrix $\mathbb{V}$, edge features matrix $\mathbb{E}$, and the label y of v_0, to construct a new graph $G_s = (V, E, \mathbb{V}, \mathbb{E}, Y)$. Finally, we learn a function $f(g) \rightarrow y$ (i.e., § IV-D and § IV-E), where g is a subgraph from the subgraph sequence S, mapping the subgraph to a low-dimensional label y (i.e., malicious or benign).

[2]In this paper, we will use "account" and "node" interchangeably.

C. Graph Construction

Before learning the features of transaction graphs, we need to obtain the transactions and construct the transactions as a network graph. We sample the original transaction network graph, generating the subgraph sequence from the account.

1) ***Transactions Retrieval****:* Ethereum is a public trading web3 platform that enables users to deploy contracts executing complex transactions, such as decentralized applications (DApps), decentralized finance (DeFi), etc. Due to the multi-party property of these transactions, the number of transactions generated on web3 is higher than that of traditional Ethereum phishing transactions, which are only simple transfers of funds between accounts. For transactions, we obtain transaction records that contain timestamps, amounts, incoming addresses, and outgoing addresses, leveraging the BlockchainSpider tool [53]. Based on the large-scale transaction data collected, we construct a multi-directed graph. In this graph, the nodes represent Ethereum accounts, and the edges represent transactions between accounts. Moreover, edges exist between two nodes, each with a timestamp and transfer amount. A multi-directed graph is a type of graph structure, that allows multiple directed edges between the same pair of nodes.

2) ***Structure Temporal Random Walk****:* The transaction data on Ethereum has grown to 2,762 million until April 2025 [24], which poses a significant challenge for deep graph learning methods that require processing the entire data. Therefore, to alleviate the requirements of large-scale transaction data on computing resources, we implement the

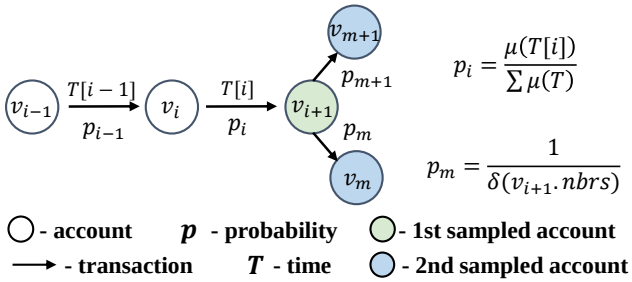


Fig. 3: An Example of Structure Temporal Random Walk. The green circle represents the first-step sampled node, and the blue circle indicates the second-step sampled nodes.

Structure Temporal Random Walk (STRWalk) techniques to sample the network. It allows us to navigate the network and gather relevant neighbors and edges while streamlining the size and intricacy of the transaction network graph.

As Algorithm 1 shows, STRWalk takes the graph $G(V, E)$, start node v, and other parameters (i.e., structure window size w, temporal interval τ, walk length ξ) as input. When the number of walks satisfies the walk length ξ in Line 12, it will return the sampled graph as output. Through inputting different start nodes, such as phishing or normal accounts, we can obtain corresponding sampled graphs and use them to train and evaluate ScamSweeper in § V.

STRWalk can be divided into two steps. Firstly, STRWalk utilizes the temporal attribute, which is an indispensable element of transactions. To account for the temporal dimension of many contemporary problems [15], [12], [26], we incorporate the temporal attributes into the node sequence sampled from the network. During Line 14-23, given the current node v_i, it gets the neighbors $\{v_i^1, v_i^2, ..., v_i^n\}$ of v_i, it computes the set of temporal attributes $T = \{t^1, t^2, ..., t^n\}$ for all the edges connected neighbors of v_i, where $t^j \in T$ means the timestamp value of edge between v_i and v_i^j, $j \in [1, n]$. During each walk, we calculate the probability P_i as shown in Eq.1, which subtracts the minimum value in T from each element.

$$P_i = \frac{\mu(T[i])}{\sum_{j=1}^{n} \mu(T[j])}, \quad \mu(T[\ell]) = T[\ell] - \text{Min}(T) + 1 \tag{1}$$

where e_i means the edge between v_i and v_{i+1}, $T[i]$ indicates the timestamp when the edge e_i happened, μ represents the difference operation from the shortest time of edges in the network. Through the Eq.1, edges closer to the start node will get higher sampling weights. The purpose of the Eq.1 is edges closer in time to the start node should be given more weight, since the closer edge is more effective at representing the start node [15]. Then, we employ the typical alias sample algorithm [13] to randomly select a neighbor of n_i according to the probability distribution P. Under the first step of STRWalk, it weights the temporal property of the network.

As for the second step of STRWalk in Line 27, which will be equivalent to the function in lines 1-10, enriching the structural information of the nodes. The purpose of this step is that although the temporal attributes are extracted through the first step, the structure information is also important for

Algorithm 1: StructureTemporalRandomWalk(G, w, τ, v, t)

Input : graph $G = (V, E)$; structure window size w; temporal interval τ; start node v; walk length ξ;
Output: sampled graph G_s

1 **Function** `Structure_Sample(`v, G`)`**:**
2 $V, E = [\,]$;
3 σ = GetMinTimestamp(v, G);
4 $edges$ = SearchEdges($v, \sigma, \sigma + \tau$); ▷ get the edges during $(\sigma, \sigma + \tau)$;
5 **for** $i \in [1, w]$ **do**
6 $E \leftarrow$ Alias_Sample(1/Num($edges$)); ▷ sample w nodes;
7 **end**
8 $V \leftarrow$ next_node(E);
9 **return** (V, E);
10 walk = [v] ▷ Initiate the walk sequence;
11 $G_s \leftarrow$`Structure_Sample(`v, G`)`;
12 **for** $i \in [1, \xi)$ **do**
13 v_i = walk[-1]; ▷ get the current node v_i;
14 Get the neighbors $v_i.nbrs$ of the current node v_i, and the loop breaks if $v_i.nbrs$ equals to 0;
15 Get the edges ϵ connected v_i and its neighbors $v_i.nbrs$;
16 σ = GetMinTimestamp(v, G);
17 Sort the edges ϵ by time in ascending order to get λ;
18 ε = Alias_Sample($\lambda / \sum \lambda$);
19 v_{i+1} = next_node(ε); ▷ get another node in the edge;
20 walk.append(v_{i+1});
21 $G_s \leftarrow$`Structure_Sample(`v_{i+1}, G`)`;
22 **end**
23 **return** G_s

graph learning. Therefore, we apply an expansion strategy to the walking sequence, extending the structure information. To augment the structural information of the nodes, Fig. 3 shows that we supplement other nodes. Suppose that the current node v_i has finished the first step of STRWalk, and starts the second step. It selects the related nodes at the same temporal interval τ. The probability distribution can be calculated as Eq.2, and then the alias sample algorithm will be used for selection.

$$P_m = \frac{1}{\delta(v_i.nbrs)} \tag{2}$$

where $\delta(v)$ represents the number of nodes that are in the same interval with node v.

Moreover, we set the time unit to *day* to avoid collecting useless information with short intervals and prevent noise with long intervals [15]. Suppose that the interval is k days, the position of the temporal interval τ can be computed as following Eq.3, where 86,400 represents one day, as the timestamp is measured in seconds.

$$\tau = \frac{t - t_{first}}{86{,}400 * k} \tag{3}$$

Then, we repeat the first and second steps until the sampled walk sequence reaches a fixed length or the nodes have no neighbors. Finally, the node sequence $S_w = \{v_0, v_1, ..., v_m\}$ is selected by the first step, and the second step obtains the subgraph sequence $\{g_{v_0}^0, g_{v_0}^1, ..., g_{v_0}^m\}$, where g_n^i from the $v_i \in S_w$ and the v_0 is the start node. Therefore, the expansion strategy integrates the structural features into the node sequence and

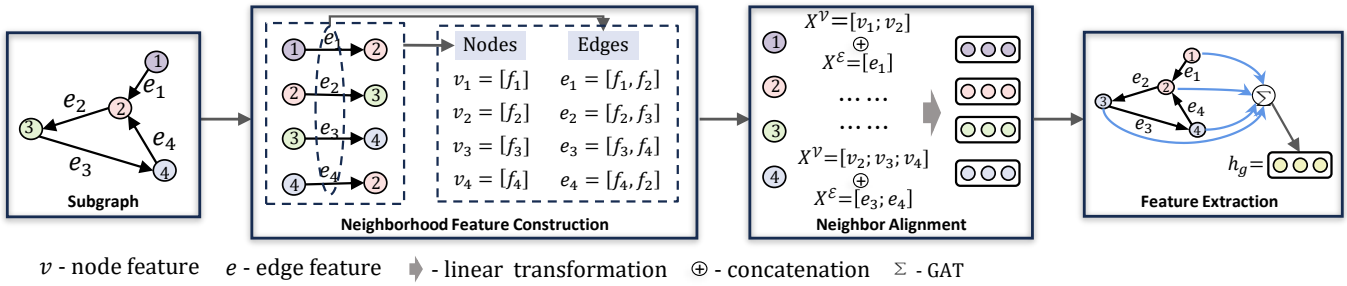


Fig. 4: An Example of Directed Graph Encoder in a Single Temporal Interval.

forms the subgraph sequence as the sampled graph G_s.

D. Directed Graph Encoder

Given the sampled graph $G_s = (V, E)$ with the start node v_0 contains m subgraphs $\{g^0_{v_0}, g^1_{v_0}, ..., g^m_{v_0}\}$ as sequence. We utilize the directed graph encoder shown in Fig. 4 to learn the structure feature of each subgraph in G_s.

*1) **Neighborhood Feature Construction**:* Let the subgraph G_g as input graph with α nodes $\mathcal{V} = \{v_1, ..., v_\alpha\}$ and β edges $\mathcal{E} = \{e_1, ..., e_\beta\} \subseteq \mathcal{V}^2$, associated with node features $X^{\mathcal{V}} \in \mathbb{R}^{\alpha \times C}$ and edge features $X^{\mathcal{E}} \in \mathbb{R}^{\beta \times C}$. Then, we construct the edge feature matrix $\mathbb{E}$ and the node feature matrix $\mathbb{V}$, converting the node sequence network $G = (V, E)$ to $G = (V, E, \mathbb{V}, \mathbb{E}, Y)$ for graph learning.

Given a node walk sequence in the network, we consider that multiple edges can be between any pair of nodes, and each edge connects two nodes. For all transactions $E = \{X_f \to X_t\}$, X_f and X_t are the sets of source and target node features, respectively, which can be defined as $X_f = [X^1_f, X^2_f, ..., X^n_f]$, $X_t = [X^1_t, X^2_t, ..., X^n_t]$, where X^i_f represents the i_{th} incoming node feature, X^i_t represents the i_{th} outgoing node feature, and n is the edge amount. The nodes and edges representation set in the graph can then be represented as following Eq.4, where e^i means the i_{th} transaction.

$$\begin{aligned} V &= \{X_f; X_t | (X^1_f; X^1_t, X^2_f; X^2_t, ..., X^n_f; X^n_t)\} \\ E &= \{X_f \to X_t | (e_1, e_2, ..., e_n)\} \end{aligned} \tag{4}$$

To construct the edge feature matrix $\mathbb{E}$, we use a matrix $W \in \mathbb{R}^{1 \times n}$ to represent $\mathbb{E} = WE$. Moreover, the node feature matrix can be repressed by $\mathbb{V} = [WX_f, WX_t]^T$. Utilizing the directed graph encoder θ in Fig. 2(b), we can learn the features of all the nodes and the corresponding edges in the latent space from the whole graph. For a node v, the derived network is represented by $G = \{g^0_v, g^1_v, ..., g^m_v\}$, where g^i_v is the subgraph at i_{th} interval, and m is the number of intervals that a node network have been survived. The learning process of the subgraph is illustrated in Fig. 4.

*2) **Neighbor Alignment**:* As the subgraph is centered on a certain start node, we need to examine the features of its neighbors and the edges, to identify the patterns and features of the graph. In the neighborhood feature construction process, we first obtain all the node and edge representations in the whole graph. Then, for each subgraph, we get all the node features $X^{\mathcal{V}}$ and the directed edge features $X^{\mathcal{E}}$ sequentially concatenated by the two nodes. For example, as for the node f_1 in the subgraph G_g in Fig. 4, the $X^{\mathcal{V}}$ is the concatenation of related node features $[v_1; v_2]$, the $X^{\mathcal{E}}$ is the concatenation of related edge features $[e_1 = [f_1; f_2]]$.

In each subgraph, the nodes contain different numbers of neighbors. Therefore, the dimensions of neighborhood features $X^{\mathcal{V}}$ and directed edge features $X^{\mathcal{E}}$ are different. To align these differences, we normalize the attribute features of the neighbor nodes, and Eq. 5 aligns the neighborhood features of the associated target node.

$$\hat{v} = \text{LeakyRelu}(\Theta_v \cdot [X^{\mathcal{V}} | X^{\mathcal{E}}] + b_0), \tag{5}$$

where Θ_v and b_0 are a linear transformation layer and the bias respectively, which are employed to align the dimensions of the neighbor features. For nodes with various numbers of neighbors, the zero padding is used to unify their dimensions.

*3) **Feature Extraction**:* The subgraph is characterized by its node-centric structure, in which each constituent node in the neighborhood possesses a different magnitude of significance to the central node. To effectively extract the feature of the central node from the perspective of the subgraph, we need to comprehend the different significance of the neighborhood with the source node. Hence, to quantify the neighborhood with different contributions, we employ the graph attention network (GAT) [54] to delineate the behavior patterns. It facilitates calculating the importance degree among nodes and learning graph-pattern features, thereby learning the structure feature in each subgraph.

Specifically, for the central node v_a in the subgraph g_{v_a}, the GAT network first obtains the attention scores of the neighbor v_b for the v_a in the subgraph g_{v_a} as following Eq.6,

$$e_{ij} = \text{LeakyRelu}(\Theta_n \cdot [h_i | h_j]), \tag{6}$$

where h_i and h_j are the hidden features of v_a and v_b, and a linear transformation Θ_n and an activation function $LeakyRelu$ are performed to compute the attention scores. The initial value of h_i is the feature of the subgraph's central node v_a. Then, to compare with all the neighbors, we employ a softmax layer to make a normalization with the scores as following Eq.7,

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{x \in N(i)} \exp(e_{ix})}, \tag{7}$$

where $N(i)$ is the neighbors of v_a. After obtaining the impor-

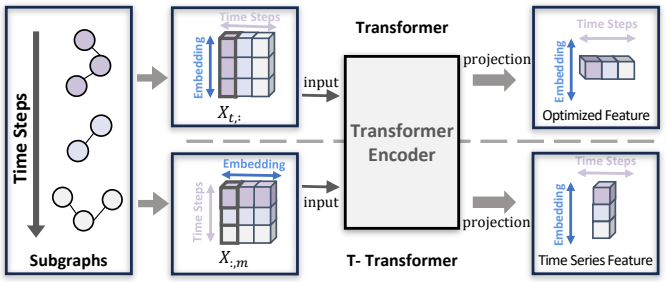


Fig. 5: A Difference Example of the Transposed and the Traditional Transformer. The upper part of the dotted line is the transposed Transformer, while the lower part represents the traditional Transformer.

tance distribution, we incorporate it into the node features to update the feature of the subgraph as following Eq.8,

$$h_g = \text{Elu}(\alpha_{ij} \cdot \Theta \cdot h_i + \sum_{x \in N(i)} \alpha_{ix} \cdot \Theta \cdot h_x), \tag{8}$$

where h_g represents the feature of the subgraph g_v, h_i is the node feature of subgraph's central node v_a, Elu is an activation function [55], Θ means a linear transformation layer.

By extracting the feature h_g of each subgraph G_g in all the temporal intervals sequentially, we can obtain the temporal subgraph feature sequence $\Phi = (h_g^0, h_g^1, ..., h_g^m) \in \mathbb{R}^{m \times D}$ for § IV-E, where D means the hidden size, m is the variable temporal length.

E. Temporal Subgraph Sequence Learning

To capture the dynamic evolution in the temporal subgraph feature sequence in § IV-D, we employ sequence learning techniques as shown in Fig. 5.

The traditional Transformer [56] learns the features of different time steps in the embedding dimension. However, to learn the dynamic evolution of the embedding over the whole time steps, we employ transposed Transformer layers for learning, which has been proven to be effective at § V-D. For example, after extracting the features in § IV-D, we divide the entire temporal graph into several intervals (see Eq. 3), resulting in a variate time series $G = (g_n^0, g_n^1, ..., g_n^m)$ and a variate time feature $\Phi = (h_g^0, h_g^1, ..., h_g^m) \in \mathbb{R}^{m \times D}$, where D means the hidden dimension, m is the variable temporal length. The dimension of each subgraph feature is fixed, assuming that $X_{t,:}$ is the variable positions at the same temporal interval, and $X_{:,m}$ is the entire time series at the same position. Fig. 5 illustrates that the Transformer [56] learns feature sequences at the same time from $X_{t,:}$ simultaneously. To learn the dynamic evolution of features, we focus on $X_{:,m}$, which captures the temporal evolution at each position of embedding. The transposed Transformer structure transposes the feature matrix to learn the same feature at different time steps. As shown in Fig. 2, we utilize the transposed layers to extract the feature representation $H = \{h_0, h_1, ..., h_{m-1}\} \in \mathbb{R}^{m \times d}$ from $X_{:,m}$, where $h_i \in \mathbb{R}^d$ captures the temporal variation of the entire temporal sequence.

First, we feed the time series into linear layers Θ_Q, Θ_K, and $\Theta_V \in \mathbb{R}^{d \times d}$, generating queries Q, keys K, and values $V \in \mathbb{R}^{m \times d}$. Then, the self-attention layer computes the importance scores of multiple tokens at the same time step, which can weigh the next hidden state in V, as following Eq.9,

$$h_s = \text{softmax}(\frac{Q \cdot K^T}{\sqrt{d}}) \cdot V, \tag{9}$$

where h_s is the output, which is the temporally weighted vector at spatial locations across varying temporal dimensions.

Furthermore, a mask mechanism is employed. The mask approach uses a vast negative or minor positive number to fill specific positions, resulting in a zero value at some positions during weight calculation. The mask approach can prevent the self-attention calculation from capturing future information, and disregard the padding position. Moreover, to enrich the feature, we implement the feed-forward layer, which is practically a two-layer fully connected network that can be calculated by Eq.10:

$$h = (\sigma(h_s \cdot W_1 + b_1)) \cdot W_2 + b_2, \tag{10}$$

where $W_1, W_2 \in \mathbb{R}^{d \times d}$ and $b_1, b_2 \in \mathbb{R}^{d \times 1}$ are the trainable parameters, σ means the Sigmoid activation function. h is the result of the feed-forward layer.

Then, we employ a deep neural network to discern temporal sequence relationships across different locations. For example, the convolutional neural network (CNN) and pooling layers [57] learn features at different locations for classification, which is predicated on the widely acknowledged ability of classification tasks. Lastly, the Projection head [56] is used to make the classification.

Furthermore, during the training process, we aimed to minimize the objective function: $O(\theta) = L(\theta) + \omega(\theta)$, where $L(\cdot)$ represents the cross-entropy loss function and $\omega(\cdot)$ denotes the regularization term to prevent overfitting.

V. Experimental Evaluation

In this section, we conduct experiments to evaluate ScamSweeper and address the following research questions (RQs):
RQ1: **What distinguishes normal and malicious accounts?** What is the difference between malicious accounts and normal accounts, and why is ScamSweeper effective?
RQ2: **Is ScamSweeper effective for detecting Web3 scams?** Will ScamSweeper work better than other methods?
RQ3: **Is each component in ScamSweeper effective?** Is STRWalk robust? Which component (graph learning or sequence learning) makes a greater contribution to ScamSweeper?
RQ4: **Is ScamSweeper effective for detecting phishing accounts?** Will ScamSweeper have better detection capabilities than existing phishing account detection methods?

A. Experimental Setup

*1) **Datasets**:* We collect a large-scale transaction dataset of phishing and web3 scam accounts for evaluation, two types of malicious accounts on the Ethereum blockchain. Leveraging the BlockchainSpider tool [53] with the depth-first algorithm, we scrape the first 18 million blocks, covering the period from July 2015 to November 2023. During this period, the

transactions of 4,905 phishing accounts labeled by Etherscan are crawled in this dataset. We also supplement the dataset with the 3,125 transaction networks from the web3 scam Database [58], which exhibits scam behavior while providing web3 services. All the data are labeled and audited by the researchers, before collecting them through an official API from the Etherscan, thus ensuring the soundness of the dataset.

We randomly divide the dataset at § V-A into 70%, 20%, and 10% for training, validation, and test set, respectively.

2) ***Baseline****:* We compare ScamSweeper with several studies [31], [33], [32], [16], [11], [59], [26]. They can divided into three types, including the random walk-based method (i.e., Yuan et al. [31] and Choi et al. [33] utilize the Node2vec and DeepWalk, respectively), the graph learning method (i.e., the GCN [48], GAT [54], GraphSAGE [60] which are embedded into Chen et al. [32], Zhou et al. [16], and Li et al. [11], respectively), and the sequence learning methods (i.e., PDTGA [59] and BERT4ETH [26] leverages the Transformer structure).

Regarding the random walk-based methods, Node2Vec [9] is a method that uses the random walk technique to learn the feature representation of accounts, which are represented as nodes in the graph or network. Yuan et al. [31] leverage Node2Vec to learn the domain representation, where the node network is generated by random walk and skip-gram. Similar to Node2Vec, Choi et al. [33] adopt DeepWalk [13], which utilizes a random walk to obtain graph networks, then generates features of nodes by the Word2Vec method.

Regarding the graph learning method, Chen et al. [32] evaluate the malicious account identification task with the GCN method, which refers to a kind of neural network that performs convolution operations on a graph. It gives weight to the features of nearby nodes as node features. Different from GCN, GAT [54] is a graph neural network that introduces an attention mechanism. Zhou et al. [16] utilizes GAT to weigh the features of neighboring nodes, so that the model concentrates more on the appropriate neighbors. In contrast to GCN and GAT, GraphSAGE [60] is optimized to generate embeddings for previously unseen data, allowing it to handle dynamic graphs.

Regarding the sequence learning method, the PDTGA [59] leverages the Transformer [56] structure, which is a prominent sequence learning model with the attention mechanism, capturing the dependencies of lengthy sequences. The encoder receives the input sequence, while the decoder generates the output sequence. The tool named BERT4ETH [26] is a Transformer-based neural network architecture that has been designed and fine-tuned explicitly on Ethereum blockchain datasets. Empirical evidence has established its efficacy in identifying and preventing the operation of phishing accounts.

3) ***Evaluation Metrics****:* In this paper, we mainly leverage the F1-score to evaluate our model's ability to detect malicious nodes on Ethereum. To gain a more comprehensive analysis of the model, we also employ accuracy, precision, and recall for evaluation. The accuracy metric indicates the percentage of correct predictions made by the model, encompassing both positive and negative predictions. The recall metric reveals the likelihood of actual web3 scams or phishing nodes in the test samples being accurately identified by the model. The precision metric demonstrates the ratio of true positives in the test samples that are correctly identified as positive by the model. The F1-score is a comprehensive performance metric that considers both precision and recall. It reflects the disparity between the model's classification outcomes and the actual values and is the harmonic mean of precision and recall. Higher F1-scores indicate superior classification performance.

4) ***Environments****:* We perform the analysis and experiments on an Ubuntu server 22.04, equipped with an NVIDIA GeForce GTX 4070TI GPU and an Intel(R) Core(TM) i9-13900KF CPU. The server boasts 128GB of RAM and 1TB of SSD storage. To learn the features of transactions, we collected transactions from labeled malicious and normal accounts during the data collection phase. To collect second-order transactions, we utilize the breadth-first algorithm and set the query timeout to 180 seconds. In the STRWalk method, we set the structure window to $\{5,10,15\}$. For optimization, we use the Adam and set the weight decay rate to 5×10^{-4}.

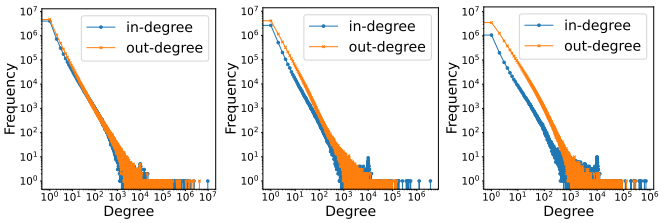
B. RQ1: Difference among Normal, Phishing, Web3 Scam Accounts

We conduct an analysis of the distribution phenomena among normal, phishing, and web3 scam accounts, fostering a deeper understanding and examination of the distinctions between malicious and normal accounts.

We collect normal, phishing, and web3 scam nodes on Ethereum shown at § V-A. The collection method of normal accounts follows Wu et al. [15], including the classes of exchange, mining, ICO wallet, and gambling, which is marked in xlabelcloud [61]. After the collection, we construct a multi-directional graph by treating the accounts as nodes and the transaction vectors between the accounts as directed edges. The attributes of the multi-directional graph show in Table I.

TABLE I: Statistical Information in the Multi-directional Graph. #Nodes means the total number of nodes, #L is the number of source nodes, #Edge indicates the number of transactions, and #SD Degree is the average variance value.

Datasets	#Nodes	#L	#Edges	#SD Degree
Normal	12,042,066	636	142,750,370	3555.35
Phishing	10,159,847	4,905	62,011,219	1285.25
Web3 Scams	8,736,430	3,125	64,265,586	541.10



(a) Normal (b) Phishing (c) Web3 Scams

Fig. 6: The Frequency Distribution of in- and out-Degrees for Nodes Classified as Normal, Phishing, and Web3 Scams. The blue dot represents the in-degree of the nodes, and the orange dot represents the out-degree of the nodes.

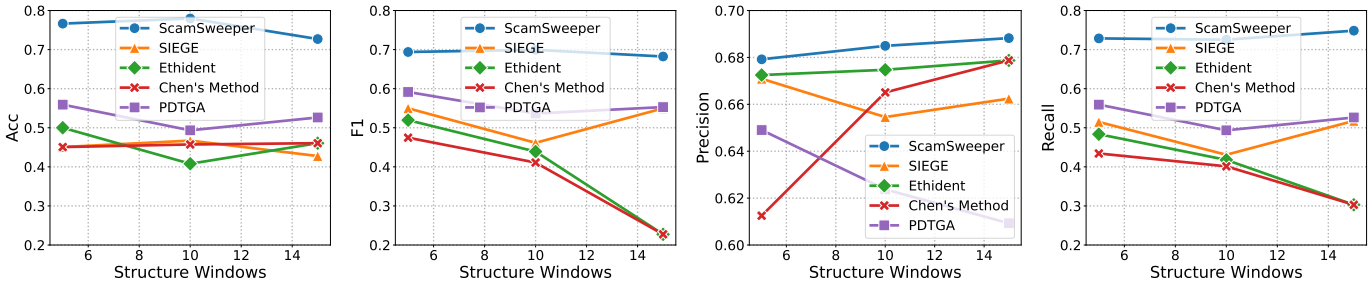


Fig. 7: The Comparison Results on Web3 Scams Detection. The blue line represents the ScamSweeper, the orange line indicates the SIEGE, the green line means the Ethident, the red line is Chen's method, and the purple line is the PDTGA.

The dataset analysis provided in Table I reveals that the network of normal nodes boasts a greater average number of edges and nodes when compared to other networks. However, the network of normal nodes also exhibits the largest standard deviation of degrees, which is indicative of a more dispersed node distribution. On the other hand, web3 scams showcase the smallest standard deviation of degrees, suggesting a more clustered node distribution where a limited number of nodes generate most transactions.

To visualize and compare the different nodes, we draw a degree-frequency graph in Fig. 6 to analyze the degrees of the nodes. The degree distribution of all categorized nodes, as depicted in Fig. 6, adheres to a power-law distribution. The long-tail phenomenon posits that most of the revenue is concentrated among these few critical nodes. Upon examination of the disparity between in-degrees and out-degrees, it becomes apparent that the entry and exit degrees of standard nodes are comparably similar. However, there is a discernible difference in the access degrees between phishing nodes and those associated with web3 scams, with the latter exhibiting a more pronounced effect. Additionally, the frequency of consumption (out-degree) transactions significantly surpasses that of profit (in-degree) transactions. The pattern implies that these nodes primarily generate income via extensive outward propagation. Furthermore, the overall number of incoming connections to nodes in the phishing network is lower than in web3 scams. Conversely, the phishing network has a higher total out-degree of nodes compared to web3 scams.

In Fig. 6, we can observe that the area of the curve, in terms of its horizontal and vertical coordinates, is represented in total degrees. The larger the area between the orange and blue curves in the graph, the greater the difference between the discrepancies. Therefore, the empirical data indicates that only a restricted number of pivotal nodes are accountable for initiating transactions in web3 scams and phishing networks. However, the number of receiving nodes in phishing is higher than in the web3 scam network, so the profit nodes of the web3 scam network are more concentrated.

Therefore, the data suggests that the web3 scams network has a higher propensity to expand outward (out-degree) and a lower inclination to expand inward (in-degree) than the phishing network. So, the directed feature learning of graph structure in ScamSweeper can match the in-out degree attribute features and distinguish normal nodes from malicious nodes. The limited number of nodes will produce a multitude of transactions, resulting in introducing a lot of noise by the methods that only consider the graph structure. Therefore, to effectively analyze the dynamic evolution in transaction data and to differentiate between normal nodes and illegal nodes (i.e., phishing and web3 scam accounts), the sequence information of the graph structure should be considered.

Answer to RQ1. The degree-frequency follows a power-law distribution, indicating that only a few nodes are crucial in the entire network. The key feature is that the three nodes possess unique in- and out-degree distributions.

C. RQ2: Detection on Web3 Scam Accounts

To validate the effectiveness of ScamSweeper in detecting web3 scams and answer RQ2, we conduct experiments with various graph learning techniques (i.e., the method utilized by Chen et al. [32], Ethident proposed by Zhou et al. [16], and SIEGE presented by Li et al. [11]), and the sequence learning method (i.e., PDTGA introduced by Wang et al. [59]).

We compare them with ScamSweeper across different structure window sizes, within the 3,125 labeled Web3 scams dataset described in Table I. Specifically, to evaluate it, we randomly select the same number of nodes from the network other than web3 scams as negative samples, balancing the dataset. Furthermore, we implement a three-layer structure for GAT, GCN, and GraphSAGE in Ethident, Chen's method, and SIEGE. GraphSAGE is a type of GNN model that samples a fixed number of neighbor nodes, and we employ a three-layer structure with mean aggregation. The PDTGA method has been enriched with several key components, including the self-attention, feedforward neural network, mask mechanism, and position embedding. Within it, the hidden dimension size has been set to 16, which is the same as ScamSweeper.

As illustrated in Fig. 7, our experimental results show that ScamSweeper performs better than other methods. As for graph methods, at a structure window of 15, ScamSweeper scores a recall of 0.75, which is 0.23-0.44 higher than other graph methods. Therefore, ScamSweeper achieves higher F1-scores than other methods, with scores of 0.69, 0.70, and 0.68 in the three structure windows. In comparison, the highest F1-scores achieved by Ethident, Chen's method, and SIEGE are 0.52, 0.47, and 0.55, respectively. Note that the three indicators

used in RQ2 are weighted for positive and negative samples, thus providing an accurate reflection to distinguish between the malicious and normal nodes. The PDTGA achieves better F1-score and recall than graph learning methods, but there is still a certain gap compared with ScamSweeper. As shown in Fig. 7, for [F1-score, precision, recall], ScamSweeper outperforms PDTGA by at least about [17.29%, 4.64%, 30.29%].

Answer to RQ2. When the structural window sets 10, ScamSweeper achieves a weighted F1-score of 0.70, significantly 17.29%-48.94% higher than other methods.

D. RQ3: Impact of Each Component

To show the efficiency of the components (§ IV-C, § IV-D, and § IV-E), we conduct an ablation experiment.

We carry out an ablation study, with the final classification outcomes as the results of the influence of various components. The experiment data are processed through the STRWalk, acquiring graphs from a large-scale network (See § IV-C). Subsequently, experiments are conducted on the graphs, enabling further comparison. In the subgraph learning stage (See § IV-D), we directly remove the graph-learning layer for comparison, illustrating the impact of graph-learning techniques within ScamSweeper. In the subgraph sequence learning phase (See § IV-E), we substitute the transposed Transformer encoder with a conventional Transformer structure, e.g., the different structure depicted in Fig. 5, indicating the influence of the transposed Transformer. Specifically, we set the number of encoder layers to 1, the number of heads to 2, the feature size to 16, the temporal interval to 7 days, and the temporal length to 3 years. Notably, we focus on evaluating the ability of each component in web3 scam detection.

TABLE II: The Ablation Results of Each Component. The @ indicates the structure window in the STRWalk. The ScamSweeper-t is without the transposed Transformer, and the ScamSweeper-g is without the graph learning.

Metric	Method	@5	@10	@15
F1-score	ScamSweeper-t	0.41	0.43	0.44
	ScamSweeper-g	0.53	0.51	0.55
	ScamSweeper	**0.78**	**0.78**	**0.73**
Weighted F1-score	ScamSweeper-t	0.42	0.46	0.47
	ScamSweeper-g	0.57	0.55	0.58
	ScamSweeper	**0.69**	**0.70**	**0.68**

Upon thorough examination of all three structural windows shown in Table II, the ScamSweeper-t in Table II shows a range of positive F1-scores of 0.41-0.44 at window size 5. This range is comparatively lower than ScamSweeper-g, which records a range of 0.51-0.55. However, when the window size is 15, ScamSweeper's ability to detect positive samples is weaker. The reason is that the window, i.e., temporal interval, contains more noisy data, resulting in a relatively chaotic feature matrix after the phase of the graph learning encoder. However, the enhanced performance of ScamSweeper-g is observed in detecting scams when the structure window is 15 from 10, which suggests that the transposed Transformer can alleviate the issue by focusing on the dynamic evolution of all the temporal features in the embedding dimension. Thus, the impact of the graph learning on ScamSweeper is more significant than the impact of the transposed Transformer.

As shown in Table II, ScamSweeper is top-performing across all three structural windows compared to other incomplete scenarios. ScamSweeper achieved weighted F1-scores of $\{0.69, 0.70, 0.68\}$ at the three structural windows, which is 0.1-0.15 higher than ScamSweeper-g and 0.21 to 0.27 higher than ScamSweeper-t. However, when the window size is 10, there is a decrease in the web3 scams detection capacity of ScamSweeper. Nonetheless, ScamSweeper's weighted F1-score is higher than others, indicating superior performance in identifying web3 scams and normal nodes.

Drawing from the experiment's findings, it may be inferred that the comparison results retain the order of ScamSweeper > only sequence learning > only graph learning, even when the STRWalk is restarted. Additionally, Table II reveals that there is a minimal variation (up to 0.05 in the F1-score and 0.02 in the weighted F1-score) among the three times of STRWalk. It suggests that the randomness of STRWalk has minimal influence on the performance of ScamSweeper.

Answer to RQ3. Graph and sequence learning components improve the F1-scores by 47.17% and 90.24%, respectively, with a structural window size of 5. Moreover, ScamSweeper has good robustness in different structural window sizes.

E. RQ4: Detection on Phishing Accounts

ScamSweeper can also identify phishing scams, showcasing its generalization capabilities. Therefore, phishing nodes are assessed to evaluate the model's performance.

TABLE III: The Comparison Results on Phishing Node Feature Detection. † , ‡ , $_{dnn}$, and $_{rf}$ refer to the utilization of STRWalk, TRWalk, DNN, and random forest, respectively.

Method	Precision	Recall	F1-score
Yuan's method† [31]	0.82	0.77	0.79
DGTSD† [33]	0.82	0.79	0.80
BERT4ETH†$_{dnn}$ [26]	0.76	0.74	0.75
BERT4ETH†$_{rf}$ [26]	0.81	0.64	0.71
ScamSweeper‡	**0.86**	**0.84**	**0.85**
ScamSweeper†	**0.95**	**0.92**	**0.94**

Since the STRWalk is divided into two steps: time and structure sampling, and only the end of time sampling is followed by structure sampling. So, we sample the network using Node2vec, DeepWalk, temporal random walk (TRWalk), and STRWalk, respectively. Moreover, to compare the effects of these walking methods at their optimal states, we follow the parameter settings at [62], which compared various walking methods. Therefore, the walk length is set to 20, the window size is 4, and the embedding dimension of the nodes is 128. The max sequence length in BERT4ETH is set to 100. In addition, our data collected in Table I shows a significant

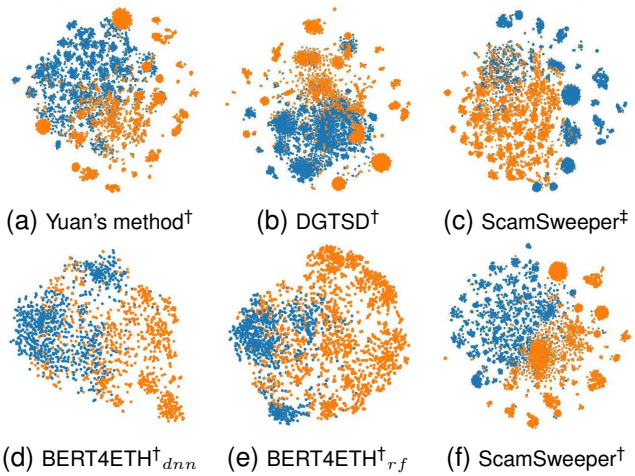


Fig. 8: The Embedding Visualization of Node Feature Distribution by T-SNE, where ■ represents the malicious accounts, and ■ means the normal accounts.

imbalance between phishing and normal nodes, with a ratio of 4905:636. Therefore, to ensure a practical evaluation of phishing node detection without introducing noise, we utilize the dataset called `EPTransNet` [63], which is a subset of our dataset containing 1,165 phishing nodes. We then supplement it with the 636 normal nodes we collected to address the imbalance of positive and negative samples in our dataset.

As shown in Table III, the TRWalk with only time and the STRWalk with time and structure in ScamSweeper can achieve better performance than existing methods. We can find that ScamSweeper with STRWalk can achieve over 17.5%, 15.85%, and 16.46% higher than other methods in F1-score, precision, and recall. ScamSweeper with TRWalk can achieve 6.25%, 4.88%, and 6.33% higher than the other methods in terms of F1-score, precision, and recall. The original Node2Vec and DeepWalk in Yuan's method and DGTSD both use the random walk to sample the network. We only show the case when Yuan's method and DGTSD use STRWalk. For further comparison, we describe the benefits of random walk and STRWalk for Yuan's method and DGTSD in § VI-C.

To visualize the distribution of classified nodes, we leverage t-SNE [64] as the classification algorithm, representing normal and malicious nodes in blue and orange, respectively. Since the number of negative nodes in the dataset is larger than the positive nodes, we sample the same number of node features from the negatives. Fig. 8 shows the performance of different models on phishing detection, i.e., Yuan's method, DGTSD, TRWalk, BERT4ETH$_{dnn}$, BERT4ETH$_{rf}$, and STRWalk. The BERT4ETH$_{dnn}$ and BERT4ETH$_{rf}$ utilize the DNN [65], [26] or random forest [26] algorithm to identify phishing nodes. ScamSweeper† refers to the phenomenon that utilizes the STRWalk. It can be found that the TRWalk demonstrates superior discriminative capability compared to Yuan's method and DGTSD, thus confirming the enhanced effectiveness of temporal attributes in detecting malicious nodes within networks. Furthermore, the ScamSweeper† with STRWalk are separable, and most malicious nodes can be linearly separated.

Answer to RQ4. ScamSweeper with STRWalk can identify phishing nodes to a certain degree, resulting in an enhancement of 17.5% in F1-score, 15.85% in precision, and 16.46% in recall.

VI. Discussion

A. Case Study

We identify a web3 scam account [66] based on the Drainer toolkit and a traditional phishing account [67] as cases, which have been reported and annotated in Etherscan [24].

*1) **Case I**:* The accounts serving web3 services send a large number of transaction requests. These attackers utilize platforms such as service providers to promote fraudulent information in large quantities. Fig. 9(a) shows an example of the transaction graph of a scam attacker that provides web3 services, where the length of the edges is inversely proportional to their timestamp. The Fig. 9(b) is the transaction graph sampled by the STRWalk. We set the value of τ to 7 days, and the structure window is set to be unrestricted. The dynamic evolutions are shown in Fig. 10. We can observe that the number of neighbor nodes gradually decreases with time. Finally, it diffuses to a large number of accounts in the last time interval.

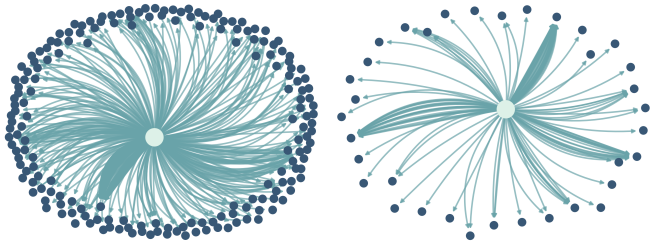


Fig. 9: An Example of Transaction Network of Web3 Scam Accounts, where ■ represents the source account of the network, and ■ means other connected accounts.

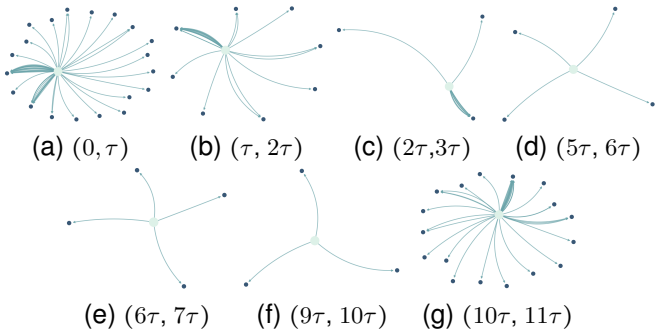
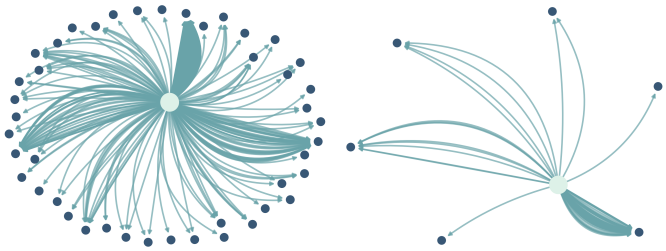


Fig. 10: A Dynamic Evolution Example of Transaction Network of the Mimic Account in the Web3 Scam as Service.

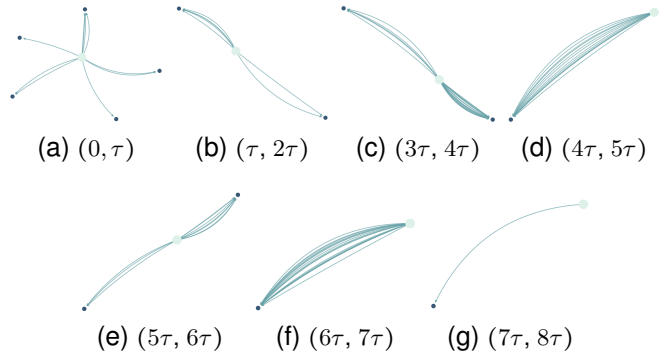
The fraudulent activity commenced by mimicking typical transaction patterns initially during $(0, \tau)$ and then gradually progressed to defrauding users during $(\tau, 10\tau)$. Throughout this phase, the transactions have been focused on fewer accounts to transfer profits into some individual accounts. After profiting, a fresh round of mimicking during $(10\tau, 11\tau)$ could have commenced.

*2) **Case II**:* Furthermore, traditional phishing accounts widely distribute phishing websites or scam information and steal the assets of some accounts (i.e., victims), without hiding behind the service providers. For the phishing scams shown in Fig. 12, we noticed that the number of accounts involved in each time interval is relatively small. However, the number of transactions between two nodes is relatively high.



(a) Original (b) Detected Pattern

Fig. 11: An Example of the Transaction Network of A Phishing Account, where ■ represents the source account of the network, and ■ means other connected accounts.



(a) $(0, \tau)$ (b) $(\tau, 2\tau)$ (c) $(3\tau, 4\tau)$ (d) $(4\tau, 5\tau)$

(e) $(5\tau, 6\tau)$ (f) $(6\tau, 7\tau)$ (g) $(7\tau, 8\tau)$

Fig. 12: A Dynamic Evolution Example of the Transaction Network of the Phishing Account.

It shows that a phishing node initiates its activity by disseminating a transaction request. It is noticeable that there is a significant volume of transactions transpiring amidst nodes during the $(\tau, 7\tau)$. This implies that the fraudulent accounts are selecting lucrative nodes and subsequently transferring their earnings during the $(7\tau, 8\tau)$. In contrast to the means employed in Web3 scams, a phishing node does not revert to its prior condition following its initial engagement with a substantial number of accounts and subsequent profiting.

B. Runtime Overhead

The reason why ScamSweeper can detect the scam account in Fig. 9 is that it focuses not only on the graph structure but also on the sequential timestamp attributes. ScamSweeper takes the transaction graph as input, and we use the STRWalk to simplify the extensive transaction network. Moreover, as shown in Fig. 9(b), ScamSweeper uses STRWalk to sample the original transactions, reducing the number of transactions from 399 to 134. The transaction pattern we detected, shown in Fig. 11(b), is more simplified, and the pattern of 79 transaction numbers is detected from the 288 original transactions. It preserves their structural and temporal features while lowering the runtime overload, which helps to process large-scale transactions effectively.

TABLE IV: The Runtime Results of Temporal Analysis with ScamSweeper on Different Graph Structures. **Spl** stands for the simplified graph obtained by sampling from STRWalk, and @5 means that the structure window size is 5.

Graph Structure	Original	Spl@5	Spl@10
Time	29.50	4.6673	5.0454
F1-score	0.87	0.76	0.67

We perform a temporal analysis on 2,187 web3 scam accounts, which are randomly obtained from the dataset described at § V-A. We first get a simplified trading graph when STRWalk's window is set to 5 and 10. The original transactions are constructed as the initial transaction graph. Fig. 7 illustrates that ScamSweeper outperforms other methodologies when the structure windows are configured at 5 and 10. Thus, we present a detailed analysis of the time consumption associated with ScamSweeper across various structure windows. As shown in Table IV, ScamSweeper takes 4.6673 and 5.0454 seconds on the simplified transaction graph with Spl@5 and Spl@10, while it takes more than 29.50 seconds on the initial transaction graph. The results show that the ScamSweeper with STRWalk can complete the detection with less time overhead.

C. The Optimization of STRWalk

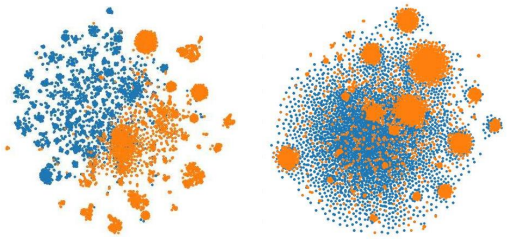
A series of procedures is conducted to assess the efficacy of STRWalk in transaction analysis. The findings, presented in Table V, provide a comparative analysis of the Node2Vec and DeepWalk methodologies applied to the original transaction network and the sampled network, which is constructed from the STRWalk algorithm, with `EPTransNet` [63]. We can see that the sampling algorithm helped by STRWalk has a performance improvement. Under the F1 score, Node2Vec can improve from 0.26 to 0.79 with the advantage of around 203.85%, and DeepWalk can improve from 0.21 to 0.80 with the advantage of about 280.95%.

TABLE V: The Comparison Results on Phishing Nodes Detection. † refers to the utilization of STRWalk.

Method	Precision	Recall	F1-score
Node2Vec [9]	0.63	0.16	0.26
DeepWalk [13]	0.58	0.13	0.21
Node2Vec† [9]	0.82	0.77	0.79
DeepWalk† [13]	0.82	0.79	0.80

D. The Temporal Setting for STRWalk

In Eq.1, we subtract the minimum value from each edge to determine the probability weight, ultimately obtaining P_i for random sampling in the first phase. It is based on the idea that transactions closer to their creation time are more important. To test this hypothesis, we perform an operation shown in Eq.11 that contrasts with Eq.1, using the difference between the maximum time and the time of each edge.



(a) ScamSweeper† (b) ScamSweeper¶

Fig. 13: The Embedding Visualization of Node Feature Distribution by T-SNE, where ■ means malicious accounts, and ■ represents normal accounts. The † and ¶ refer to the STRWalk with Eq.1 and Eq.11, respectively

$$P_i = \frac{\mu(T[i])}{\sum_{j=1}^{n} \mu(T[j])}, \qquad \mu(T[\ell]) = \mathrm{Max}(T) - T[\ell] + 1 \tag{11}$$

After embedding the operation in Eq.11 into STRWalk and training with ScamSweeper, we visualize malicious nodes and normal nodes using the t-SNE method in Fig. 13.

In Fig. 13, it is evident that although the majority of malicious nodes have been successfully clustered, there remains a notable overlap with normal nodes, presenting a significant challenge for effective segmentation.

E. Threat to Validity

Internal validity: (1) In the web3 scams dataset (§ V-B), there are 8,736,430 nodes in our dataset, while there are only 3,125 malicious nodes. To balance it, we randomly sample an equal number of non-malicious nodes as negative samples. **(2)** In the phishing dataset (§ V-B), we collect 4,905 phishing and 636 normal nodes. We leverage the 636 normal nodes and EPTransNet, which is a subset dataset with 1,165 real phishing nodes. Overall, we construct a 1165:636 positive and negative dataset ratio to alleviate the imbalance.

External Validity: All the experimental datasets utilized in our study are collected from the real world, **(1)** Our dataset in RQ1 (§ V-B), where both phishing and normal nodes are obtained in Etherescan through the labels in XlabelCloud [61]. **(2)** As for the datasets of web3 scams, they are real scams that have been manually audited by Scamsniffer before collecting their transaction networks in Ethereum. Thus, these datasets are deemed highly suitable for evaluation. Note that all nodes present within the network can be detected concurrently, if necessary. However, due to imbalance issues, not all negative samples are subjected to testing.

VII. Conclusion

Web3 services are widely distributed in the blockchain ecosystem, while many illegal accounts conceal malicious behavior when providing services. This paper proposes ScamSweeper, a detection model for detecting mimic accounts in web3 scams on Ethereum. We utilize the structure temporal random walk to sample the node network, which helps to alleviate the large-scale transaction network data. ScamSweeper can learn the sequence feature in the directed graph structure, exploring the dynamic evolution of time series. Through evaluating a real-world web3 scam dataset of transactions, ScamSweeper outperforms other methods by 15% in weighted F1-score. Moreover, ScamSweeper has an advantage of 17.5% in the F1-score on a phishing dataset.

VIII. Acknowledgments

This work is sponsored by the National Natural Science Foundation of China (No.62362021, No.62402146, and No.62032025), and CCF-Tencent Rhino-Bird Open Research Fund (No.RAGR20230115).

References

[1] W. Li, Z. Liu, X. Li, and S. Nie, "Detecting malicious accounts in web3 through transaction graph," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024, pp. 1–2.

[2] Y. Huang, X. Li, Y. Wu, C. Xue, J. Li, Y. Lin *et al.*, "Blockchain-based isotopic big data-driven tracing of global pm sources and interventions," *Nature Communications*, vol. 16, no. 1, p. 3901, 2025.

[3] Y. Yu, G.-P. Liu, Y. Huang, C. Y. Chung, and Y.-Z. Li, "A blockchain consensus mechanism for real-time regulation of renewable energy power systems," *Nature communications*, vol. 15, no. 1, p. 10620, 2024.

[4] H. Kim, J. Cui, E. Jang, C. Lee *et al.*, "Drainclog: Detecting rogue accounts with illegally-obtained nfts using classifiers learned on graphs," in *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, 2024, pp. 1–18.

[5] S. Ratra, M. Ghosh, N. Baliyan, J. Rashmitha Mohan, and S. Singh, "Graph neural network based phishing account detection in ethereum," *The Computer Journal*, vol. 67, no. 12, pp. 3160–3168, 2024.

[6] S. Zhang, W. Li, X. Li, and B. Liu, "Authros: Secure data sharing among robot operating systems based on ethereum," in *Proceedings of the IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, 2022, pp. 147–156.

[7] Y. Niu, X. Li, H. Peng, and W. Li, "Unveiling wash trading in popular nft markets," in *Companion Proceedings of the ACM Web Conference 2024*, 2024, pp. 730–733.

[8] W. Li, X. Li, Z. Li, and Y. Zhang, "Cobra: interaction-aware bytecode-level vulnerability detector for smart contracts," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024, pp. 1358–1369.

[9] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM international conference on Knowledge discovery and data mining (KDD)*, 2016, pp. 855–864.

[10] B. He, Y. Chen, Z. Chen, X. Hu, Y. Hu, L. Wu *et al.*, "Txphishscope: Towards detecting and understanding transaction-based phishing on ethereum," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023, pp. 120–134.

[11] S. Li, R. Wang, H. Wu, S. Zhong, and F. Xu, "Siege: Self-supervised incremental deep graph learning for ethereum phishing scam detection," in *Proceedings of the 31st ACM International Conference on Multimedia (MM)*, 2023, pp. 8881–8890.

[12] S. Li, G. Gou, C. Liu, C. Hou *et al.*, "Ttagn: Temporal transaction aggregation graph network for ethereum phishing scams detection," in *Proceedings of the ACM Web Conference (WWW)*, 2022, pp. 661–669.

[13] 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 (KDD)*, 2014, pp. 701–710.

[14] F. Poursafaei, R. Rabbany, and Z. Zilic, "Sigtran: signature vectors for detecting illicit activities in blockchain transaction networks," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2021, pp. 27–39.

[15] J. Wu, B. Huang, J. Liu, Q. Li, and Z. Zheng, "Understanding the dynamic and microscopic traits of typical ethereum accounts," *Information Processing & Management*, vol. 60, no. 4, pp. 103 384–103 400, 2023.

[16] J. Zhou, C. Hu, J. Chi, J. Wu *et al.*, "Behavior-aware account de-anonymization on ethereum interaction graph," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3433–3448, 2022.

[17] X. Li, T. Chen, X. Luo, and C. Wang, "Clue: towards discovering locked cryptocurrencies in ethereum," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC)*, 2021, pp. 1584–1587.

[18] X. Li, T. Chen, X. Luo, and J. Yu, "Characterizing erasable accounts in ethereum," in *Proceedings of the International Conference on Information Security (ISC)*, 2020, pp. 352–371.

[19] J. Coghlan, "Pink, pussy, venom, inferno," 2023, https://cointelegraph.com/news/cypto-drainers-pink-pussy-venom-and-inferno-steal-millions.

[20] Researchers, "$1.25 million stolen in nft airdrop phishing scam linked to inferno drainer," 2023, https://drops.scamsniffer.io/post/1-25m-stolen-due-to-nft-airdrop-phishing-on-polygon/.

[21] J. Liu, J. Chen, J. Wu, Z. Wu, J. Fang, and Z. Zheng, "Fishing for fraudsters: Uncovering ethereum phishing gangs with blockchain data," *IEEE Transactions on Information Forensics and Security*, 2024.

[22] J. Wu, D. Lin, Q. Fu, S. Yang, T. Chen *et al.*, "Towards understanding asset flows in crypto money laundering through the lenses of ethereum heists," *IEEE Transactions on Information Forensics and Security*, 2023.

[23] J. Yang, J. Liu, D. Lin, J. Wu, B. Huang *et al.*, "Who stole my nft? investigating web3 nft phishing scams on ethereum," *IEEE Transactions on Information Forensics and Security*, 2024.

[24] Etherscan, "The ethereum data explorer," 2025, https://etherscan.io/.

[25] W. Li, X. Li, Y. Zhang, and Z. Li, "Defitail: Defi protocol inspection through cross-contract execution analysis," in *Proceedings of the ACM Web Conference (WWW)*, 2024, pp. 786–789.

[26] S. Hu, Z. Zhang, B. Luo, S. Lu, B. He, and L. Liu, "Bert4eth: A pre-trained transformer for ethereum fraud detection," in *Proceedings of the ACM Web Conference (WWW)*, 2023, pp. 2189–2197.

[27] Z. Li, W. Li, X. Li, and Y. Zhang, "Guardians of the ledger: Protecting decentralized exchanges from state derailment defects," *IEEE Transactions on Reliability*, vol. 74, no. 3, pp. 3629–3641, 2024.

[28] X. Zhang, L. Hao, G. Gui *et al.*, "An automatic and efficient malware traffic classification method for secure internet of things," *IEEE Internet of Things Journal*, vol. 11, no. 5, pp. 8448–8458, 2023.

[29] W. Chen, X. Guo, Z. Chen, Z. Zheng, and Y. Lu, "Phishing scam detection on ethereum: Towards financial security for blockchain ecosystem." in *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, 2020, pp. 4456–4462.

[30] S. Farrugia *et al.*, "Detection of illicit accounts over the ethereum blockchain," *Expert Systems with Applications*, vol. 150, pp. 113 318–113 329, 2020.

[31] Q. Yuan, B. Huang, J. Zhang, J. Wu *et al.*, "Detecting phishing scams on ethereum based on transaction records," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[32] 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*, vol. 21, no. 1, pp. 1–16, 2020.

[33] S.-H. Choi and S.-J. Buu, "Learning to traverse cryptocurrency transaction graphs based on transformer network for phishing scam detection," *Electronics*, vol. 13, no. 7, p. 1298, 2024.

[34] W. Chen, X. Guo, Z. Chen *et al.*, "Honeypot contract risk warning on ethereum smart contracts," in *Proceedings of the IEEE International Conference on Joint Cloud Computing (JCC)*, 2020, pp. 1–8.

[35] H. Hu, Q. Bai, and Y. Xu, "Scsguard: Deep scam detection for ethereum smart contracts," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2022, pp. 1–6.

[36] Z. Zheng, W. Chen, Z. Zhong *et al.*, "Securing the ethereum from smart ponzi schemes: Identification using static features," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–28, 2023.

[37] Q. Huang, M. Yamada *et al.*, "Graphlime: Local interpretable model explanations for graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 1–15, 2022.

[38] F. Cernera, M. La Morgia *et al.*, "Token spammers, rug pulls, and sniper bots: An analysis of the ecosystem of tokens in ethereum and in the binance smart chain (*bnb*)," in *Proceedings of the 32nd USENIX Security Symposium (USENIX Security)*, 2023, pp. 3349–3366.

[39] Q. Kong, J. Chen, Y. Wang, Z. Jiang, and Z. Zheng, "Defitainter: Detecting price manipulation vulnerabilities in defi protocols," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023, pp. 1144–1156.

[40] M. Varun, B. Palanisamy, and S. Sural, "Mitigating frontrunning attacks in ethereum," in *Proceedings of the 4th ACM International Symposium on Information, Computer and Communications Security (ASIA-CCS)*, 2022, pp. 115–124.

[41] S. Hu, Z. Zhang, S. Lu, B. He, and Z. Li, "Sequence-based target coin prediction for cryptocurrency pump-and-dump," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–19, 2023.

[42] M. La Morgia, A. Mei, F. Sassi, and J. Stefa, "The doge of wall street: Analysis and detection of pump and dump cryptocurrency manipulations," *ACM Transactions on Internet Technology*, vol. 23, no. 1, pp. 1–28, 2023.

[43] D. Buterez, J. P. Janet, D. Oglic, and P. Liò, "An end-to-end attention-based approach for learning on graphs," *Nature Communications*, vol. 16, no. 1, p. 5244, 2025.

[44] Y. Yang, Y. Cui, X. Zeng, Y. Zhang, M. Loza, S.-J. Park, and K. Nakai, "Staig: Spatial transcriptomics analysis via image-aided graph contrastive learning for domain exploration and alignment-free integration," *Nature Communications*, vol. 16, no. 1, p. 1067, 2025.

[45] Y. Kong, G. Shi, R. Wu, and Y. Zhang, "k-core: Theories and applications," *Physics Reports*, vol. 832, pp. 1–32, 2019.

[46] Z. Ma *et al.*, "Community detection with contextual multilayer networks," *IEEE Transactions on Information Theory*, vol. 69, no. 5, pp. 3203–3239, 2023.

[47] J. Kauffman, E. Holmes, A. Vaid, A. W. Charney, P. Kovatch, J. Lampert, A. Sakhuja, M. Zitnik, B. S. Glicksberg, I. Hofer *et al.*, "Infehr: Clinical phenotype resolution through deep geometric learning on electronic health records," *Nature Communications*, vol. 16, no. 1, p. 8475, 2025.

[48] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017, pp. 1–14.

[49] Y. Pan, Z. Xu, L. T. Li, Y. Yang, and M. Zhang, "Automated generation of security-centric descriptions for smart contract bytecode," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023, pp. 1244–1256.

[50] J. G. Carvajal-Patiño, V. Mallet, D. Becerra, L. F. Niño Vasquez, C. Oliver, and J. Waldispühl, "Rnamigos2: accelerated structure-based rna virtual screening with deep graph learning," *Nature Communications*, vol. 16, no. 1, pp. 1–12, 2025.

[51] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.

[52] Z. Wu, J. Liu, J. Wu *et al.*, "Know your transactions: Real-time and generic transaction semantic representation on blockchain & web3 ecosystem," in *Proceedings of the ACM Web Conference (WWW)*, 2023, pp. 1918–1927.

[53] ——, "Tracer: Scalable graph-based transaction tracing for account-based blockchain trading systems," *IEEE Transactions on Information Forensics and Security*, pp. 2609–2621, 2023.

[54] P. Veličković, G. Cucurull, A. Casanova, A. Romero *et al.*, "Graph attention networks," in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018, pp. 1–12.

[55] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez *et al.*, "Attention is all you need," in *Proceedings of the Advances in neural information processing systems (NIPS)*, 2017, pp. 1–11.

[57] Z. Zeng, R. Kaur, S. Siddagangappa, S. Rahimi *et al.*, "Financial time series forecasting using cnn and transformer," 2023, pp. 1–4.

[58] Scamsniffer, "Web3 scams database," 2024, https://github.com/scamsniffer/scam-database.

[59] L. Wang, M. Xu, and H. Cheng, "Phishing scams detection via temporal graph attention network in ethereum," *Information Processing & Management*, vol. 60, no. 4, pp. 103 412–103 431, 2023.

[60] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 1–11.

[61] XBlock, "Xlabelcloud," 2024, https://www.xblock.pro/cloud.

[62] D. Lin, J. Wu *et al.*, "Modeling and understanding ethereum transaction records via a complex network approach," *IEEE Transactions on Circuits and Systems*, vol. 67, no. 11, pp. 2737–2741, 2020.

[63] EPTransNet, "Eptransnet," 2023, https://xblock.pro/##/dataset/13.

[64] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, pp. 1–27, 2008.

[65] S. Chen, S. Wei, C. Liu, and W. Yang, "Dycl: Dynamic neural network compilation via program rewriting and graph optimization," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023, pp. 614–626.

[66] Researcher, "$5.9 million stolen by scam as a service provider called inferno drainer," 2023, https://drops.scamsniffer.io/post/5-9-million-stolen-by-scam-as-a-service-provider-called-inferno-drainer/.

[67] ——, "Etherscan," 2023, https://etherscan.io/address/0x0169a0ab7443bb52f3d16cbd371a10eed335b3a5.