Expert Systems With Applications 298 (2026) 129919

Contents lists available at ScienceDirect

Expert Systems With Applications

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

PDHG: An Ethereum phishing detection approach via heterogeneous graph transformer

Lei Wang*, Yihan Mi, Yanan Zhang, Jialin Zhang

Department of Management Science and Engineering, and Data Science and Systems Science (DTripleS) Lab, Nanjing Forestry University, Longpan Road 159, Nanjing, 210037, China

ARTICLE INFO

Keywords:
Ethereum
Phishing scams detection
Heterogeneous graph transformer
Explainer

ABSTRACT

Phishing scams have emerged as a significant threat within the Ethereum ecosystem. Cutting-edge Ethereum phishing scams detection techniques mostly treat accounts in Ethereum as homogeneous nodes in transaction graphs. Existing detection approaches model Ethereum transaction records as homogeneous transaction graphs and use graph representation learning for account classification. However, those approaches often overlook the heterogeneity between accounts and transactions, making it difficult to capture the diversity of interactions and features among accounts. In this paper, a heterogeneous graph transformer (HGT)-based phishing account identification approach called PDHG is proposed. Specifically, PDHG models the transaction network between accounts as a heterogeneous graph based on different attributes of Ethereum accounts, allowing for a more comprehensive description of the structure and behavioral patterns of the transaction network. To enhance the explainability, PDHG leverages PDHGexplainer as the explainer for the detection results. We compare PDHG with other existing detection models. The experimental results demonstrate that PDHG achieves an AUC score of 96.04 % and a recall score of 89.87 %, surpassing the state-of-the-art approaches.

1. Introduction

According to the 2024 Security Annual Report released by OKLink[1], on-chain security incidents worldwide caused approximately $1.945 billion in losses in 2024. Phishing scams accounted for the largest share of these losses, reaching $705 million, or 36 % of the total. Therefore, identifying the phishing accounts and those behavioral characteristics of phishing attacks and scams on Ethereum has become an important research topic (Hou et al., 2025; Tang et al., 2024; Wu et al., 2022).

Let us consider a cryptocurrency exchange interacting with merchant wallets and a large population of user accounts on Ethereum. If a small group of attacker-controlled addresses begins to impersonate a trusted service, they quickly funnel funds through token contracts and a set of high-volume intermediary addresses. Such abnormal transaction pattern (i.e., phishing scam) can develop within just a few hours. If not detected and investigated promptly, it may lead to substantial financial losses. Moreover, for practical Ethereum phishing detection, it is essential not only to achieve fast and accurate identification of suspicious accounts, but also to provide concise, human-understandable evidence that supports operational decisions for regulatory authorities.

Based on the technical foundations of mainstream Ethereum phishing detection, we can classify them into three main approaches. The first approach primarily relies on traditional machine learning techniques, as presented in Abdelhamid et al. (2014), focusing on basic statistical features such as account or transaction features, in-degree, and out-degree to assess the similarity between accounts. However, these methods overlook the complex topological structure of the Ethereum transaction network, potentially limiting their effectiveness in phishing detection. The second method is an end-to-end deep learning based phishing detection method for Ethernet networks. Although this method is able to automatically extract features and is highly adaptive, it still faces problems such as difficult data labeling, poor interpretability and limited generalisation capability (Ayllón-Gavilán et al., 2025; Wang et al., 2025b, 2021; Wen et al., 2023). In particular, the third approach adopts graph representation learning methodologies, conceptualizing Ethereum account transactions as a transaction network. Those approaches employ random walk algorithms (Hou et al., 2022; Li, 2023) and graph neural network models (Lin et al., 2024; Wang et al., 2025a). Graph representation learning approaches have made significant advances in recent years, accurately and efficiently uncovering fraudulent accounts within

* Corresponding author.
E-mail addresses: leiwang@njfu.edu.cn (L. Wang), miyihan@njfu.edu.cn (Y. Mi).

[1] http://oklink.com/

https://doi.org/10.1016/j.eswa.2025.129919
Received 22 June 2025; Received in revised form 17 September 2025; Accepted 28 September 2025
Available online 1 October 2025

the Ethereum ecosystem. We identify the following three key challenges that remain in the application of Ethereum phishing scam detection.

1) Information Heterogeneity. The main forms of Ethereum phishing scams include impersonating exchanges, disguising as ICOs, creating fake trading websites, and forging smart contracts. Cryptocurrencies are primarily exchanged for cash through several methods, including cryptocurrency trading platforms, using cryptocurrency debit cards, Peer-to-Peer (P2P) trading, and cryptocurrency exchange counters. These activities usually involve addresses such as wallets, exchange platforms and token contracts. Therefore, we believe that fully exploring heterogeneous information between accounts, wallets, exchanges, and token contracts in the Ethereum transaction graph is more effective in detecting phishing accounts than modeling the Ethereum transaction graph as a homogeneous graph.

2) Operational Efficiency. Homogeneous graph models for fraud detection rely on the expertise in rule creation, feature engineering. Similarly, traditional GCN models based on a full-view approach require a large amount of computational resources and lengthy training time. Furthermore, most heterogeneous graph models rely on meta-path design. Both of these approaches suffer from poor scalability and are time-consuming and labor-intensive. While some advanced heterogeneous models like RGCN (Attar et al., 2024) can be computationally intensive.

3) Explainability. First, model risk management is of critical importance to financial regulators, who highlight errors within models and their potential misuse as primary concerns. Black-box models obscure errors, increasing operational risks. Improving explainability enables better performance of fraud detection, reducing operational risks associated with model misuse. Second, model decisions significantly influence financial institutions' business decisions. Explainability enhances trust and credibility in model outcomes, empowering decision-makers to understand, assess, and validate model rationale. Public trust is vital for financial institutions. Lack of explainability, leading to erroneous actions and user losses, can damage public trust. Enhancing the explainability in decision-making can foster trust and bolster an institution's reputation.

This paper tackles the challenges of phishing scam detection from a heterogeneous graph-centric standpoint. Different from the conventional approaches (Ayllón-Gavilán et al., 2025; Jin et al., 2025; Sui et al., 2025), this graph enables us to capture intricate heterogeneous relationships and foster more descriptive representations of the nodes by integrating multiple node types including accounts, exchanges, and token contracts. In addition, the detector aggregates information from various node types through different pathways, eliminating the need for manual specification of meta-paths. This is crucial as it negates the requirement for pre-defining risk propagation meta-paths or pre-processing path representations, as required by prior works (Huang et al., 2023a; Zhang et al., 2024; Zhong et al., 2020). Moreover, in this work, an integrated explainer module that concatenates the feature mask and neighborhood mask is introduced. It not only considers the local feature contribution, but also combines the global neighborhood information to ensure stable and reliable interpretations.

To the best of our knowledge, this is the first work that applies a self-attentive Heterogeneous Graph Transformer (HGT) with an interpreter to phishing detection on the Ethereum blockchain. The contributions of this work are summarized as follows.

- We present PDHG, an approach for detecting Ethereum phishing scams. PDHG constructs a heterogeneous transaction network using publicly available transaction data within Ethereum. PDHG utilizes the extracted temporal heterogeneous Ethereum transaction network information throughout the node representation process. It enriches node representations via a heterogeneous network representation learning approach, i.e., HGT with self-attentive mechanism, which integrates node embedding, edge embedding, and attribute embedding.
- We integrate the explainers into our framework and propose PDHG-explainer to provide the intuitive explanations for the predictions of PDHG detector. We provide a quantitative evaluation of PDHG-explainer. This integration equips auditors, regulators, and decision-makers with the necessary insights into how the transaction detector labels accounts, enabling them to make more informed and well-reasoned decisions.
- We conduct a comprehensive evaluation based on three collected Ethereum phishing scam datasets. The results show that PDHG outperforms state-of-the-art methods on a variety of metrics. Heterogeneous weight parameterization contributes significantly to improving detection effectiveness, while temporal modeling is crucial for accurately identifying network phishing accounts.

The rest of this paper is organized as follows. Section 2 presents formal definitions of the problems to be investigated in this paper. In Section 3, we present the architecture and the approach of PDHG in detail. Experiments and the results analysis are shown in Section 4. Section 5 summarizes the related works. Finally, we draw a conclusion in Section 6.

2. Problem statement

Let $G = (V, E)$ denote the heterogeneous graph representing the Ethereum transaction network, where V represents the set of nodes and E denotes the set of edges. Each node v is categorized into a specific type $t(v)$, indicating varying types of accounts. The phishing scams detection task is defined as a node classification problem within this heterogeneous graph, where nodes are categorized into one of two classes, i.e., phishing accounts or legitimate accounts.

To classify different nodes, we employ Heterogeneous Graph Transformer (HGT) model, which learns information propagation among nodes for classification. In the HGT model, each node v is characterized by a feature vector x_v encompassing the node's inherent features and information concerning its neighboring nodes. This information dissemination among neighboring nodes occurs across various types of edges.

The HGT model leverages the Transformer network architecture to grasp information propagation among nodes. Specifically, for each node v, the HGT model aggregates insights from its neighboring nodes via multiple layers of transformer encoders. This aggregated information is then fused with the node's own features to generate the node's representation. Subsequently, the representation x_v of each node is inputted into a classifier to predict the node's class label.

The mathematical definition of the heterogeneous graph node classification problem based on the HGT model for Ethereum phishing scam detection is:

$$\arg \max_{\{y_v\}_{v\in V}} \sum_{v\in V} \mathcal{L}(g(h(x_v, \{x_u\}_{u\in N(v)}, t(v))), y_v), \tag{1}$$

where h represents the HGT model for encoding node features, g is the classifier for predicting node class labels, and $\mathcal{L}$ is the loss function, typically chosen as cross-entropy loss or another suitable loss function for classification tasks. Our goal is to find the node label assignment that minimizes the loss function, thereby achieving phishing scams detection in the Ethereum transaction network.

To enhance the interpretability of the detection result, we propose PDHGexplainer in this paper. The transaction graph node explanation task in this paper can be formalized as follows.

Given a specific node v with its k-hop neighborhood subgraph $G = (V_v, E_v, X_v)$, where the graph neural network classification model is denoted as $f : V \to Y$, and the label for node v is y_v, the matrix X represents the feature vectors of the nodes within the k-hop neighborhoods of v. Assuming that there are N nodes (including node v itself) in the k-hop neighborhoods, $X_v = [x_1, x_2, \dots, x_N]$, where x_i is the feature vector of node i, $x_i = [a_{i1}, a_{i2}, \dots, a_{iF}]$, and a_{ij} represents the j-th dimension feature value of node i. E_v denotes the edge set in the transaction network among these N nodes.

Additionally, given the set of node indices within the k-hop neighborhood $n = \{0, 1, \dots, N-1\}$ (where index 0 represents the node v itself), the maximum number C of important features or nodes that the explainer can recognize is defined. The set of indices $Q = \{q_1, q_2, \dots, q_s\}$, where $q_i \in [1, F + N - 1]$ and $s \le C$, represents the recognized important features or nodes.

Moreover, the transaction graph node attribution task in this paper is formalized as follows. Given the k-hop neighborhood subgraph $G = (V_v, E_v, X_v)$ of node v, the label y_v, and the graph neural network classification model $f : V \to Y$, the explainability method of the graph neural network is employed to attribute features of node v and its k-hop neighborhood nodes, resulting in a set $Q = \{q_1, q_2, \dots, q_s\}$ containing not more than C elements. This set is chosen to optimize the classification model's recognition performance by using only the features of nodes in Q and their neighborhood nodes, aiming to approximate the performance achieved when using the complete set of node features and neighborhood nodes.

3. PDHG approach

3.1. Overview of PDHG

The overall architecture of PDHG is illustrated in Fig. 1. It comprises five main components: the construction of the transaction graph module, the subgraph sampling module, the feature engineering module, the detector module, and the explainer module.

First, the initial step involves harnessing the vast amount of transaction data originating from Ethereum to construct a comprehensive Ethereum transaction graph. Within this intricate graph, individual nodes represent distinct accounts, while edges delineate the transactions occurring between them.

Second, due to the immense size of the initial transaction graph, we utilize the random walk approach for subgraph sampling. Our process commences with the random selection of a node from the graph, followed by sequentially selecting its neighboring nodes as the subsequent steps, iterating this sequence until the desired number of neighboring nodes is achieved. This iterative approach effectively enables us to extract a transaction subgraph of the desired size and complexity, thereby balancing data representation with computational efficiency and storage requirements.

Third, due to the anonymity of Ethereum transactions, the data we collected can be considered as weighted directed edges, lacking inherent information about node portraits. Therefore, we perform feature engineering on transaction information between nodes to obtain node features.

Fourth, the classifier model f is trained utilizing the node feature matrix X, the graph adjacency matrix A, and the pre-labeled node label set Y.

Finally, we employ a newly proposed mask generation algorithm in the mask generator to construct feature mask vector M_F and neighborhood node mask vector M_N. Through the graph generator, the newly generated graphs, manipulated with the masks, are mapped into the vector space of f, resulting in the newly generated node features X' and the newly generated adjacency matrix A'. Then, using the classifier model f, we calculate the new classification result $f(V', A', T', X')$. By constructing a dataset with the mask vector Z and the classification result $f(V', A', T', X')$ under the influence of the mask, we train a white-box linear model to fit the data. By obtaining the weight parameters, we filter out important features or neighboring nodes based on the weights, and output the index set Q as an explanation.

3.2. Constructing temporal heterogeneous transaction graph

This section explains how to construct a heterogeneous transaction graph utilizing a substantial amount of transaction data gathered from the Ethereum transaction network.

Similar to traditional transaction graphs, in the Ethereum transaction graph, nodes represent accounts, and edges depict transactions between these accounts. However, Ethereum's raw transaction data presents several challenges for heterogeneous graph modeling.

First, since transactions occur over time, the edges in the transaction graph are timestamped. This necessitates preserving temporal transaction information. PDHG incorporates time information by mapping it onto a sinusoidal curve, encoding it in a continuous and smooth manner. Specifically, for each time step t, d denotes the dimension of the time encoding, and its corresponding positional encoding $\boldsymbol{\phi}(t)$ is defined as a vector containing multiple sine functions as:

$$\boldsymbol{\phi}(t) = \left[\sin(\omega_1 \cdot t), \sin(\omega_2 \cdot t), \dots, \sin(\omega_d \cdot t)\right], \tag{2}$$

where ω_i represents the frequency of the sine function, typically set as:

$$\omega_i = 2\pi \cdot \frac{i}{d}. \tag{3}$$

Second, the Ethereum transaction graph is extremely large and sparse with millions of addresses and billions of transactions. Building a

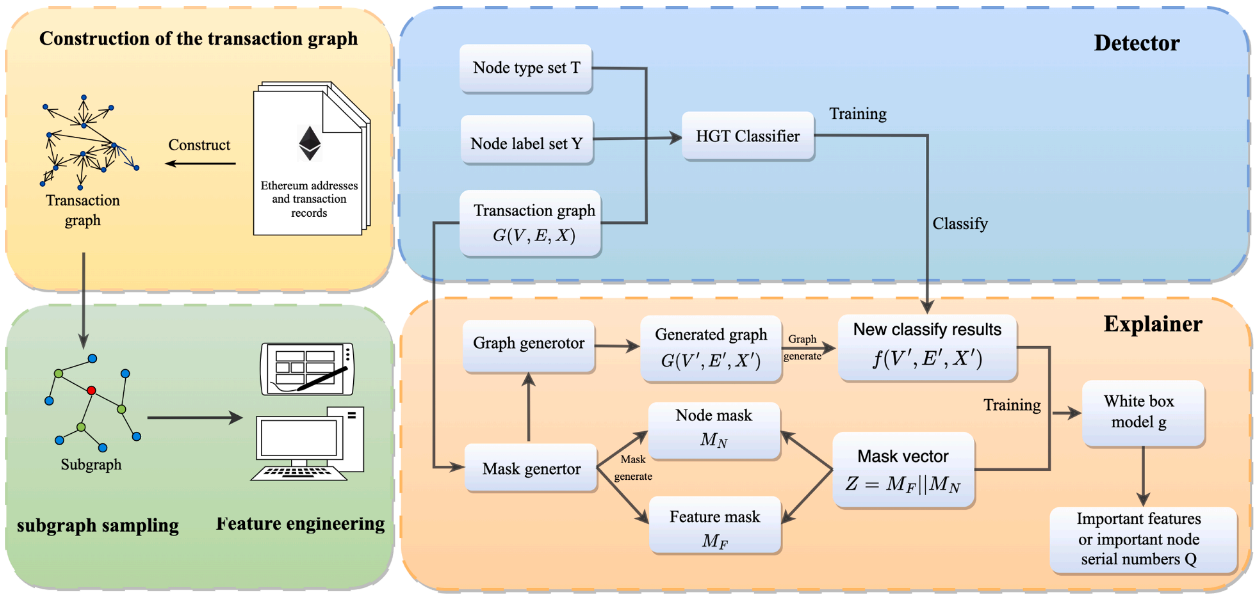

Fig. 1. The overall architecture of PDHG.

full hetero-graph is impractical. PDHG addresses this via subgraph sampling. This technique involves randomly selecting a starting node and subsequently choosing a neighbor at random, repeating this until we have gathered the desired number of neighboring nodes.

Third, Ethereum addresses lack inherent labels, making it impossible to distinguish wallets, exchanges, or contracts by default. To resolve this, we enhance our data by retrieving node role details from OKLink. While this website offers hundreds of labels to characterize account node types, we selectively utilize labels pertinent to phishing transactions, including exchanges, token contracts, exchange accounts, wallets, and other malicious accounts. For remaining accounts, we label them as “account”. These labels are then incorporated as node attributes into the transaction graph to represent the roles of nodes. While the majority represent regular accounts, a subset represents various types, such as token contracts, exchanges, etc. Although these account types may not have a direct relevance to phishing scam detection, as they cannot be labeled as “fraudulent accounts”, we hypothesize that they can still provide valuable insights into suspicious activities surrounding transactions and accounts. For instance, fraudsters might leverage channels like exchanges, wallets, or token contracts for money laundering or maintain numerous accounts for illicit activities.

3.3. Subgraph sampling

This section describes the subgraph sampling strategy adopted to improve scalability, as well as sampling parameters.

When constructing our detection model, we initially adopt the native implementation of Heterogeneous Graph Transformer (HGT), but find that HGSampling incurs significant computational cost. Thus, we borrow from GraphSAGE’s sampling strategy.

We develop an enhanced sampling method that initiates by sampling a node’s k-hop neighborhood and then integrates feature information from its neighboring nodes. In contrast to the HGSampling strategy employed in Heterogeneous Graph Transformer (HGT), our approach does not strictly adhere to maintaining a similar distribution of node and edge types after sampling. While HGSampling aims to minimize information loss and sample variance by preserving these distributions, the sparsity of the Ethereum transaction graph necessitates a more efficient approach. Maintaining strict similarity in node and edge types in the sampled subgraph incurs significant computational overhead in the Ethereum context.

Please note that the choice of k-hop neighbor sampling is crucial, especially in complex Ethereum transaction networks. Smaller k-values (e.g., $k=1$ or $k=2$) are suitable for capturing local relationships, fitting simpler transaction patterns, while larger k-values (e.g., $k=3$ or $k=4$) allow the model to capture deeper layers of node interactions, revealing more complex phishing behaviors. These deeper relationships, including transactions between multiple accounts and cross-account money flows, are often indicative of phishing accounts. Increasing the k-value improves the model’s ability to detect complex behaviors but also increases computational complexity and memory overhead. Therefore, we optimize the k-value through hyperparameter tuning, ensuring the model’s accuracy while avoiding excessive computational resource consumption.

3.4. Feature engineering

To prepare for subsequent node embedding learning, we engage in feature engineering on the gathered subgraph data.

Given the anonymity inherent in blockchain platforms, individual nodes lack personally identifiable information. Hence, feature engineering relies on extracting and amalgamating basic insights from node relationships. On-chain forensic practice and prior works (Chen et al., 2021a; Tharani et al., 2024) indicate that transfer amounts, timing and local connectivity are primary on-chain signals for scams, so we include features that directly quantify these aspects. Node attributes include: 1) Node labels (exchange, token-contract, wallet, malicious, or default account) obtained from OKLink and attached to each address; and 2) Behavioral features including 14 hand-engineered statistics per node: Total node degree, Out-degree, In-degree, In-degree ratio, Out-degree ratio, Total transaction amount, Outgoing transaction amount, Incoming transaction amount, Difference between incoming and outgoing transaction amounts, Ratio of incoming to outgoing amounts, Proportion of neighbors’ incoming transactions, Ratio of outgoing neighbors, Total number of neighbors, and Reciprocal of transaction frequency. For edge features, our attention is directed towards three crucial factors: transaction direction, amount, and timestamp.

3.5. Detector

This section provides a detailed introduction to the detector, especially the hierarchical structure and message passing mechanism of self-attention HGT.

The foundation of our detector lies in processing a heterogeneous graph as input. This graph encompasses characteristics of both the intended or target nodes X_{v_t} and the originating or source nodes X_{v_s}. Additionally, it accounts for the specific types of these nodes $\tau(v_t)$, $\tau(v_s)$, as well as the types of connecting edges $\phi(e)$. In the case of account nodes, their features are defined by a unique company risk identifier. The other node varieties, however, start with unpopulated features and only acquire inputs after the initial convolutional layer has been applied. To represent these various types, we utilize a one-hot encoding system.

Specifically, the feature vector for each node type is represented as a one-hot vector, where

$$X_{v_t} = [1, 0, 0, \dots,], X_{v_s} = [0, 0, 0, \dots,], \tag{4}$$

These vectors represent the different node types, where the 1 corresponds to the specific type of node.

The heterogeneous graph is then subjected to L layers of heterogeneous convolution, each equipped with a self-attention mechanism. The inaugural layer, denoted as $L^{(0)}$, integrates transactional data, embeddings for node types (pertaining to both source and target nodes), and embeddings for edge types. Afterward, these elements undergo a transformation into query, key, and value vectors as

$$Q = W_q X, \quad K = W_k X, \quad V = W_v X, \tag{5}$$

where W_q,W_k,W_v are the learnable weight matrices for the query, key, and value respectively, and X is the input feature matrix.

Subsequently, the self-attention scores are calculated for both the source and target nodes, culminating in a normalization process that is executed layer by layer.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{6}$$

where d_k is the dimensionality of the key vectors. This step computes the attention weights, which are then used to update the representations of the nodes.

The resulting output is then passed through a ReLU activation function:

$$Relu(x) = max(0, x). \tag{7}$$

It paves the way for subsequent convolutional layers.

Specifically, unlike GAT, which only considers the attention calculation method of neighbor nodes, Eq. (8) used in HGT also considers the type of edge connecting nodes u and v when calculating the attention value of a node. Similarly, $MSG-head^i(u, e, v)$ is the output of the i-th head in the message passing step, and $M-Linear_{\phi(u)}$ is the output of the $i-th$ head according to the type of node u. The $l-1$ layer representation is mapped to $H^{(l-1)}(u)$ to form an information vector.

Considering that two types of nodes may be connected by multiple types of edges, the information vector also needs to be transformed by

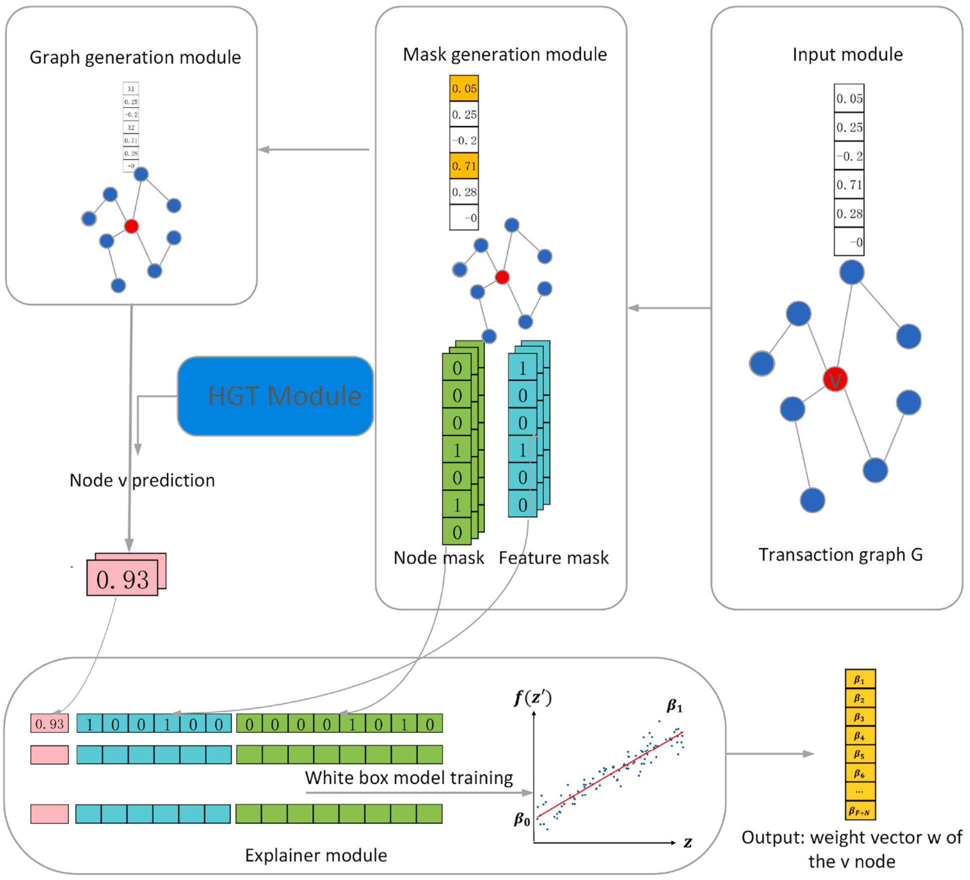

Fig. 2. Framework of PDHGexplainer.

the matrix $W^{MSG}_{\psi(e)}$ as:

$$\text{Message}_{HGT}(u, e, v) = \underset{i\in[1,h]}{||} MSG - head^i(u, e, v)$$
$$MSG - head^i(u, e, v) = \text{M} - \text{linear}_{\phi(u)}(H(l-1)(u))W^{MSG}_{\psi(e)}. \tag{8}$$

Specifically, each source node u of type $\varphi(u)$ is processed by a type-specific linear projection M-linear$_{\varphi(u)}$, and each edge of type $\psi(e)$ has its own weight matrix W$^{MSG}_{\psi(e)}$. As a result, the i-th message head is computed by first applying M-linear$_{\varphi(u)}$ to the previous-layer representation $H(l-1)(u)$ and then multiplying by W$^{MSG}_{\psi(e)}$. Likewise, the attention score Attention$_{HGT}(u, e, v)$ is made type-aware: the features of u and v are projected by node-type-specific matrices, and their similarity is modulated by the learned matrix for $\psi(e)$.

When aggregating the neighbor node information, firstly, the feature vectors of the neighbor nodes are weighted and summed according to the size of the attention value as:

$$\tilde{H}^{(l)}(v) = \underset{\forall u\in N(v)}{\oplus} (\text{Attention}_{HGT}(u, e, v) \cdot \text{Message}_{HGT}(u, e, v)). \tag{9}$$

Then, $\tilde{H}^{(l)}(v)$ is mapped to the representation space of the node type corresponding to the node v is located, and finally combined with the representation space of the $l-1$ layer of node v, to get the representation vector output $\tilde{H}^{(l)}(v)$ of the l layer as:

$$H^{(l)}(v) = \sigma(A-\text{linear}_{\phi(v)}(\tilde{H}^{(l)}(v))) + H^{(l-1)}(v). \tag{10}$$

Once the graph data traverses the L convolutional layers, a *tanh* activation function is applied to refine the transactional representations derived from the Graph Neural Network (GNN):

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{11}$$

These enhanced representations are then combined with the original transactional features and fed into a feedforward neural network comprising two hidden layers. The network computes the output as:

$$\hat{y} = softmax(W_{out}Relu(W_2Relu(W_1X + b_1) + b_2) + b_{out}), \tag{12}$$

where W_1, W_2 and W_{out} are weight matrices, b_1, $b_2$and b_{out} are biases. During this stage, dropout, layer normalization, and ReLU activation are employed to compute the predictive risk scores and their corresponding labels.

The detector's performance is evaluated using a loss function that quantifies the cross-entropy between the true label and the probability score, obtained by the softmax function:

$$\mathcal{L} = \sum_{i=1}^{N} y_i \log(\hat{y}_i), \tag{13}$$

where N is the number of classes, y_i is the true label, and $\hat{y}_i$ is the predicted probability for the class. The loss function calculates the discrepancy between the predicted and actual risk labels, guiding the model's optimization.

3.6. PDHGexplainer

In this section, we train a white-box linear model to fit the classification results of the black-box graph neural network model. By concatenating the feature mask M_F and neighborhood mask M_N of node v, we obtain $z = M_N||M_F$ as the sample feature. The new classification probability value $f_v(V', A', X')$ of the perturbed dataset of node v is used as the sample y value.

The framework of the explainer module is illustrated in Fig. 2. We transform the node interpretation task into a linear regression problem of finding the model weight parameters w given known sample features z and sample y values ($y \in [0, 1]$). Therefore, the explainer module is continuously trained on the perturbed dataset $\{z, f(z')\}$ to learn the weight vector of the dimensions of z as a model explanation. Compared with traditional GNNExplainer and PGExplainer, our interpreter not only considers the local feature contribution, but also combines the global neighborhood information to ensure the stability and reliability of the interpretation results. In particular, for regulators, it provides concise and interpretable evidence rather than opaque risk scores.

3.6.1. Mask generation module

To highlight the most relevant subgraph structure, this module details how masks over nodes and edges are dynamically created from the detector's outputs.

In the mask selection process, instead of soft masks, PDHGexplainer employs two discrete node feature masks, i.e., $M_F \in \{0,1\}^F$ and $M_N \in \{0,1\}^N$, to store mask information. This avoids the fuzzy activations that can introduce randomness. In addition, PDHGexplainer applies Feature Normalization to stabilize the mask weights. It standardizes feature scales so that the mask learning is not driven by wildly differing magnitudes, which reduces noise in attribution. Thus, we denote z as the sampled feature set, where $z = (M_F || M_N)$. The generated masks are then fed into the graph module to reconstruct perturbed subgraphs.

PDHGexplainer also incorporates several techniques to ensure stable and reliable explanations. We adopt a two-step mask optimization strategy. A Self-Attention mechanism is used to assign preliminary weights to features and neighborhoods. The mask is then dynamically adjusted during training to preserve features and nodes that contribute more to classification.

To minimize computational complexity, this paper divides the original integrated consideration of the joint impact of the node v feature subset and the k-hop neighborhood nodes subset on the classification results into two subsets: 1) Considering only the node features. This allows us to measure the impact of each node feature on the classification results. 2) Considering the k-hop neighborhood nodes. This enables us to evaluate the contribution of neighboring nodes to the classification outcome.

3.6.2. Graph generation module

In the graph generation module, our goal is to quantify the marginal contribution of the sampled feature set z to the original prediction results. This process determines the importance of this feature set for classification outcomes. Hence, We define the graph generation function $Gen(V, A, X, M_F, M_N) \to (V', A', X')$, which maps the node feature matrix, the adjacency matrix and the mask matrix into the original vector space. Unlike GNNExplainer, we then design a dual mask strategy to isolate individual impacts of node features and neighborhood nodes on classification. Moreover, to further optimize the interpreter's performance, we also add a Global Mean Imputation strategy to fill in unselected feature values, reducing the instability caused by feature masking.

For node features, unsampled dimensions are replaced by the global mean vector $\mu = [\mathbb{E}(a_{*1}), \cdots, \mathbb{E}(a_{*F})]^T$, yielding:

$$\forall x_i \in X_v, \quad x_i' = M_F \odot x_i + (1 - M_F) \odot \mu. \tag{14}$$

Here, M_F is the node feature mask and $\odot$ denotes element-wise multiplication. The formula's physical meaning is that the new node feature replaces unsampled features with global means while retaining original values of sampled features.

For neighborhood structure, the adjacency matrix is reconstructed by masking edges from unsampled neighbors via:

$$(V', A') = (1 - M_N') \odot I(V, A), \tag{15}$$

where $I(\cdot)$ is the indicator function identifying valid k-hop edges. We set the edges between nodes outside the k-hop neighborhood of node v to be zero in the adjacency matrix. M_N' is the node mask matrix transformed by the node mask vector M_N of the k-hop neighborhood subgraph of node v.

In fact, the results (V', A', X') generated by the graph generation function consist of two parts, i.e., (V, A, X') and $(V', A', \bar{X})$. The former only considers the influence of node v's features while keeping the neighborhood structure unchanged. The latter fills all nodes with means, i.e., $M_F = \vec{0}$, changing only the neighborhood structure. Thus, through this method, the tasks of feature-level interpretation and graph-level interpretation in GNN are cleverly combined.

3.6.3. Explainer module

In the explanation module, we aim to train a surrogate regression model on mask-perturbed data and output human-interpretable evidence for the detector's decisions.

The output $z' = (V', A', X')$ of the graph generation module is passed to the GNN model f, and the embedding representation of each sample $(z', f(z'))$ is stored. Based on the traditional linear Regression model, the explainer fits a Locally Weighted Regression (LWR), instead of plain Least Squares Regression, to the perturbed data, which further improves the nonlinear adaptability of the interpretation results. We define the explanation model $g(\cdot)$ on the dataset $D = (z', f(z'))$. Then, the explanation function of the GNN model can be represented as follows:

$$f(z') = \beta_0 + \beta_1 Z_1 + \beta_2 Z_2 + \cdots + \beta_{F+N} Z_{F+N}. \tag{16}$$

Among them, Z_i represents the i^{th} dimension feature of z', so various feature combinations and corresponding model prediction scores can be trained to fit the parameter vector $\vec{w} = [\beta_0, \beta_1, \beta_2, \dots, \beta_{F+N}]$. β is the feature attribution value obtained by global mean, and the remaining β represents the individual contribution of the i^{th} dimension feature or neighboring node to the classification result. The objective of the designed explanation model in this study is to obtain the parameter vector w, and then select no more than C high-weight parameters, add their serial numbers to the set Q of important features or node serial numbers, and use them as the output of the explanation model. In addition, we provide quantitative experimental analyses, including sparsity and confidence indicators, to assess the interpreter's validity and stability.

Please note that the explainability is essential for deploying detection models in regulatory, compliance, and forensic auditing contexts. PDHGexplainer improves interpretability by quantifying the impact of node features and structural neighbors on predictions. This allows for transparent decision-making and facilitates post-hoc analysis. When a suspicious address is flagged, PDHGexplainer can pinpoint key factors such as unusual transaction frequency or concentrated activity within short time. These insights help auditors and regulators understand the model's reasoning, create verifiable audit trails, and take targeted enforcement actions.

4. Experimental evaluation

This section presents the experimental evaluation to demonstrate the effectiveness of PDHG in identifying phishing accounts within the intricate Ethereum transaction network.

4.1. Dataset

4.1.1. Dataset overview

We acquire a publicly available transaction graph dataset from the Xblock website[2], representing the Ethereum transaction network. This dataset, obtained through a second-order BFS method, encompasses a substantial Ethereum transaction network comprising 2,973,489 nodes, 13,551,303 edges, including 1165 nodes labeled as phishing accounts.

To preserve temporal detail, we retain multiple directed edges between the same ordered address pair when multiple transactions occur. At the same time, we compute aggregated statistics (e.g., total transferred amount) as separate features. Furthermore, to address the class imbalance, we construct negative seeds by uniformly sampling an equal number of unlabeled addresses. Specifically, we draw 1165 unlabeled

[2] http://xblock.pro/

Table 1
Proportion of the number of nodes of different types.

Node type	Proportion
account	95.20%
token contract	4.30%
exchange	0.31%
wallet	0.59%

Table 2
Statistics of the datasets.

Subgraph	Labeled nodes	Nodes	Edges
D_1	100	32,492	1,990,996
D_2	150	40,464	2,088,221
D_3	200	51,810	2,332,935

nodes and designate them as outliers. All random choices are controlled by a fixed random seed. The sampling and downstream experiments are repeated across 5 independent runs. With this expanded labeled set, we then extract the second-order neighbors of these focal nodes and the connecting edges between them, forming transaction subgraph capturing the local network structure around these nodes. Following subgraph extraction, each transaction subgraph consists of 9629 nodes and 386,612 edges.

4.1.2. Integration of heterogeneous graphs

Although the majority of nodes represent ordinary accounts, a subset possesses distinct identity types, such as exchanges and token contracts. While node types themselves may not directly signify involvement in phishing scams, these specialized nodes, including exchanges and token contracts, could play roles in facilitating such illicit activities. Leveraging transaction information between nodes and these specialized types may aid in identifying accounts implicated in phishing scams. Consequently, we introduce heterogeneity into the transaction graph and employ HGT models based on this premise.

As the transaction network inherently lacks node types, we source node roles from the OKLink website.[3] While OKLink offers a plethora of labels for Ethereum accounts, we specifically focus on labels potentially linked to phishing transactions, such as exchanges, token contracts, and wallets. These labels are incorporated as node attributes into the transaction graph. Notably, not all nodes in the transaction network possess corresponding node labels. Nodes lacking specific labels are assigned the default label ‘account’. Table 1 delineates the distribution of node types across the dataset.

4.1.3. Dataset splitting

Due to the original graph containing 2.97 million nodes, the computational and storage costs are prohibitively high. Therefore, we adopt the approach outlined in Sections 3.2 and 3.3, which involves constructing temporal transaction graphs and performing subgraph sampling. This method produces three transaction graphs, denoted D_1, D_2, and D_3, each containing 100, 150, and 200 phishing-tagged accounts, respectively to evaluate the model's performance across datasets of varying sizes. Further details about the dataset are presented in Table 2.

For the phishing detection task, we split the data chronologically, allocating the earliest 70 % of the data to the training set and the remaining 30 % to the test set. When extracting neighborhoods for the training split, edges that occur after the training window are excluded so that models cannot access future interactions.

4.1.4. Data accessibility and ethical considerations

All data used in this study are derived from public on-chain records and public label lists. The processing steps are implemented with openly accessible tools and can be reproduced. We have published the dataset on Mendeley Data, which is available at https://data.mendeley.com/datasets/xgm4dn7ggs/1. Furthermore, this work exclusively uses pseudonymous blockchain data without any personally identifiable information. We recommend that any deployment of PDHG's outputs be conducted under regulatory oversight, and that final labeling decisions involve human review to prevent false positives.

4.2. Experimental settings

4.2.1. Baseline methods

We compare the Detector of PDHG with four categories of Ethereum network phishing scams detection methods, which include:

1) Conventional Machine Learning and Tabular Baselines:

- **Features-only:** Features-only uses hand-crafted tabular features with simple heuristic or statistical rules, without any machine learning model.
- **Logistic Regression:** Logistic Regression (Hosmer & Lemeshow, 2000) is a classical statistical model for binary classification. It applies a logistic function to a linear combination of features, converting outputs into 0-1 probability scores for classification.
- **Random Forest:** Random Forest (Breiman, 2001) is an ensemble algorithm constructing multiple decision trees via bootstrap sampling and random feature selection. It aggregates tree outputs, with classification decided by majority vote, reducing variance and enhancing robustness.
- **Extreme Gradient Boosting (XGBoost):** XGBoost (Chen & Guestrin, 2016) is a high-performance gradient boosting implementation that builds trees sequentially to correct prior errors via regularized loss minimization, supporting efficient missing value handling and tree pruning for optimized performance.

2) Network Embedding Methods Based on Random Walks:

- **DeepWalk:** DeepWalk (Li, 2023) employs random walks and vector representation to classify nodes. It extracts node sequences akin to sentences from the network graph during the random walk phase and generates representations as vectors.
- **Node2Vec:** Node2Vec (Yuan et al., 2020b) extends DeepWalk by incorporating both DFS and BFS random walks, providing a comprehensive approach to graph embedding.
- **LINE:** LINE (Tang et al., 2015) combines first-order and second-order similarities to derive node representations, addressing the limitation of DeepWalk in modeling first-order similarity relationships.

3) Homogeneous GNN Models:

- **GCN:** GCN (Yan et al., 2022) updates node representation vectors by considering neighboring node features, thereby enhancing clustering effectiveness.
- **GAT:** GAT (Brody et al., 2022) utilizes an attention mechanism to adaptively allocate weights to neighboring nodes, significantly improving the representation capability of graph neural network models.
- **GraphSage:** GraphSAGE (Lu, 2023) is an inductive GNN model that relies on a fixed number of sampled neighbor nodes.
- **PDTGA:** PDTGA (Wang et al., 2023b) utilizes a temporal graph attention model to dynamically model the evolution of Ethereum transaction graphs, enhancing phishing scam detection's effectiveness.

4) Heterogeneous GNN Models:

- **HAN:** HAN (Zou et al., 2024) proficiently models various types of nodes and edges in hierarchical heterogeneous graphs, effectively capturing both graph structure and content.
- **RGCN:** RGCN (Attar et al., 2024) manages interactions among diverse types of relationships and entities, making it suitable for processing large-scale relational data like knowledge graphs.

[3] http://https://www.oklink.com/zh-hans

- **GEM:** GEM (Liu et al., 2018) addresses graph heterogeneity by proposing an attention mechanism to learn the importance of different types of nodes, while utilizing a summation operator to model aggregation patterns.

In addition, quantitative analysis and comparison of the performance of PDHGexplainer and the following three explainers are conducted for comparison.

- **SubgraphX.** SubgraphX (Yuan et al., 2021) focuses on providing explanations at the subgraph level in deep graph models. Leveraging the Monte Carlo Tree Search algorithm, SubgraphX explores various subgraphs through node pruning and selects the most relevant subgraph from the search path as the explanation for a prediction. This approach offers insights into the crucial subgraph structures influencing the model's decision-making process.
- **PGExplainer.** PGExplainer (Luo et al., 2020) shares a similar objective with GNNExplainer, aiming to elucidate the decision-making mechanism of GNN models to aid users in understanding the models' predictive behavior on graph data. It achieves this by integrating both local and global information to explain the predictions made by GNN models.
- **GNNExplainer.** GNNExplainer (Li et al., 2024a) provides insights into Graph Neural Network (GNN) models by elucidating the significance of each node and edge within a given input graph. By inputting a graph along with a target node or edge into the GNN model, GNNExplainer computes the contribution of each element to the model's output. This process enables users to grasp how the model leverages various components of the input graph in its decision-making process.

4.2.2. Evaluation metrics

To rigorously assess the effectiveness of various detection methods, we utilize a set of comprehensive evaluation metrics.

- **Precision**: This metric measures the accuracy of our detectors by quantifying the proportion of accurately identified phishing nodes among all nodes labeled as phishing.
- **Recall**: Recall evaluates the ability of our detectors to capture all actual phishing nodes. It calculates the percentage of true phishing nodes correctly identified out of the total number of phishing nodes present.
- **F1**: Serving as a balanced performance indicator, i.e., the F1 score. It combines precision and recall into a single metric. Calculated as the harmonic mean of these two measures, it offers a holistic evaluation of a detector's overall accuracy.
- **AUC**: AUC provides a threshold-insensitive evaluation of detector performance. It quantifies the area beneath the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds.

For the evaluation of explainer, this study adopts two indicators of sparsity and fidelity for quantitative evaluation. Here, we specifically use Fidelity +. The sparsity and fidelity are calculated as follows.

$$Sparsity = \frac{1}{N}\sum_{i=1}^{N}(1-\frac{Q}{F+M-1}), \tag{17}$$

where Q represents the number of important features and nodes identified by the explainer model for node i, $F+M-1$ represents the sum of all self-features and neighboring nodes of node i. The larger the sparsity, the higher the importance of the important features or nodes identified for explaining the results of node i, and the better the interpretation performance. Specifically, if for a certain node, most of its features and neighboring nodes are identified as important by an explainer. Then the sparsity of this node will be close to 0, which means that we cannot find the feature or neighboring node that can best "cause" it to be abnormal, so the performance of this explainer is very poor. We need to calculate the sparsity of all nodes and take the mean value as the final sparsity.

In Eq. (18), $Fidelity^{+}$ refers to the change in classification results after removing these Q features or nodes, $\hat{y}_i^{F+M-1-Q}$ represents the classification label of node i obtained by removing these Q features or nodes from the original data, and y_i represents the original classification label of node i. $I(\cdot)$ is an indicator function that equals 1 when the value in brackets satisfies the discriminant condition and 0 otherwise. The higher the fidelity, the smaller the error between the classification result of node i using only these Q features or nodes and its original classification result, and the higher the credibility of interpretation. We also need to calculate the fidelity of all nodes and take their mean value as the final fidelity.

$$Fidelity^{+} = \frac{1}{N}\sum_{i=1}^{N}(I(\hat{y}_i = y_i) - I(\hat{y}_i^{F+M-1-Q} = y_i)). \tag{18}$$

4.2.3. Implementation details

The experiments are conducted on a single Linux server featuring an NVIDIA GeForce RTX 4090 graphics card with 24GB of video memory, complemented by 10 Xeon Platinum 8352V processors and a total of 64GB of RAM.

During the training phase, the baseline and PDHG detector models employ identical hyperparameters, i.e., a hidden dimension of 400 (hid = 400), 6 attention heads (heads = 6), 2 layers (layers = 2), a dropout rate of 0.2, the Adam optimizer, gradient clipping at 0.25, a maximum of 128 epochs, and a patience of 32 epochs for early stopping. For the graph embedding methods DeepWalk and Node2Vec, the walk length is set to 20, while the window size is 4. Additionally, for Node2Vec, the parameters p and q are tuned to 0.25 and 0.4, respectively. These hyperparameters remain consistent across all datasets.

4.3. Effectiveness assessment

In this section, we empirically evaluate the performance of our phishing account detector and explainer in the context of Ethereum transaction graphs, leveraging the aforementioned datasets and baselines.

4.3.1. Effectiveness of detector

We summarize the key differences between the proposed method and baseline methods in Table 3. Clearly, PDHG effectively captures both heterogeneous network structures and temporal dynamics–capabilities that other baseline methods lack. Moreover, it achieves this without relying on predefined meta-paths, while still maintaining model interpretability, offering significant advantages in handling complex heterogeneous time-series networks. The performance evaluation of all aforementioned baseline methods on the phishing scam detection is presented in Table 4. The results demonstrate the effectiveness of the proposed approach.

1) Superiority of PDHG in Handling Imbalanced Data

PDHG exhibits outstanding performance in the node classification task. The AUC score, a robust metric for evaluating classifier performance on imbalanced datasets, particularly highlights PDHG's effectiveness. In the context of the highly imbalanced Ethereum transaction network dataset, where phishing nodes are significantly outnumbered by non-phishing nodes, PDHG consistently achieves notably higher AUC scores across datasets of varying sizes compared to baseline methods. Even when compared to the state-of-the-art PDTGA method, PDHG maintains a performance advantage of over 3 % on each dataset. PDHG's adaptability to various dataset sizes and its capability to scale as the data complexity grows make it a robust choice for large-scale networks. This scalability is particularly beneficial when applying the method to other network types beyond Ethereum, as it can effectively handle both small and large datasets, adjusting its performance according to data size and imbalance. Moreover, PDHG reduces the noise and redundancy issues

Table 3
Key differences between the proposed method and baseline methods.

Model	Heterogeneity	Temporal Encoding	Requires Meta-Path	Explainability
Conventional Machine Learning and Tabular Baselines	×	×	×	×
Network Embedding Methods Based on Random Walks	×	×	×	×
Homogeneous GNN Models	×	×	×	×
Heterogeneous GNN Models	✓	×	✓	×
PDHG	✓	✓	×	✓

Table 4
Performance comparison of the proposed method and baseline methods across three datasets.

Detector	D_1				D_2				D_3			
	AUC	Recall	Precision	F1	AUC	Recall	Precision	F1	AUC	Recall	Precision	F1
Features-only	0.7056	0.3818	0.6893	0.4914	0.6988	0.3154	0.6130	0.4165	0.6988	0.3733	0.6338	0.4698
Logistic Regression	0.7098	0.4213	0.6912	0.5236	0.6991	0.3995	0.6797	0.5035	0.7197	0.4492	0.6999	0.5475
Random Forest	0.8147	0.6004	0.7098	0.6501	0.7948	0.5849	0.6952	0.6355	0.8043	0.6397	0.7349	0.6843
XGBoost	0.8214	0.6211	0.7210	0.6674	0.8042	0.6557	0.7218	0.6871	0.8109	0.6596	0.7390	0.6975
DeepWalk	0.7187	0.5156	0.6999	0.5938	0.6713	0.5562	0.4352	0.4883	0.7229	0.3816	0.5786	0.4599
Node2Vec	0.6353	0.4072	0.5904	0.4820	0.6640	0.5341	0.4229	0.4720	0.7840	0.6482	0.4849	0.5548
LINE	0.7774	0.7011	0.6302	0.6638	0.7455	0.6165	0.6240	0.6202	0.8054	0.5984	0.6046	0.6015
GAT	0.6880	0.7193	0.7850	0.7507	0.7042	0.7518	0.7978	0.7741	0.7193	0.7477	0.8144	0.7796
GraphSAGE	0.6586	0.8335	0.8012	0.8170	0.6354	0.7503	0.7828	0.7662	0.6409	0.7786	0.8061	0.7921
GCN	0.6985	0.5581	0.6140	0.5847	0.7647	0.6799	0.7361	0.7069	0.7379	0.6388	0.6199	0.6292
PDTGA	0.8973	0.8258	0.7971	0.8112	0.9096	0.8474	0.7826	0.8137	0.9194	0.8610	0.7775	0.8171
HAN	0.7217	0.4685	0.8624	0.6072	0.7479	0.5833	0.8846	0.7030	0.7567	0.6220	0.8833	0.7300
RGCN	0.8006	0.7305	0.8779	0.7974	0.8196	0.7442	0.9155	0.8210	0.8489	0.7562	0.8882	0.8169
GEM	0.8064	0.4423	0.8722	0.5869	0.8335	0.5519	0.8912	0.6817	0.8381	0.5566	0.8861	0.6837
PDHG	0.9293	0.8551	0.8255	0.8401	0.9387	0.8745	0.8076	0.8397	0.9604	0.8994	0.8122	0.8536

by combining subgraph sampling with heterogeneous transformer learning. First, it operates on sampled k-hop subgraphs in a GraphSAGE-style manner rather than learning from full-graph, minimizing exposure to irrelevant or duplicate contexts. Second, it adopts HGT with cross-type parameter sharing, ensuring the model prioritizes type-aware, informative interactions over noisy local co-occurrences.

2) Increasing Advantage of PDHG with Dataset Scaling

PDHG is designed with scalability in mind. While any deep GNN can be computationally intensive, PDHG's use of GraphSAGE-style sampling and the inherent efficiency of HGT mean it scales better than naive full-graph GNNs. With the increase in dataset size, the advantage of PDHG over other methods also escalates. In comparison with the best-performing PDTGA method overall, the AUC score advantage of our method expands from 3.2 to 4.1 percentage points, and the Recall score advantage increases from 2.93 to 3.84 percentage points. It is evident that while PDHG exhibits relatively minor performance advantage on small-scale datasets, its superiority gradually becomes apparent as the dataset size grows. This indicates PDHG's capability to better handle the complexity and heterogeneity of large-scale graph data in the Ethereum transaction network, resulting in improved performance on large-scale datasets. PDHG's increasing performance advantage with dataset scaling illustrates its ability to adapt to larger, more complex datasets. This characteristic makes it highly scalable for different application scenarios, particularly in dynamic or expanding transaction networks where new data is continuously integrated. In summary, PDHG is practical for subgraphs up to 50k nodes and 2-3M edges on a single RTX4090.

3) Benefit of Graph Modeling over Conventional Machine Learning and Tabular Baselines

The conventional machine learning (ML) methods (i.e., Logistic Regression, Random Forest, and XGBoost) are better than the tabular baseline (i.e., Features only), with AUC values ranging from 0.69 to 0.83. These results confirm that transaction-level statistical features contain non-trivial signals for phishing detection. Specifically, the gap between these baselines and graph-based models remains significant, particularly in terms of F1 and recall. This indicates that tabular baselines and conventional ML methods struggle to fully capture higher-order structural dependencies that characterize fraudulent behaviors. In contrast, graph-based models exploit the relational structure of the Ethereum transaction network, enabling them to encode both direct and indirect interactions among accounts. By modeling heterogeneous node and edge types, PDHG further enhances this capability, producing strong improvements in every metric. The superiority of PDHG underscores the importance of explicitly leveraging graph topology for Ethereum phishing detection.

4) Weak Performance of Random Walk Methods

The random walk method exhibits the weakest overall performance among the analyzed methods. DeepWalk captures node similarity by representing nodes as word vectors using a bag-of-words model. However, its performance improvement is limited as the dataset size increases, likely due to the increase in noise and redundancy in large datasets. Node2Vec, similar to DeepWalk, is also based on random walks but introduces more flexibility to capture more complex relationships between nodes. Nevertheless, it shows no significant improvement on large datasets. LINE emerges as the top performer, which is likely attributed to its emphasis on capturing direct relationships between nodes. This approach enables LINE to effectively capture global relationships, particularly as the dataset expands, resulting in a notable performance boost. While random walk methods like DeepWalk and Node2Vec offer limited scalability, LINE's approach is more adaptable to larger datasets.

5) Comparative Analysis of Homogeneous Graph Models

Approaches leveraging homogeneous graph models typically exhibit superior performance compared to random walk models. These methods excel at effectively extracting node features and integrating them into the spatial structure, a crucial aspect given the intricate nature of transaction networks where phishing accounts often lurk. Among the tested models, GAT surpasses GraphSAGE and GCN due to its capacity to explore label distribution during sampling process and adeptly gather features from neighboring nodes. While PDTGA demonstrates commendable performance, ranking second only to PDHG, its enhancement of GAT, achieved by substituting self-attention mechanism's positional encoding module with relative time encoding, enables it to handle dynamic graphs. By aggregating the features of neighboring nodes and related edges during the embedding process, PDTGA significantly enhances its node representation capability, highlighting its edge over other homogeneous graph models. Nonetheless, its adaptation to large-scale, intricate heterogeneous graph data remains limited. In contrast, PDHG, tailored to address the nuances of heterogeneous graphs, holds a competitive edge in handling such data. Homogeneous models like GAT are effective for simpler graph structures, but PDHG stands out for its ability to handle heterogeneous graphs, which are more common in real-world scenarios. PDHG's scalability in dealing with increasingly complex heterogeneous networks, where multiple node and edge types exist, makes it more adaptable across various industries and larger datasets. The method can handle dynamic and evolving networks more effectively than homogeneous models.

6) Advantages of Heterogeneous Graph Models over Homogeneous Graph Models

Overall, the heterogeneous graph models outperform the homogeneous graph models, largely due to their superior capability in navigating the complexity of heterogeneous graphs.

These models exhibit proficiency in managing various node and edge types, flexibly utilizing features and relationships, and employing sophisticated information aggregation mechanisms. These attributes empower heterogeneous graph models to excel in node classification tasks. PDHG offers significant scalability and adaptability advantages, especially in complex environments with diverse data sources. As more heterogeneous data is integrated, PDHG's flexibility allows it to continually improve its classification capabilities, making it suitable for applications requiring high scalability and adaptability.

4.3.2. *Effectiveness of explainer*

From Table 5, we can see that:

1) The fidelity of the SubgraphX model is second only to the model proposed in this study, but its sparsity is very low, which means that the model uses too many features to explain and is not representative.

To investigate the reason, we analyze the explanation principle of the SubgraphX model and find that this model only considers the impact of important neighboring nodes on the classification results, without considering important node features. The experiments designed in this study incorporate both node features and neighboring nodes into the explanation category. In addition, the Ethereum transaction network has its own characteristics, where the average degree of transaction nodes is low, meaning that there are fewer neighboring nodes to choose from. Since each node has an impact on the prediction results, SubgraphX will mark all neighboring nodes of the node to be explained as important nodes for the classification results. Consequently, the explanation results lack discriminative power and are not informative. While SubgraphX offers good fidelity in simpler graphs, but its performance decreases when applied to more complex heterogeneous networks. Its lack of scalability makes it less adaptable to datasets with intricate structures and diverse node features, which limits its usefulness for large-scale applications.

2) The GNNExplainer model has a high sparsity but a low fidelity. The high sparsity indicates that the model can select the most representative and important features from a large set of candidate features, indicating that the model performs well in selecting important nodes. The reason for the low fidelity is that we analyze the explanation principle of this model and come to the following conclusion. Firstly, the GNNExplainer model is a perturbation-based explanation method, and its mask generation strategy is special. Both node feature masking and edge masking use soft masking. According to the calculation method of fidelity in this article, using soft masks will result in a lower fidelity index. Secondly, GNNExplainer only masks its own node features, while the method in this study masks all nodes within the neighborhood, so it does not block the impact of neighboring node features on its own node features. Therefore, GNNExplainer has poor explainability at the feature level. The GNNExplainer model's approach is more scalable compared to SubgraphX, but its effectiveness is still limited when applied to complex, dynamic networks. Its lack of adaptability to neighborhood features also impacts its performance in networks with intricate relationships between nodes.

Table 5
Performance comparison of different explainers.

ID	Explainer	Sparsity	Fidelity$^+$
1	SubgraphX	0.0200	0.3189
2	PGExplainer	0.6570	0.1234
3	GNNExplainer	0.8963	0.1069
4	PDHGexplainer (Our approach)	0.5978	0.4531

3) The PGExplainer model introduces a reparameterization technique compared to GNNExplainer, improving explanation efficiency and accuracy. PGExplainer has a sparsity of 0.59, indicating that the model maintains a certain level of simplicity while retaining enough information to ensure comprehensiveness when selecting key nodes and edges. In contrast, GNNExplainer has a higher sparsity of 0.89, meaning it focuses on even fewer nodes and edges, providing a more compressed explanation. While the higher sparsity in GNNExplainer may lead to more concise explanations, it can also risk oversimplification, potentially missing important features that are essential for the prediction. PGExplainer's moderate sparsity allows it to effectively highlight important features while avoiding over-simplification or over-complication, providing higher fidelity and maintaining computational efficiency. Therefore, PGExplainer strikes a better balance between computational cost and explanation quality, especially when interpreting complex graph structures. Meanwhile, instead of producing opaque risk scores alone, PDHGexplainer generates compact, actionable artifacts that can be directly integrated into standard compliance workflows.

4) PDHGexplainer works by perturbing node features and neighborhood structure and fitting a simple linear surrogate to approximate the GNN's output. It uses only masked inputs and reconstructed adjacency or feature matrices. Therefore, the explainer is model-agnostic. In principle, PDHGexplainer could be applied to any graph neural network model, as long as one can mask node features and neighbors and observe changes in the model's predictions. Notably, practical application to other models may require modest adaptation of the mask and reconstruction pipeline to match the target model's input format. For example, ensuring that the feature mask has the same dimensionality, ordering, and preprocessing semantics as the target model's input vector.

4.4. *Ablation study*

The core components of HGT are Heterogeneous Weight Parameterization (Heter) and Relative Time Encoding (RTE). To further analyze their effects, in this section, we conduct ablation study by removing them from HGT. Specifically, we divide the experiments into three groups for comparison: 1) Removing the heterogeneous weight parameterization from the HGT model; 2) Removing the relative time encoding from the HGT model; 3) Simultaneously removing both the heterogeneous weight parameterization and the relative time encoding from the HGT model.

By analyzing Figs. 3 and 4, the following conclusions can be drawn.

1) Detectors equipped with a complete HGT module demonstrate the highest classification performance, while those lacking heterogeneous weight parameterization and relative time encoding exhibit the poorest performance. This underscores the positive impact of both heterogeneous weight parameterization and relative time encoding on enhancing the efficiency of phishing scam account detection.

2) Detectors without relative time encoding only slightly under perform compared to detectors with a complete HGT module. This indicates that heterogeneous weight parameterization plays a more signif-

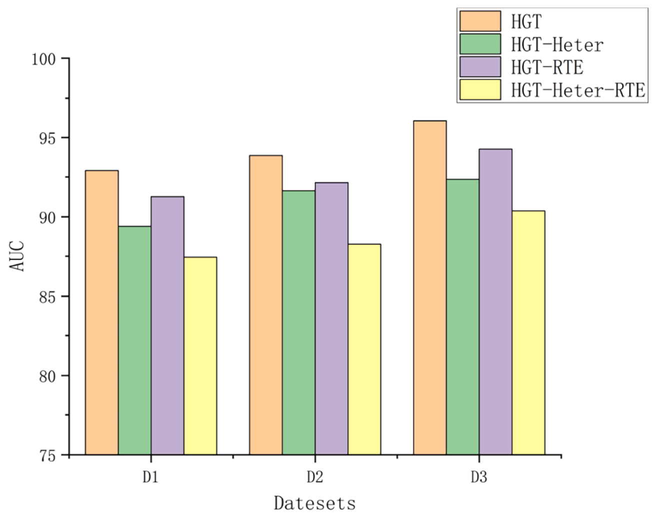

Fig. 3. AUC of ablation experiments.

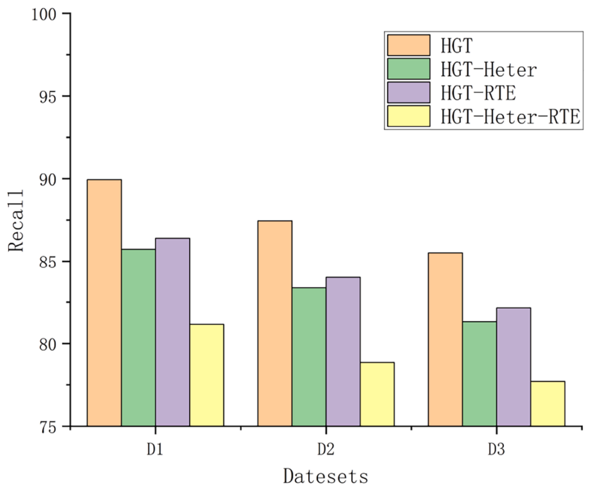

Fig. 4. Recall of ablation experiments.

icant role in detecting network phishing scam accounts than relative time encoding. Removing this heterogeneous parameterization causes the greatest drop in recall, indicating PDHG loses its ability to leverage type-distinctive transaction patterns. Heterogeneous GNN models, supporting various types of nodes and edges, are pivotal for improving the detection capability of network phishing scam accounts. The integration of heterogeneous weight parameterization and time encoding modules facilitates the extraction of temporal patterns from transaction interactions between different node types, resulting in more expressive node representations.

3) Eliminating heterogeneous weight parameterization or relative time encoding has a greater impact on the Recall score of the detector than on the AUC score. Although the AUC score serves as a comprehensive indicator of the model performance, the recall score specifically highlights the proportion of accurately detected phishing accounts among the true positive cases. The experimental findings highlight the importance of heterogeneous weight parameterization and time encoding in enhancing the classifier's ability to correctly identify positives. The balance between precision and recall is crucial in the modeling process. When F1 scores are comparable, models with higher recall are preferred. Transaction time features are essential for augmenting the model's phishing scam account detection capability. Without encoding timestamps with sinusoidal position vectors, PDHG can't effectively capture temporal transaction patterns, disabling the learning of more informative node representations.

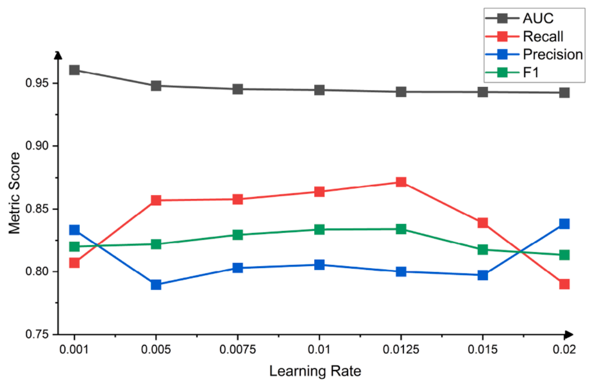

Fig. 5. Results of learning rate sensitive experiments.

4.5. Sensitivity analysis

Given the significant role of the HGT layer in our model, we delve into a comprehensive analysis of its parameters' influence on detection performance. Our focus is primarily on crucial factors such as learning rate, attention heads, and the convolutional layers. All sensitivity analysis experiments are conducted using dataset D_3.

4.5.1. Impact of learning rate

During the training process of a machine learning model, the learning rate serves as a crucial hyperparameter. To gain a deeper understanding of how the learning rate affects model performance, we conduct detailed experiments with a range of initial learning rates, including 0.001, 0.005, 0.0075, 0.01, 0.0125, 0.015, 0.02. As shown in the experimental results depicted in Fig. 5, we observe a notable impact of the learning rate on the model's AUC score.

Specifically, as the learning rate increases, the AUC score appears a clear downward trend. However, this decreasing trend becomes less pronounced when the learning rate exceeds 0.005. This suggests that the model is more sensitive to changes in the learning rate within a smaller range.

In terms of recall score, we observe an initial increase followed by a decrease, with the recall peaking notably at a learning rate of 0.0125. Conversely, the precision score displays an opposite trend to the recall score. Further analysis reveals that the F1 score also reaches its peak when the learning rate is set to 0.0125.

Based on the variations observed in these metrics, we conclude that setting the initial learning rate to 0.0125 yields optimal classification performance. This finding provides valuable guidance for selecting an appropriate learning rate when training similar models in the future.

4.5.2. Impact of attention size and layer configuration

In our exploration of the Heterogeneous Graph Transformer (HGT) architecture, we delve into the nuances of its attention mechanisms and layer stacking. We conduct a thorough sensitivity analysis to investigate how varying the number of attention heads and layers within the HGT embedding layer affects the overall classification performance of the model.

Specifically, we experiment with different configurations, ranging from 1 to 2 layers and from 1 to 6 attention heads. Our goal is to determine the optimal setup that maximizes the model's ability to accurately classify the accounts. The results of experiments are presented in Figs. 6 and 7.

A notable discovery emerges during our experimentation: Consistently deploying two stacked layers of the HGT resulted in superior AUC and Recall, irrespective of the size of attention heads utilized, compared

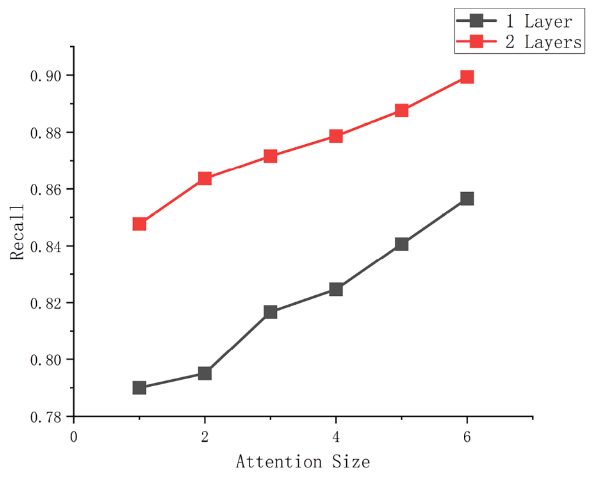

Fig. 6. Comparative analysis of AUC scores across different attention head sizes and layer numbers.

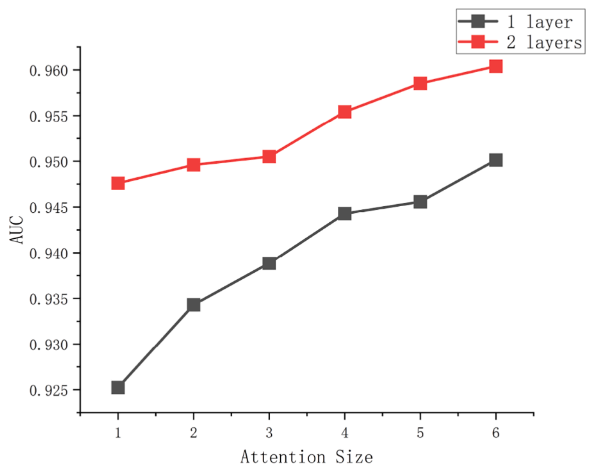

Fig. 7. Comparison of recall scores across various attention head sizes and layer numbers.

to utilizing a single layer. This finding underscores the importance of layer stacking in boosting the model's classification performance. By adding another layer, the model can capture more complex patterns and relationships within the data, resulting in improved predictions.

Moreover, we notice a positive correlation between the number of attention heads and the model's performance. Both AUC and Recall scores increase as we gradually increase the number of attention heads, for both single-layer and two-layer configurations. This trend suggests that the attention mechanism plays a crucial role in enhancing the model's ability to focus on relevant information and ignore irrelevant details. By attending to multiple aspects of the input data simultaneously, the model can make more informed and accurate predictions.

4.6. Scalability Analysis

To further validate PDHG's scalability beyond our primary datasets, we introduce an additional publicly available Ethereum dataset, PTXPHISH, which contains 5000 payload-based phishing examples.[4] Based on the previous experiments, we select the best-performing representative from each baseline category for comparison. We retrain and evaluate these methods on PTXPHISH following the same training and evaluation procedures described above. Table 6 summarizes PDHG and the top baselines' performances on PTXPHISH.

[4] https://github.com/blocksecteam/PTXPhish

Table 6
Performance comparison of top-performing methods on PTXPHISH.

Category	Detector	AUC	Recall	Precision	F1
Conventional Machine Learning	XGBoost	0.8354	0.6420	0.7051	0.6721
Network Embedding Methods	LINE	0.7921	0.5883	0.6527	0.6188
Homogeneous GNN Models	PDTGA	0.9127	0.8165	0.7832	0.7995
Heterogeneous Graph Networks	RGCN	0.8819	0.7564	0.8117	0.7831
Proposed Method	PDHG	0.9453	0.8821	0.8214	0.8507

First, PDHG achieves the highest recall, while maintaining strong precision and F1. Second, PDTGA and RGCN perform comparably but consistently lag behind, as PDHG's temporal encoding appears to provide additional gains. Finally, XGBoost and LINE obtain lower AUC and recall, suggesting that static tabular features or static embeddings alone do not fully capture the rich, time-varying, and type-aware signals present in phishing attacks. The results validate PDHG's scalability on independent Ethereum datasets, indicating that its superior performance is not an artifact of the specific datasets.

Nevertheless, several limitations should be noted. First, PDHG relies on the quality and representativeness of labeled phishing data. If new phishing strategies emerge that are very different from the training set, PDHG may fail to recognize them. Second, because PDHG depends on transaction features, it may under perform in cases where phishing accounts cleverly mask their transaction patterns. Finally, PDHG's computational complexity is higher than very lightweight models. However, the slight increase in computational overhead is deserving for improving the detection accuracy. In operational settings where computational overhead is critical possibly at some accuracy cost, like real-time detection on streaming data, a simpler method might be preferred.

5. Related work

5.1. Detection of Ethereum phishing scams

In the realm of detecting Ethereum phishing scam accounts, existing methods typically fall into two main categories.

Early detection methods primarily rely on shallow models, such as traditional machine learning approaches with dedicated feature engineering, emphasizing statistical features. For example, Ibrahim et al. (2021) used the correlation coefficient to select the most effective features from 42 statistical features. They constructed a new dataset using only six features and employed three machine learning algorithms – decision tree (j48), random forest, and K-nearest neighbors (KNN) for classification. Wang et al. (2023a) proposed TTPS, a Long Short-Term Memory (LSTM) detection method considering time series transaction information of smart contracts. TTPS considered both temporal account features and code features of smart contracts. Adaptive synthetic sampling (ADASYN) was employed to effectively expand the feature data of small samples. It is worth mentioning that Liu et al. (2024) first investigated the characteristics of Ethereum phishing gangs from the perspectives of individuals, pairs, and high-order patterns. By analyzing the features, they found that although the Ethereum transaction graph is sparse, phishing accounts in the same gang share specific transaction patterns and introduced a novel detection model named PGDetector.

The latter employs various network embedding techniques, including DeepWalk (Li, 2023), Node2Vec (Yuan et al., 2020b), and graph convolutional networks (GCN) (Yan et al., 2022), to extract deep features. To address the issue of large-scale network sampling, Yuan et al. (2020c) improved Graph2Vec (Narayanan et al., 2017), which extracts rooted subgraphs to represent all graphs in the form of a set of subgraphs. Wang et al. (2023c) adopted random walk method and divided representation learning into node representation learning, edge representation learning, and attribute representation learning, and then combined these three types of approaches. Chen et al. (2021b) first applied Graph Convolutional Networks to detect phishing scams in

Ethereum. Their GCN model effectively integrated node features and spatial structure features of the network. Huang et al. (2024) proposed a novel framework named phishing detection on Ethereum via augmentation ego-graph based on graph neural network (PEAE-GNN). Specifically, a new graph-level representation method was proposed, which obtained the representation by sorting the updated node features in descending order and then taking the average of the top n node features. Hou et al. (2025) designed node state representations to facilitate random walk sampling, and then manually extracted 8-D features from the resulting dataset as basic features. Temporal features were co-learned via a combination of a long short-term memory network and contrastive learning.

Recently, there has been a trend towards addressing real-world anomaly detection problems using heterogeneous graphs (e.g., spam review detection, suspicious user identification) or combining homogeneous and heterogeneous graphs. Those approaches enable the aggregation of information propagation through various types of nodes and edges. Kim et al. (2023) and Huang et al. (2023a) employed heterogeneous graph-based supervised learning to detect phishing accounts, achieving promising results. Zhang et al. (2024) built the evolutionary patterns of Ethereum account transactions into a diffusion network graph in continuous time. This method can continuously capture new transaction features based on existing transactions, thereby facilitating the identification of phishing accounts. Yang et al. (2024) constructed a temporal multi-edge transaction-directed graph that incorporates diverse node types (e.g., externally owned accounts vs. contract accounts) and edge attributes (e.g., transaction amounts, timestamps, gas consumption).

5.2. Explainability in GNN

The explainability and transparency of predictions made by Graph Neural Networks (GNNs) have attracted significant attention in recent years. The research surrounding this area can be broadly categorized into two main dimensions: the input level and the model level.

At the input level, various methodologies have been developed to elucidate how specific input features or graph components influence the final prediction. One of the most notable frameworks in this domain is GNNExplainer (Li et al., 2024a), which used a subgraph generation approach to highlight the most critical nodes and edges contributing to the model's decision. Additionally, GraphLIME (Huang et al., 2023b) extended the Local Interpretable Model-agnostic Explanations (LIME) framework to GNNs by approximating the complex graph model with interpretable, locally linear surrogate models, thus offering a way to explain predictions in a way that is understandable for end-users. L2xGnn (Serra & Niepert, 2024) learned a mechanism for each input graph to filter subgraphs (Motifs) with explanatory properties. These selected subgraphs are dedicated to subsequent message-passing operations of the GNN. Another input-level methodology is proposed by Wang et al. (2025c), which introduced a graph segmentation learning module to divide the input graph into explanatory and redundant subgraphs. These methods emphasized revealing which specific graph structures or features play pivotal roles in influencing a given prediction, thus aiding domain experts in understanding and trusting model outputs.

At the model level, explainability strategies aim to enhance the transparency of the GNN architecture itself. A well-known approach in this regard is XGNN (Yuan et al., 2020a), which incorporated techniques such as attention mechanisms to interpret the importance of different graph components during the model's operation. GAN-GNNExplainer (Li et al., 2024b) employed a generator to produce explanations and a discriminator to oversee the generation process, enhancing the reliability of the outputs. KnowGNN (Ma et al., 2025) innovatively integrated prior knowledge and graph structural relationships into the global explanation process of the model, addressing the defects of existing model-level methods and enhancing the structural sensitivity of explanations.

Unlike most existing Ethereum phishing scam detection approaches, PDHG models the transaction network as a heterogeneous graph that incorporates various types of accounts, eliminating the need for manual meta-path design. By introducing time encoding, it also effectively utilizes transaction time information. Moreover, PDHG leverages PDHGexplainer–including feature masking and neighborhood node masking–to explain detection results and enhance practical utility. These allow PDHG to outperform representative detection approaches as well as the state-of-the-art explainability methods in GNNs.

6. Conclusion

In this study, we introduce PDHG, a novel approach for the detection of Ethereum phishing scam accounts, with a emphasis on providing transparent and explainable model predictions. Our approach revolves around the construction of a heterogeneous graph and the utilization of a self-attention heterogeneous graph neural network for the classification of fraudulent accounts. Moreover, we present a novel framework for graph interpretation, integrating a learnable graph interpreter that explains the significance of node features and their relationships within the graph. Through extensive experiments on real Ethereum transaction datasets, the results showcase the superior performance of PDHG in both phishing detection effectiveness and the model explanations.

In future works, we plan to integrate the detection model with smart contracts. We will also investigate how to use HGT for more types of Ethereum fraud detection applications. Furthermore, we aim to explore the application of PDHG to other blockchain platforms such as Bitcoin, Binance, or Tron. Such extensions would validate PDHG's generalizability to heterogeneous temporal transaction networks beyond Ethereum.

CRediT authorship contribution statement

Lei Wang: Conceptualization, Methodology, Formal analysis, Writing – original draft, Writing – review & editing; **Yihan Mi:** Software, Writing – original draft; **Yanan Zhang:** Software, Writing – original draft; **Jialin Zhang:** Software, Writing – original draft.

Data availability

The authors have shared the link to the dataset at the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by the NSFC (nos 62372239, 62372240) and the Humanity and Social Science Fund of Ministry of Education of China (no 23YJAZH141).

References

Abdelhamid, N., Ayesh, A., & Thabtah, F. (2014). Phishing detection based associative classification data mining. *Expert Systems with Applications*, *41*(13), 5948–5959.

Attar, Z. E., Hoarau, K., Hadjadj-Aoul, Y., & Texier, G. (2024). RGCN for beyond pairwise training: Generalizing monitors selection in network tomography. In *International symposium on networks, computers and communications, ISNCC 2024, Washington, DC, USA, October 22–25, 2024* (pp. 1–6).

Ayllón-Gavilán, R., Guijo-Rubio, D., Gutiérrez, P. A., Bagnall, A., & Hervás-Martínez, C. (2025). Convolutional- and deep learning-based techniques for time series ordinal classification. *IEEE Transactions on Cybernetics*, *55*(2), 537–549.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Brody, S., Alon, U., & Yahav, E. (2022). How attentive are graph attention networks? In *The tenth international conference on learning representations, ICLR* (pp. 1–19).

Chen, L., Peng, J., Liu, Y., Li, J., Xie, F., & Zheng, Z. (2021a). Phishing scams detection in Ethereum transaction network. *ACM Transactions on Internet Technology*, *21*(1), 10:1–10:16.

Chen, L., Peng, J., Liu, Y., Li, J., Xie, F., & Zheng, Z. (2021b). Phishing scams detection in Ethereum transaction network. *ACM Transactions on Internet Technology*, *21*(1), 10:1–10:16.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco, CA, USA, August 13–17, 2016* (pp. 785–794). ACM.

Hosmer, D. W., & Lemeshow, S. (2000). Applied logistic regression, second edition. Wiley.

Hou, W., Cui, B., Chen, Y., Li, R., & Song, W. (2025). TSFF: A triple-stream feature fusion method for Ethereum phishing scam detection. *IEEE Internet of Things Journal*, *12*, 2623–2632.

Hou, W., Cui, B., & Li, R. (2022). Detecting phishing scams on Ethereum using graph convolutional networks with conditional random field. In *24th IEEE int conf on high performance computing & communications; 8th int conf on data science & systems; 20th int conf on smart city; 8th int conf on dependability in sensor, cloud & big data systems & application, hpcc/dss/smartcity/dependsys 2022, Hainan, China, December 18–20, 2022* (pp. 1495–1500).

Huang, B., Liu, J., Wu, J., Li, Q., & Lin, D. (2023a). Ethereum phishing fraud detection based on heterogeneous transaction subnets. In *IEEE international symposium on circuits and systems, ISCAS 2023, Monterey, CA, USA, May 21–25, 2023* (pp. 1–5). IEEE.

Huang, H., Zhang, X., Wang, J., Gao, C., Li, X., Zhu, R., & Ma, Q. (2024). PEAE-GNN: Phishing detection on Ethereum via augmentation ego-graph based on graph neural network. *IEEE Transactions on Computational Social Systems*, *11*(3), 4326–4339.

Huang, Q., Yamada, M., Tian, Y., Singh, D., & Chang, Y. (2023b). GraphLIME:local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *35*(7), 6968–6972.

Ibrahim, R. F., Elian, A. M., & Ababneh, M. (2021). Illicit account detection in the Ethereum blockchain using machine learning. In *2021 international conference on information technology (ICIT)* (pp. 488–493). IEEE.

Jin, C., Zhou, J., Xie, C., Yu, S., Xuan, Q., & Yang, X. (2025). Enhancing Ethereum fraud detection via generative and contrastive self-supervision. *IEEE Transactions on Information Forensics & Security*, *20*, 839–853.

Kim, J., Lee, S., Kim, Y., Ahn, S., & Cho, S. (2023). Graph learning-based blockchain phishing account detection with a heterogeneous transaction graph. *Sensors*, *23*(1), 463.

Li, H., Han, Z., & Sun, Y. (2024a). Cgmega: Explainable graph neural network framework with attention mechanisms for cancer gene module dissection. *Nature Communications*, *15*, 5997.

Li, J. (2023). Multichannel walk embedding based Ethereum phishing scam detection method. In *Proceedings of the 28th IEEE international conference on parallel and distributed systems (ICPADS)* (pp. 289–295).

Li, Y., Zhou, J., Zheng, B., Shafiabady, N., & Chen, F. (2024b). Reliable and faithful generative explainers for graph neural networks. *Machine Learning and Knowledge Extraction*, *6*(4), 2913–2929.

Lin, D., Wu, J., Huang, T., Lin, K., & Zheng, Z. (2024). Who is who on Ethereum? Account labeling using heterophilic graph convolutional network. *IEEE Transactions on Systems, Man, and Cybernetics Systems*, *54*(3), 1541–1553.

Liu, J., Chen, J., Wu, J., Wu, Z., Fang, J., & Zheng, Z. (2024). Fishing for fraudsters: Uncovering Ethereum phishing gangs with blockchain data. *IEEE Transactions on Information Forensics and Security*, *19*, 3038–3050.

Liu, Z., Chen, C., Yang, X., Zhou, J., Li, X., & Song, L. (2018). Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM international conference on information and knowledge management* (pp. 2077–2085).

Lu, Y. (2023). Enhanced network security protection through data analysis and machine learning: An application of graphSAGE for anomaly detection and operational intelligence. *Journal of Computing and Information Technology*, *31*, 233–250.

Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., & Zhang, X. (2020). Parameterized explainer for graph neural network. *Advances in Neural Information Processing Systems*, *33*, 19620–19631.

Ma, Y., Liu, X., Guo, C., Jin, B., & Liu, H. (2025). KnowGNN: A knowledge-aware and structure-sensitive model-level explainer for graph neural networks. *Applied Intelligence*, *55*(2), 126.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., & Jaiswal, S. (2017). graph2vec: Learning distributed representations of graphs. arXiv:1707.05005.

Serra, G., & Niepert, M. (2024). L2XGNN: Learning to explain graph neural networks. *Machine Learning*, *113*(9), 6787–6809.

Sui, H., Zhang, J., Chen, B., Wu, D., Sun, X., & Palaiahnakote, S. (2025). Epad: Ethereum phishing scam detection via graph contrastive learning. *Expert Systems with Applications*, *288*, 125789.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web* (pp. 1067–1077).

Tang, M., Ye, M., Chen, W., & Zhou, D. (2024). BiLSTM4DPS: An attention-based biLSTM approach for detecting phishing scams in Ethereum. *Expert Systems with Applications*, *256*, 124941.

Tharani, J. S., Hóu, Z., Charles, E. Y. A., Rathore, P., Palaniswami, M., & Muthukkumarasamy, V. (2024). Unified feature engineering for detection of malicious entities in blockchain networks. *IEEE Transactions on Information Forensics and Security*, *19*, 8924–8938.

Wang, J., Chen, P., Xu, X., Wu, J., Shen, M., Xuan, Q., & Yang, X. (2025a). Tsgn: Transaction subgraph networks assisting phishing detection in Ethereum. *IEEE Transactions on Dependable and Secure Computing*, (pp. 1–16).

Wang, L., Cheng, H., Sun, Z., Tian, A., & Yang, Z. (2025b). PSPL: A ponzi scheme smart contracts detection approach via compressed sensing oversampling-based peephole LSTM. *Future Generations Computer Systems : FGCS*, *166*, 107655.

Wang, L., Cheng, H., Zheng, Z., Yang, A., & Xu, M. (2023a). Temporal transaction information-aware ponzi scheme detection for Ethereum smart contracts. *Engineering Applications of Artificial Intelligence*, *126*, 107022.

Wang, L., Cheng, H., Zheng, Z., Yang, A., & Zhu, X. (2021). Ponzi scheme detection via oversampling-based long short-term memory for smart contracts. *Knowledge-Based Systems*, *228*, 107312.

Wang, L., Xu, M., & Cheng, H. (2023b). Phishing scams detection via temporal graph attention network in Ethereum. *Information Processing & Management*, *60*(4), 103412.

Wang, Y., Liu, Z., Xu, J., & Yan, W. (2023c). Heterogeneous network representation learning approach for Ethereum identity identification. *IEEE Transactions on Computational Social Systems*, *10*(3), 890–899.

Wang, Z., Guo, J., Liang, J., Liang, J., Cheng, S., & Zhang, J. (2025c). Graph segmentation and contrastive enhanced explainer for graph neural networks. In *AAAI-25, sponsored by the association for the advancement of artificial intelligence, February 25–March 4, 2025, Philadelphia, PA, USA* (pp. 21393–21401).

Wen, T., Xiao, Y., Wang, A., & Wang, H. (2023). A novel hybrid feature fusion model for detecting phishing scam on Ethereum using deep neural network. *Expert Systems with Applications*, *211*, 118463.

Wu, J., Yuan, Q., Lin, D., You, W., Chen, W., Chen, C., & Zheng, Z. (2022). Who are the phishers? phishing scam detection on Ethereum via network embedding. *IEEE Transactions on Systems, Man, and Cybernetics Systems*, *52*(2), 1156–1166.

Yan, Y., Hashemi, M., Swersky, K., Yang, Y., & Koutra, D. (2022). Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *IEEE international conference on data mining, ICDM 2022, Orlando, FL, USA, November 28–December 1, 2022* (pp. 1287–1292).

Yang, J., Yu, W., Wu, J., Lin, D., Wu, Z., & Zheng, Z. (2024). 2dynethnet: A two-dimensional streaming framework for Ethereum phishing scam detection. *IEEE Transactions on Information Forensics and Security*, *19*, 9924–9937.

Yuan, H., Tang, J., Hu, X., & Ji, S. (2020a). Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 430–438).

Yuan, H., Yu, H., Wang, J., Li, K., & Ji, S. (2021). On explainability of graph neural networks via subgraph explorations. In *International conference on machine learning* (pp. 12241–12252).

Yuan, Q., Huang, B., Zhang, J., Wu, J., Zhang, H., & Zhang, X. (2020b). Detecting phishing scams on Ethereum based on transaction records. In *IEEE international symposium on circuits and systems, ISCAS 2020, Sevilla, Spain, October 10–21, 2020* (pp. 1–5).

Yuan, Z., Yuan, Q., & Wu, J. (2020c). Phishing detection on Ethereum via learning representation of transaction subgraphs. In *Blockchain and trustworthy systems: Second international conference, blocksys 2020, Dali, China, August 6–7, 2020, revised selected papers 2* (pp. 178–191).

Zhang, J., Sui, H., Sun, X., Ge, C., Zhou, L., & Susilo, W. (2024). Grabphisher: Phishing scams detection in Ethereum via temporally evolving GNNs. *IEEE Transactions on Services Computing*, *17*(6), 3727–3741.

Zhong, Q., Liu, Y., Ao, X., Hu, B., Feng, J., Tang, J., & He, Q. (2020). Financial defaulter detection on online credit payment via multi-view attributed heterogeneous information network. In *Proceedings of the web conference 2020* (pp. 785–795).

Zou, Z., Wang, B., Wan, H., Jin, H., Li, X., & Cao, Y. (2024). Hantracer: Leveraging heterogeneous graph attention network for large-scale requirements-code traceability link recovery. In *31st Asia-Pacific software engineering conference, APSEC 2024, Chongqing, China, December 3–6, 2024* (pp. 211–220).