# Effective Illicit Account Detection on Large Cryptocurrency MultiGraphs

Zhihao Ding
tommy-zh.ding@connect.polyu.hk
Department of Computing
Hong Kong Polytechnic University
Hong Kong SAR, China

Jieming Shi*
jieming.shi@polyu.edu.hk
Department of Computing
Hong Kong Polytechnic University
Hong Kong SAR, China

Qing Li
csqli@comp.polyu.edu.hk
Department of Computing
Hong Kong Polytechnic University
Hong Kong SAR, China

Jiannong Cao
csjcao@comp.polyu.edu.hk
Department of Computing
Hong Kong Polytechnic University
Hong Kong SAR, China

## Abstract

Cryptocurrencies are rapidly expanding and becoming vital in digital financial markets. However, the rise in cryptocurrency-related illicit activities has led to significant losses for users. To protect the security of these platforms, it is critical to identify illicit accounts effectively. Current detection methods mainly depend on feature engineering or are inadequate to leverage the complex information within cryptocurrency transaction networks, resulting in suboptimal performance. In this paper, we present DIAM, an effective method for detecting illicit accounts in cryptocurrency transaction networks modeled by directed multi-graphs with attributed edges. DIAM first features an Edge2Seq module that captures intrinsic transaction patterns from parallel edges by considering edge attributes and their directed sequences, to generate effective node representations. Then in DIAM, we design a multigraph Discrepancy (MGD) module with a tailored message passing mechanism to capture the discrepant features between normal and illicit nodes over the multigraph topology, assisted by an attention mechanism. DIAM integrates these techniques for end-to-end training to detect illicit accounts from legitimate ones. Extensive experiments, comparing against 15 existing solutions on 4 large cryptocurrency datasets of Bitcoin and Ethereum, demonstrate that DIAM consistently outperforms others in accurately identifying illicit accounts. For example, on a Bitcoin dataset with 20 million nodes and 203 million edges, DIAM attains an F1 score of 96.55%, markedly surpassing the runner-up's score of 83.92%. The code is available at https://github.com/TommyDzh/DIAM.

## CCS Concepts

• **Computing methodologies** → *Supervised learning by classification.*

*Corresponding Author.


*CIKM '24, October 21–25, 2024, Boise, ID, USA*

ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679707

## Keywords

Illicit Account Detection, Cryptocurrency Transaction Networks, Multigraphs.

**ACM Reference Format:**
Zhihao Ding, Jieming Shi, Qing Li, and Jiannong Cao. 2024. Effective Illicit Account Detection on Large Cryptocurrency MultiGraphs . In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3627673.3679707

## 1 Introduction

Cryptocurrencies, *e.g.*, Ethereum and Bitcoin, are of growing importance, due to the nature of decentralization and pseudo-anonymity based on blockchain technology. As of May 2024, Bitcoin and Ethereum are the top-2 largest cryptocurrencies with $1.5 trillion market capitalization in total [8]. In addition to the cryptocurrency transactions among normal accounts, illicit accounts are also taking advantage of Bitcoin and Ethereum for illegal activities, such as phishing scams [5, 6], and money laundering [33], which put normal users at risk of financial loss and hinder the development of the blockchain ecosystem.

Hence, we study the detection of illicit accounts on cryptocurrency transaction networks. This task is particularly challenging due to the huge number of transactions and the inherent anonymity of cryptocurrency accounts, which lack the portrait information crucial for identifying illicit activities. Some pioneer solutions [1, 5, 6, 25] mainly rely on feature engineering to extract handcrafted features from transactions, which highly depends on domain expertise. There are also studies using Graph Neural Networks (GNNs) for detection [2, 25, 27, 32, 35]. Common GNNs, such as GCNs [18] and GATs [29], mainly rely on the homophily assumption that connected nodes share similar representations and belong to the same class [44]. This may not be true for illicit account detection. Specifically, illicit accounts are usually much fewer than normal accounts, and they may exhibit discrepant patterns over their neighboring accounts, most of which are normal. Such discrepancy should be captured for effective detection of illicit accounts. A recent method [15] adopts transformers to learn account representations from Ethereum transaction sequences for detection, without explicitly considering the network topology of transactions.

As reviewed in Section 2, general graph anomaly detection methods [10, 42] can be customized for illicit account detection, but yield moderate performance in experiments.

To effectively model the cryptocurrency transaction network, we conceptualize it as a *directed multi-graph* with multiple edges connecting nodes, each edge representing a transaction between accounts. These edges are enriched with edge attributes such as transaction timestamps and amounts, enabling comprehensive representation of transactional activities in both Bitcoin and Ethereum networks. An illustrative example is depicted in Figure 1. For instance, edge $e_{10}$ is a transaction from nodes $v_6$ to $v_7$ with transaction timestamp, value, *etc.*, as edge attributes. Multiple edges exist between nodes, *e.g.*, edges $e_4, e_5, e_6$ between $v_3$ and $v_4$, representing three transactions. A node, *e.g.*, $v_4$, has incoming and outgoing transactions as listed in Figure 1. The transaction timestamps indicate the sequential dependency of edges between nodes.

In this paper, we present DIAM, an effective method to Detect Illicit Accounts on directed Multigraphs with edge attributes for cryptocurrencies. In a nutshell, DIAM consists of well-thought-out technical designs to holistically utilize all of the directed multigraph topology, edge attributes, and parallel edge sequential dependencies, as shown in Figure 1. First, DIAM incorporates an *Edge2Seq* module designed to autonomously learn effective representations that maintain the inherent transaction patterns depicted by directed parallel edges with attributes. In particular, Edge2Seq identifies and captures the sequential patterns of transactions by assembling sequences of edge attributes. It then integrates both the attributes of the edges and the dependencies within these sequences into the representations of nodes. To further utilize the multigraph topology and handle the discrepancy issue mentioned above, we then develop an *Multigraph Discrepancy* (*MGD*) module in DIAM. As illustrated in Figure 1, illicit node $v_4$ is closely connected to benign nodes $v_1, v_2, v_3$, while their representations should be discrepant to distinguish $v_4$ from others. To achieve this, we design MGD to propagate not only node representations, but also the discrepancies between nodes, along directed multiple edges, with the help of a dedicated attention mechanism and learnable transformation. In other words, MGD can preserve both similar and discrepant features, which are vital for effective illicit account detection. DIAM stacks multiple MGD modules to consider multi-hop multigraph topology. Finally, assembling all techniques, DIAM is trained in an end-to-end manner, to minimize a cross-entropy loss. We evaluate DIAM against 15 existing solutions over 4 large real-world cryptocurrency datasets of Bitcoin and Ethereum. Extensive experiments validate that DIAM consistently achieves the highest accuracy on all datasets, outperforming competitors often by a significant margin. Summing up, our contributions are as follows:

- We study illicit account detection on transaction networks of cryptocurrencies, and present DIAM, an effective method over large directed multigraphs with edge attributes.
- In DIAM, we develop an Edge2Seq module that automatically encodes edge attributes, edge sequence dependencies, and edge directions into node representations.
- We further design MGD, a multigraph discrepancy module to effectively preserve the representation discrepancies between illicit and benign nodes on the multigraph.

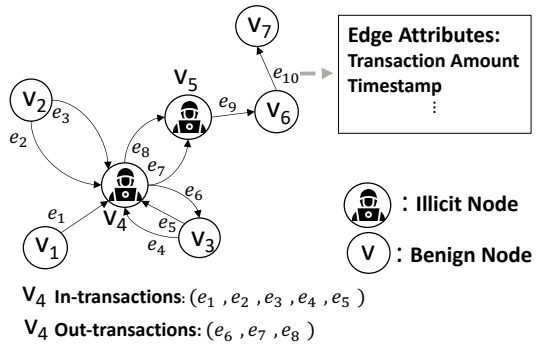



**Figure 1: A directed multigraph with edge attributes.**

- The superiority of DIAM is validated via extensive experiments by comparing 15 baselines on 4 real datasets.

## 2 Related Work

Our work is related to studies on illicit account detection on cryptocurrency, and graph-based anomaly detection.

**Illicit Account Detection on Cryptocurrency.** Early studies mostly rely on tedious feature engineering to obtain statistical features, such as the sum, average, standard deviation of transaction amounts and time [2, 5, 25]. These studies then employ on-the-rack classifiers (*e.g.*, XGBoost [3] and LightGBM [17]) over the extracted features to detect illicit accounts [1, 2, 5]. To further exploit the graph topological characteristics of cryptocurrency transaction networks, recent studies [25, 35] incorporate graph mining methods for illicit account detection. Random-walk based node embedding is adopted in [35], and Node2vec [13] and Ri-walk [24] are used in [25] to extract structural information for illicit account detection. Most of these studies still use on handcrafted node features. A recent method [15] uses transformers to learn expressive representations from Ethereum transaction sequences, but does not exploit the multigraph structure of cryptocurrency transactions. There are GNN-based methods on cryptocurrency transaction networks [2, 19, 27, 32]. An end-to-end GCN is trained in [32] for anti-money laundering in Bitcoin. EdgeProp [27] augments edge attributes in GNNs to identify illicit accounts in Ethereum. In [19], GNNs and self-supervised learning are incorporated to detect phishing scams. These studies usually focus on one cryptocurrency type, either Ethereum or Bitcoin. On the other hand, we exploit the topological and sequential semantics of the directed multigraph data model, and develop techniques to automatically learn deep intrinsic node representations that are highly effective for illicit account detection on both Bitcoin and Ethereum cryptocurrencies.

**Graph-based Anomaly Detection.** There exist studies on anomaly detection over general graphs [9–11, 16, 21, 22, 26, 31, 39, 40, 43], and representative methods are based on classic GNNs, such as GCN [18], Sage [14], and GAT [29]. GINE [16] and TransConv [26] incorporate edge features in GNNs for anomaly detection. However, abnormal nodes may have discrepant features, compared with normal ones [23], and often hide themselves in camouflage [11]. To alleviate the issue, CARE-GNN [11] trains a predictor to measure the similarity between target nodes and their neighborhoods and adopts reinforcement learning for detection. In [9], a new framework is

proposed to use attention mechanism and generative adversarial learning. Camouflage behaviors are captured by subtractive aggregation on GNNs in [43]. PC-GNN [22] samples neighbors from the same class and relieve class imbalance, while meta-learning is used in [10] and decoupling with self-supervised learning is developed in [31]. FRAUDRE [39] takes mean aggregation of neighborhood differences and representations, and develops a loss function to remedy class imbalance for anomaly detection. Note that these methods are designed for relation graphs, and we set the number of relations as 1 to run them on the multigraph data model in this work. Even though these methods can be customized for the illicit account detection problem, they are not catered for the unique characteristics of cryptocurrency transactions and often produce suboptimal performance in experiments.

## 3 Problem Formulation

**Data Model.** Let $G = (V, E, \mathbf{X}_E)$ be a directed multigraph, consisting of (i) a node set $V$ that contains $n$ nodes, (ii) a set of directed edges $E$ of size $m$, and (iii) an edge attribute matrix $\mathbf{X}_E \in \mathbb{R}^{m \times d}$, each row of which is a $d$-dimensional vector serving as the edge attributes to encode the details of the corresponding transaction. In a multigraph $G$, nodes $v$ and $u$ can have parallel edges with different edge attributes. Let $N_{out}(v)$ be the *multiset* of node $v$'s outgoing neighbors. If a node $u$ has more than one edge to $v$, $u$ will have multiple occurrences in $N_{out}(v)$. Similarly, let $N_{in}(v)$ be the multiset of node $v$'s incoming neighbors.

Given a collection of transactions, we can build its directed edge-attributed multigraph as follows. An Ethereum transaction is a message sent from a sender address (i.e., account) $v$, to a receiver address $u$ at a certain time with transaction details, forming a directed edge, *e.g.*, edge $e_{10}$ in Figure 1. Bitcoin transactions are similar but with differences. A bitcoin transaction can contain multiple sender accounts and receiver accounts, who may send or receive different amounts of Bitcoin respectively in the transaction [36]. Given a Bitcoin transaction, we will create a directed edge $e$ from every sender $v$ to every receiver $u$ in the transaction, with the corresponding transaction details from $v$ to $u$ as edge attributes. For interested readers, see [34] for a comprehensive introduction of Bitcoin and Ethereum.

**Problem Definition.** Given a directed multigraph $G = (V, E, \mathbf{X}_E)$ we formulate the problem of illicit account detection on directed multigraphs with edge attributes as a classification task. Let $Y_{\mathcal{L}}$ be the set of the partially observed node labels, and each node label $y_v \in Y_{\mathcal{L}}$ takes value either 1 or 0, indicating the node to be illicit or not. The objective is to learn a binary classifier $f$ that can accurately detect the illicit accounts in the set of unobserved node labels $Y_{\mathcal{U}}$ to be predicted in $G$, $f : G = (V, E, \mathbf{X}_E, Y_{\mathcal{L}}) \mapsto Y_{\mathcal{L}} \cup Y_{\mathcal{U}}$.

Bitcoin and Ethereum are distributed public ledgers recording all transactions anonymously accessible to the public [2, 41], which facilitates the build of multigraphs. For node labels, since the addresses in cryptocurrencies are unique and immutable, there are websites and forums, like WalletExplorer [30] and EtherScan [12], providing illicit label information, e.g., phishing. As described in Section 5.1, we crawl such information as ground-truth labels.

## 4 The Proposed Method

**Overview.** Figure 2 illustrates the DIAM method, which inputs a directed, edge-attributed multigraph $G$ representing a transaction network. The first module in DIAM is Edge2Seq detailed in Section 4.1, which automatically derives the expressive representation of a node by considering the sequences of both incoming and outgoing edges. As shown in Figure 2, for a node $v$ (*e.g.*, $v_4$), Edge2Seq first builds an incoming sequence $X_v^{in}$ and an outgoing sequence $X_v^{out}$ that consist of $v$'s incoming and outgoing edge attributes in chronological order, respectively. Intuitively, $X_v^{out}$ and $X_v^{in}$ describe different sequential transaction patterns of node $v$, when $v$ serves as a sender or a receiver respectively. Then Edge2Seq employs GRUs [7] to learn the sequence representations of both $X_v^{out}$ and $X_v^{in}$, which are then processed by pooling operations, to get representations $\mathbf{h}_{v_{out}}$ and $\mathbf{h}_{v_{in}}$ respectively. Then $\mathbf{h}_{v_{out}}$ and $\mathbf{h}_{v_{in}}$ are concatenated together to be the node representation $\mathbf{h}_v$ of $v$, encapsulating the bidirectional transaction patterns and their temporal dependencies. The node representations $\mathbf{h}_v$ for all $v \in V$ learned by Edge2Seq are then regarded as initial inputs fed into the proposed multigraph discrepancy (MGD) module presented in Section 4.2. In an MGD, a target node $v$ receives messages from its incoming and outgoing neighborhoods separately (*e.g.*, $v_4$ in the multigraph discrepancy module of Figure 2). The incoming and outgoing messages, denoted as $\mathbf{r}_{v_{in}}$ and $\mathbf{r}_{v_{out}}$, contain both neighbor representations and their *discrepancies* with the target node, in order to preserve distinguishable features for illicit account detection. Then an attention mechanism is designed in MGD to integrate $v$'s representation $\mathbf{z}_v$, incoming message $\mathbf{r}_{v_{in}}$, and outgoing message $\mathbf{r}_{v_{out}}$ together via attentions $\alpha_{v,1}$, $\alpha_{v,2}$, and $\alpha_{v,3}$. DIAM stacks multiple MGD layers to consider multi-hop multigraph topology to learn more expressive discrepancy-aware node representations. The last component of DIAM is a multilayer perceptron (MLP) to learn illicit probability $p_v$ of node $v$. DIAM is trained to minimize a binary cross-entropy loss in Section 4.3.

### 4.1 Edge2Seq

High-quality node representations are essential for detecting illicit accounts. As discussed in Section 1, cryptocurrency accounts often lack profile information, and illicit accounts in transaction networks frequently disguise their native features to blend in with legitimate nodes, a challenge exacerbated by the decentralized and pseudo-anonymous nature of cryptocurrencies. Current methods mostly rely on feature engineering to extract statistical features, which demands domain expert knowledge.

Here we develop Edge2Seq to automatically generate high-quality node representations that capture the essential transaction patterns within nodes. Briefly, Edge2Seq combines edge attributes (transaction details), parallel edge sequential dependencies (transaction relationships), and edge directions (transaction flow directions) within the directed edge-attributed multigraph data model. In particular, Edge2Seq treats the incoming and outgoing edges of a node distinctly, recognizing that they represent different directions of money flow, which is key for identifying transaction patterns in cryptocurrency networks. To effectively discern these directional transaction patterns, our model constructs separate incoming and outgoing sequences for each node $v$ in multigraph $G$, sorted by

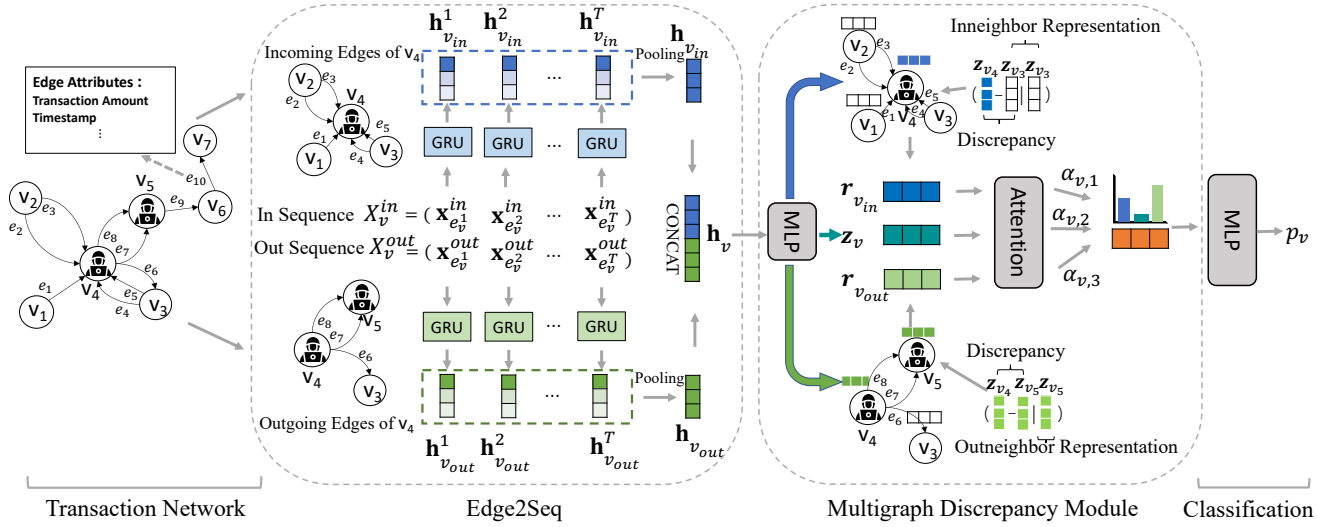



**Figure 2: The DIAM framework with an input transaction network modeled as a directed multigraph with edge attributes.**

timestamps. We employ GRUs to process these sequences to learn representations. These representations then serve as the node representations for further training. Next, we detail Edge2Seq in two main steps: edge sequence generation and edge sequence encoding.

**Edge Sequence Generation.** Given a node $v$ of the input multigraph $G$, Edge2Seq first builds two sequences for it. In particular, for all outgoing edges of $v$, Edge2Seq sorts the outgoing edges in chronological order according to the timestamps on edges, and gets $E_v^{out} = (e_v^1, e_v^2 ..., e_v^T)$, the sequence of $T$ sorted outgoing edges of $v$. For instance, in Figure 1, node $v_4$ has outgoing edge sequence $(e_6, e_7, e_8)$. Edge2Seq then extracts the corresponding edge attributes accordingly, and builds the outgoing edge attribute sequence of $v$, $X_v^{out} = (\mathbf{x}_{e_v^1}, \mathbf{x}_{e_v^2}, ..., \mathbf{x}_{e_v^T})$. Then, similarly, we also build an incoming edge attribute sequence $X_v^{in}$. Obviously, sequences $X_v^{out}$ and $X_v^{in}$ of node $v$ consider both edge sequence and edge attributes, and also utilize parallel edges between $v$ and its neighbors. Intuitively, $X_v^{out}$ (resp. $X_v^{in}$) represents the transaction behaviors of node $v$ when $v$ serves as a sender (resp. receiver).

Note that an account can participate in thousands of transactions, resulting to substantially long sequences. The number of transactions of accounts commonly follows the power-law distribution [4]. In other words, only a few nodes have excessively long sequences $X_v^{out}$ or $X_v^{in}$. To reduce the computational costs associated with handling extremely long sequences, we apply a common trick [20, 28] by limiting the sequence length to be at most $T_{\max}$ and retaining the most recent edges. In experiments, we study the impact of varying $T_{\max}$. In addition, for nodes without any incoming or outgoing edges, we add self-loops to generate sequences.

**Edge Sequence Encoding.** After generating sequences $X_v^{out}$ and $X_v^{in}$ for node $v$ in the input multigraph $G$, we encode the sequences into the representation of node $v$. We use node $v$'s length-$T$ outgoing sequence $X_v^{out} = (\mathbf{x}_{e_v^1}, \mathbf{x}_{e_v^2}, ..., \mathbf{x}_{e_v^T})$ to explain the process, and that of $X_v^{in}$ naturally follows. In particular, as shown in Eq. (1), starting from $t = 1$, until the end of the length-$T$ sequence $X_v^{out}$, we first apply a linear transformation on edge attributes $\mathbf{x}_{e_v^t}$ to get $\mathbf{z}_{e_v^t}^{out}$ via a one-layer MLP with learnable $\mathbf{W}_{out}$ and $\mathbf{b}_{out}$. Then we apply GRU over $\mathbf{z}_{e_v^t}^{out}$ and the $(t-1)$-th hidden state $\mathbf{h}_{v_{out}}^{t-1}$, to get the updated $\mathbf{h}_{v_{out}}^t$ at the $t$-th position of sequence $X_v^{out}$:

$$\begin{aligned} \mathbf{z}_{e_v^t}^{out} &= \mathbf{W}_{out}\mathbf{x}_{e_v^t} + \mathbf{b}_{out}, \\ \mathbf{h}_{v_{out}}^t &= \mathrm{GRU}_{out}(\mathbf{z}_{e_v^t}^{out}, \mathbf{h}_{v_{out}}^{t-1}), \end{aligned} \tag{1}$$

where $\mathbf{W}_{out} \in \mathbb{R}^{\frac{c}{2} \times d}$ and $\mathbf{b}_{out} \in \mathbb{R}^{\frac{c}{2}}$ are learnable parameters, and $c$ is the representation dimension. By convention, the initial hidden state of GRU, $\mathbf{h}_{v_{out}}^{t=0}$, is set to be zero.

Essentially, we generate a representation $\mathbf{h}_{v_{out}}^t$ for each outgoing edge at position $t \in [1, T]$ of sequence $X_v^{out}$. Then we apply element-wise max-pooling to get the representation $\mathbf{h}_{v_{out}}$ of sequence $X_v^{out}$,

$$\mathbf{h}_{v_{out}} = \varphi_{pool}{}_{\forall t \in [1,T]}(\mathbf{h}_{v_{out}}^t), \tag{2}$$

where $\varphi_{pool}(\cdot)$ is the max-pooling operation.

Then, we apply a similar procedure over the incoming sequence $X_v^{in}$ of node $v$ by using another $\mathrm{GRU}_{in}$, to get the incoming sequence representation $\mathbf{h}_{v_{in}}$. Finally, we obtain the representation $\mathbf{h}_v$ of node $v$ by concatenating $\mathbf{h}_{v_{in}}$ and $\mathbf{h}_{v_{out}}$ in Eq. (3).

$$\mathbf{h}_v = \mathbf{h}_{v_{out}} || \mathbf{h}_{v_{in}}. \tag{3}$$

Since we obtain the representations $\mathbf{h}_{v_{in}}$ and $\mathbf{h}_{v_{out}}$ based on the incoming and outgoing edge attribute sequences of $v$ respectively, inherently node representation $\mathbf{h}_v$ can preserve the hidden transaction patterns of node $v$ in both directions.

## 4.2 MGD

Note that the representation $\mathbf{h}_v$ of node $v$ obtained by Edge2Seq in Section 4.1 only captures $v$'s individual transaction features contained in its outgoing and incoming edges. In this section, we aim to leverage the multi-hop multigraph topology to enhance the representation for illicit account detection. One straightforward way is to adopt conventional GNNs. However, as explained, conventional

GNNs heavily rely on the assumption that similar nodes tend to connect to each other and share similar representations [14], which may be less effective on the task of illicit account detection on multigraphs [9, 11]. Intuitively, illicit and normal nodes, despite potential close connections, should have distinct representations. An effective model should be capable of learning these discrepant representations between closely connected normal and illicit nodes.

To accomplish this, we present a Multigraph Discrepancy module (MGD), with three technical designs: (i) directed discrepancy-aware message passing with sum pooling, (ii) layer-wise learnable transformations, and (iii) an attention mechanism over directional representations, to learn expressive representations.

The MGD is discrepancy-aware, transforming and transmitting not just node representations, but also the discrepancies between nodes through a proposed message passing mechanism on multigraphs. Moreover, for a target node $v$, MGD separately considers the discrepancies of its incoming and outgoing neighbors, acknowledging that a node's behavior can vary as either a sender or receiver of transactions. As confirmed in our experiments, MGD outperforms existing counterparts in illicit account detection.

In DIAM, let $L$ be the total number of MGD modules stacked together. The first MGD layer takes the representations $\mathbf{h}_v$ of nodes $v \in V$ learned by Edge2Seq in Section 4.1 as input. Without ambiguity, let $\mathbf{h}_v{}^{(\ell=0)}$ represent the input of the first MGD layer. As shown in Eq. (4), the $\ell$-th MGD first applies a layer-wise linear transformation with learnable weights $\mathbf{W}_2^{(\ell)}$ and $\mathbf{b}_2^{(\ell)}$ to convert representation $\mathbf{h}_v^{(\ell-1)}$ to intermediate $\mathbf{z}_v^{(\ell)}$ via a one-layer MLP. Then for an in-neighbor $u \in N_{in}(v)$, the message passed from $u$ to $v$ in the $\ell$-th MGD is $\mathbf{W}_3^{(\ell)}(\mathbf{z}_u^{(\ell)} || (\mathbf{z}_v^{(\ell)} - \mathbf{z}_u^{(\ell)}))$, which includes both in-neighbor $u$'s representation $\mathbf{z}_u^{(\ell)}$ and its discrepancy $(\mathbf{z}_v^{(\ell)} - \mathbf{z}_u^{(\ell)})$ with target node $v$, followed by a learnable linear transformation using $\mathbf{W}_3^{(\ell)}$. Aggregating all such information for every $u \in N_{in}(v)$, we obtain $\mathbf{r}_{v_{in}}^{(\ell)}$ that is the discrepancy-aware incoming message that node $v$ receives from its incoming neighborhood. Note that $N_{in}(v)$ is a multiset of node $v$'s in-neighbors in the input multigraph $G$, and thus, we consider parallel edges during the message passing. Similarly, we can get the discrepancy-aware outgoing message $\mathbf{r}_{v_{out}}^{(\ell)}$ that $v$ receives from its outgoing neighborhood $N_{out}(v)$, as shown in Eq. (4). Specifically, $\mathbf{r}_{v_{out}}^{(\ell)}$ considers every out-neighbor $u$'s representation as well as its discrepancy with $v$. Finally, we develop an attention mechanism to integrate the three aspects, namely $v$'s representation $\mathbf{z}_v^{(\ell)}$, discrepancy-aware incoming and outgoing messages $\mathbf{r}_{v_{in}}^{(\ell)}$ and $\mathbf{r}_{v_{out}}^{(\ell)}$, via attention $\alpha_{v,1}$, $\alpha_{v,2}$, and $\alpha_{v,3}$, to get node representation $\mathbf{h}_v^{(\ell)}$ at the $\ell$-th MGD.

$$
\begin{aligned}
\mathbf{z}_v^{(\ell)} &= \mathbf{W}_2^{(\ell)}\mathbf{h}_v^{(\ell-1)} + \mathbf{b}_2^{(\ell)}, \\
\mathbf{r}_{v_{in}}^{(\ell)} &= \sum_{\forall u \in N_{in}(v)} \mathbf{W}_3^{(\ell)}(\mathbf{z}_u^{(\ell)} || (\mathbf{z}_v^{(\ell)} - \mathbf{z}_u^{(\ell)})), \\
\mathbf{r}_{v_{out}}^{(\ell)} &= \sum_{\forall u \in N_{out}(v)} \mathbf{W}_3^{(\ell)}(\mathbf{z}_u^{(\ell)} || (\mathbf{z}_v^{(\ell)} - \mathbf{z}_u^{(\ell)})), \\
\mathbf{h}_v^{(\ell)} &= \alpha_{v,1}\mathbf{z}_v^{(\ell)} + \alpha_{v,2}\mathbf{r}_{v_{in}}^{(\ell)} + \alpha_{v,3}\mathbf{r}_{v_{out}}^{(\ell)},
\end{aligned}
\tag{4}
$$

where $N_{in}(v)$ and $N_{out}(v)$ are the multisets of $v$'s incoming and outgoing neighbors respectively; $\mathbf{W}_2^{(\ell)} \in \mathbb{R}^{c\times c}$, $\mathbf{b}_2^{(\ell)} \in \mathbb{R}^c$, and $\mathbf{W}_3^{(\ell)} \in \mathbb{R}^{c\times 2c}$ are learnable parameters; $\alpha_{v,1}$, $\alpha_{v,2}$, and $\alpha_{v,3}$ are attention weights.

Attentions $\alpha_{v,1}$, $\alpha_{v,2}$, and $\alpha_{v,3}$ are calculated by Eq. (5). A larger attention weight indicates that the corresponding aspect is more important in the message passing process, which provides a flexible way to aggregate the messages in Eq. (4).

$$
\begin{aligned}
&w_{v,1} = \sigma(\mathbf{z}_v^{(\ell)} \cdot \mathbf{q}); w_{v,2} = \sigma(\mathbf{r}_{v_{in}}^{(\ell)} \cdot \mathbf{q}); w_{v,3} = \sigma(\mathbf{r}_{v_{out}}^{(\ell)} \cdot \mathbf{q}), \\
&\alpha_{v,k} = \mathrm{softmax}((w_{v,1}, w_{v,2}, w_{v,3}))_k,
\end{aligned}
\tag{5}
$$

where $\sigma$ is LeakyReLU activation, $\mathbf{q} \in \mathbb{R}^c$ is the learnable attention vector, softmax is a normalization, and $k = 1, 2, 3$.

**Discussion.** There are several ways to handle the discrepancy issue in literature. Here we highlight the technical differences of MGD compared with existing work [9, 11, 23, 31, 39]. Moreover, we experimentally compare MGD with these methods in Section 5. Compared with our MGD, the counterpart in FRAUDRE, dubbed as FRA, (Eq. (2) in [39]) does not have the latter two designs in MGD and uses mean pooling. As analyzed in [37], sum pooling yields higher expressive power than mean pooling, particularly for multiset neighborhoods of multigraphs in this paper. Further, the attention mechanism and learnable layer-wise transformations in MGD enable the flexible pass and aggregation of both incoming and outgoing discrepancy-aware messages along parallel edges. Thus, MGD is technically different from FRAUDRE. In [9], GDN *only* aggregates the representation differences between a target node and its neighbors, while omitting neighbor representations themselves (Eq. (1) and (2) in [9]). Contrarily, our MGD passes richer messages containing *both* neighbor discrepancies and neighbor representations. There are also different methodologies in [11, 23, 31]. In [11, 23], they train samplers to identify discrepant neighbors, *e.g.*, via reinforcement learning in [11]. DCI [31] adopts self-supervised learning and clustering to decouple representation learning and classification. In experiments, DIAM outperforms these existing methods for illicit account detection on directed multigraphs with edge attributes, validating the effectiveness of our designs in MGD.

## 4.3 Objective

DIAM works in an end-to-end manner to detect illicit accounts on directed multigraphs with edge attributes. At the last $L$-th MGD layer of DIAM, we get the final representations $\mathbf{h}_v^{(L)}$ of nodes $v$. For all labeled nodes $v$, we send their representations into a binary classifier, which is a 2-layer MLP network with a sigmoid unit as shown in Eq. (6), to generate the illicit probability $p_v$ of a node $v$. Obviously, $1 - p_v$ is the normal probability of node $v$.

$$
p_v = \mathrm{sigmoid}(\mathrm{MLP}(\mathbf{h}_v^{(L)})) \tag{6}
$$

We adopt the standard binary cross-entropy loss for training:

$$
\mathrm{Loss}(\mathbf{\Theta}) = -\sum_{y_v \in Y_{\mathcal{L}}} (y_v \log(p_v) + (1 - y_v)\log(1 - p_v)), \tag{7}
$$

where $Y_{\mathcal{L}}$ is the set of groundtruth node labels, $y_v$ is the label of node $v$, $\mathbf{\Theta}$ contains all parameters of DIAM.

**Analysis.** We provide the time complexity analysis of DIAM. In Edge2Seq, the time complexities of one-layer MLP transformation, GRU, max-pooling are $O(T_{\max}|V|dc)$, $O(T_{\max}|V|c^2)$, and $O(T_{\max}|V|c)$ respectively, where $T_{\max}$ is the maximum sequence length, $|V|$ is the number of nodes, $d$ and $c$ are the dimensions of edge attributes and hidden representations. The overall time complexity of Edge2Seq is $O(T_{\max}|V|c(c+d))$. In MGD, the time complexity of message passing operation on incoming and outgoing neighbors is the same as vanilla message passing-based GNNs like Sage [14] and GAT [29], which is $O(|V|c^2 + |E|c)$, where $|E|$ is the number of edges. The time complexity of attention mechanism is $O(|V|c)$, and the time complexity of the two-layer MLP is $O(|V|(c^2 + 2c))$. Combining the time of all above components, we get the time complexity of DIAM as $O(T_{\max}|V|c(c+d) + |E|c)$.

## 5 Experiments

We experimentally evaluate DIAM against 15 baselines on 4 real-world transaction networks of cryptocurrency datasets, with the aim to answer the following 5 research questions:

- **RQ1:** How does DIAM perform in terms of effectiveness, compared with existing state of the art?
- **RQ2:** How does the MGD module perform, compared with existing counterparts?
- **RQ3:** How does the Edge2Seq module perform, compared with manual feature engineering?
- **RQ4:** How is the training efficiency of DIAM?
- **RQ5:** How does DIAM perform in sensitivity analysis?

### 5.1 Experimental Setup

**Datasets.** We evaluate on 4 large cryptocurrency datasets, including 2 Ethereum datasets and 2 Bitcoin datasets. The statistics of the datasets are listed in Table 1. The first three datasets are from existing works, and we create the last largest Bitcoin dataset with more than 20 million nodes and 203 million edges. We obtain ground-truth labels of the datasets by crawling illicit and normal account labels from reliable sources, including Etherscan [12] and WalletExplorer [30]. Ethereum-S [38] and Ethereum-P [2] are two Ethereum transaction networks. In both datasets, every edge has two attributes: transaction amount and timestamp. The labeled illicit nodes are the addresses that conduct phishing scams in these two datasets. For Ethereum-P dataset from [2], it only contains illicit node labels. We enhance the dataset by identifying the benign accounts (*e.g.*, wallets and finance services) in Ethereum-P from Etherscan [12] as normal node labels. Bitcoin-M [33] contains the first 1.5 million transactions from June 2015. As explained in Section 3, a Bitcoin transaction can involve multiple senders and receivers. After built as a multigraph, Bitcoin-M has about 2.5 million nodes and 14 million edges. In Bitcoin-M, an edge has 5 attributes: input amount, output amount, number of inputs, number of outputs, and timestamp. We build the largest Bitcoin-L based on all transactions happened from June to September 2015. Bitcoin-L has more than 20 million nodes and 200 million edges, and each edge has 8 attributes: input amount, output amount, number of inputs, number of outputs, fee, total value of all inputs, total value of all outputs, and timestamp. We obtain the labeled data in Bitcoin-M and Bitcoin-L by crawling from WalletExplorer [30]. Following [33], Bitcoin addresses belonging to gambling and mixing services are regarded as illicit accounts due to their strong association with money laundering, while the addresses in other types are normal accounts. Parallel edges between nodes are common in the datasets. For instance, in Ethereum-P, there are 5,353,834 connected node pairs, and 1,287,910 of them have more than one edge (24.06%).

**Table 1: Statistics of the datasets.**

| Dataset | #Nodes | #Edges | #Edge attribute | #Illicit | #Normal | Illicit:Normal |
|---|---|---|---|---|---|---|
| Ethereum-S [38] | 1,329,729 | 6,794,521 | 2 | 1,660 | 1,700 | 1:1.02 |
| Ethereum-P [2] | 2,973,489 | 13,551,303 | 2 | 1,165 | 3,418 | 1:2.93 |
| Bitcoin-M [33] | 2,505,841 | 14,181,316 | 5 | 46,930 | 213,026 | 1:4.54 |
| Bitcoin-L | 20,085,231 | 203,419,765 | 8 | 362,391 | 1,271,556 | 1: 3.51 |

**Baselines.** We compare with 15 competitors in 3 categories, which are reviewed in Section 2.

- *Cryptocurrency illicit account detection methods,* including Pdetector [2], SigTran [25], EdgeProp [27], BERT4ETH [15].
- *Graph-based anomaly detection methods,* including CARE-GNN [11], DCI [31], PC-GNN[22], GDN from AEGIS [9], and FRAUDRE [39]. Specifically, the baseline GDN is a message passing module in AEGIS, while AEGIS itself is unsupervised and thus not compared in the supervised setting.
  CARE-GNN, PC-GNN, and FRAUDRE are designed for relation graphs, and we set the number of relations as 1, to run them.
- *GNN models,* including GCN [18], Sage [14], GAT [29], GATE [29], GINE [16], and TransConv [26].

**Implementation Details.** We implement DIAM and GNN-based models using Pytorch and Pytorch Geometric. We also use Pytorch to implement GDN and Pdetector following the respective papers. For the other competitors, we use the codes provided by the authors. All experiments are conducted on a Linux server with Intel Xeon Gold 6226R 2.90GHz CPU and an Nvidia RTX 3090 GPU card. For the baselines requiring initial node features as input, following the way in [25], we obtain node features, such as node degree and total received/sent amount, by feature engineering for the baselines. Particularly, in this way, we get 48, 48, 69, and 89 node features for datasets Ethereum-P, Ethereum-S, Bitcoin-M, and Bitcoin-L respectively. In terms of Pdetector, we extract the 8 specific node features suggested in its paper [2] for its training to make a fair comparison. GDN, EdgeProp, as well as the GNN-based models, are not originally designed for the binary classification task in this paper. Therefore, we regard them as the encoder to generate node representations, which are then sent to a 2-layer MLP classifier with the same objective in Section 4.3.

**Parameter Settings.** We set embedding dimension ($c = 128$), the number of GNN layers (2), learning rate (0.001), dropout rate (0.2). In DIAM, we set maximum sequence length $T_{max} = 32$. We study the impact of $T_{max}$ in Section 5.5. For all methods, we adopt Adam optimizer, mini-batch training [14] with batch size 128. If not specified, rectified linear units (ReLU) is used as the activation function. For all GNN models, GDN, EdgeProp, and our method requiring neighborhood sampling, given a target node, we randomly sample its 1 and 2-hop neighbors with sample size 25 and 10 respectively. For other settings in baselines, we follow the instructions in their respective papers. The number of training epochs is set as 30 in Ethereum-S, Ethereum-P, and Bitcoin-M, and set as 10 in Bitcoin-L.

**Table 2: Overall results on all datasets (in percentage %). Bold: best. Underline: runner-up. Relative improvements by DIAM over runner-ups in brackets.**

| Method | Ethereum-S | | | | Ethereum-P | | | | Bitcoin-M | | | | Bitcoin-L | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | AUC | Precision | Recall | F1 | AUC | Precision | Recall | F1 | AUC | Precision | Recall | F1 | AUC |
| **GCN** | 81.21 | 96.35 | 88.09 | 87.52 | 86.07 | 80.15 | 82.97 | 87.98 | 79.90 | 81.21 | 80.49 | 88.33 | 80.11 | 83.35 | 81.68 | 88.72 |
| **Sage** | 92.92 | 89.95 | 91.39 | 91.71 | 90.49 | 91.14 | 90.81 | 94.04 | 87.17 | 83.27 | 85.16 | 90.28 | 83.16 | 84.79 | 83.92 | 89.93 |
| **GAT** | 85.99 | 93.55 | 89.60 | 89.52 | 85.11 | 85.37 | 85.06 | 90.23 | 86.16 | 81.45 | 83.71 | 89.27 | 79.45 | 65.73 | 71.80 | 80.44 |
| **GATE** | 66.49 | 90.66 | 76.70 | 73.62 | 88.66 | 85.41 | 86.98 | 90.95 | 71.28 | 67.06 | 68.96 | 80.52 | 67.76 | 36.86 | 47.58 | 65.87 |
| **GINE** | 75.65 | 88.63 | 81.58 | 80.66 | 82.45 | 81.75 | 82.02 | 88.07 | 64.68 | 61.88 | 63.14 | 77.17 | 70.06 | 55.45 | 61.70 | 74.27 |
| **TransConv** | 90.97 | 86.16 | 88.47 | 89.00 | 84.92 | 91.25 | 87.95 | 93.03 | 70.55 | 56.43 | 62.62 | 75.61 | 73.93 | 64.09 | 68.52 | 78.79 |
| **GDN** | 85.55 | 83.79 | 84.63 | 85.14 | 82.42 | 84.00 | 83.19 | 89.13 | 81.56 | 74.76 | 77.99 | 85.51 | 73.68 | 45.92 | 56.57 | 70.62 |
| **CARE-GNN** | 79.81 | 88.53 | 83.93 | 83.61 | 72.72 | 82.76 | 77.41 | 86.42 | 33.41 | 71.93 | 45.36 | 70.04 | 31.02 | 73.29 | 43.57 | 63.50 |
| **DCI** | 71.77 | 92.18 | 80.71 | 78.86 | 75.44 | 76.95 | 76.19 | 84.47 | 82.19 | 51.47 | 63.30 | 74.50 | 81.18 | 51.77 | 62.53 | 73.91 |
| **PC-GNN** | 83.34 | 81.28 | 82.26 | 82.87 | 79.18 | 89.38 | 83.96 | 90.93 | 36.89 | 73.01 | 48.87 | 72.72 | 30.04 | 82.68 | 44.07 | 64.01 |
| **FRAUDRE** | 73.3 | 95.26 | 82.83 | 81.1 | 84.06 | 80.28 | 82.10 | 82.89 | 36.05 | 72.97 | 48.23 | 72.27 | 33.46 | 76.67 | 46.59 | 66.69 |
| **SigTran** | 87.10 | 93.33 | 90.11 | 90.23 | 69.41 | 55.47 | 61.66 | 73.90 | 75.97 | 52.24 | 61.91 | 74.30 | 83.25 | 75.22 | 79.03 | 85.45 |
| **Pdetector** | 79.27 | 91.60 | 84.99 | 84.65 | 82.37 | 83.58 | 82.97 | 88.98 | 80.43 | 58.77 | 67.92 | 77.81 | 77.08 | 52.76 | 62.64 | 74.14 |
| **EdgeProp** | 81.59 | 85.36 | 83.30 | 83.38 | 89.49 | 91.78 | 90.57 | 94.15 | 73.82 | 69.21 | 71.39 | 81.88 | 72.39 | 67.09 | 69.51 | 79.84 |
| **BERT4ETH** | 85.54 | 87.65 | 86.58 | 86.93 | 88.35 | 80.29 | 84.13 | 88.48 | 80.03 | 59.95 | 68.55 | 78.32 | 84.70 | 77.36 | 80.87 | 86.69 |
| **DIAM** | **97.11** (+4.5%) | **96.68** (+0.3%) | **96.89** (+6.0%) | **96.97** (+5.7%) | **94.82** (+4.8%) | **92.95** (+1.3%) | **93.86** (+3.4%) | **95.66** (+1.6%) | **92.83** (+6.5%) | **90.39** (+8.6%) | **91.59** (+7.6%) | **94.43** (+4.6%) | **97.72** (+15.4%) | **95.40** (+12.5%) | **96.55** (+15.1%) | **97.39** (+8.3%) |

**Evaluation Settings.** We adopt 4 evaluation metrics: Precision, Recall, F1 score, and Area Under ROC curve (AUC for short). All metrics indicate better performance when they are higher. For each dataset, we split all labeled nodes into training, validation, and testing sets with ratio 2:1:1. Each model is trained on the training set. When a model achieves the highest F1 score on the validation set, we report the evaluation results on the testing set as the model's performance. For each method, we train it for 5 times and report the average value of each evaluation metric. We also study the training time and the impact when varying training set size as well as the percentage of illicit node labels.

## 5.2 Overall Effectiveness

To answer RQ1, we report the overall results of DIAM and all competitors on all datasets in Table 2. First, observe that DIAM consistently achieves the highest accuracy by all evaluation metrics over all datasets, outperforming all baselines often by a significant margin. For instance, on Ethereum-S, DIAM achieves 96.89% F1 score, while the F1 of the best competitor Sage is 91.39%, indicating a relative improvement of 6%. On Ethereum-P, DIAM has precision 94.82%, outperforming the best competitor by a relative improvement of 4.8%. On Bitcoin-M and Bitcoin-L, DIAM also achieves the highest accuracy for illicit account detection. In particular, DIAM achieves 91.59% and 96.55% F1 scores on Bitcoin-M and Bitcoin-L, 7.6% and 15.1% relatively higher than the best baselines, respectively. Another observation is that the performance gain of DIAM is larger on the largest Bitcoin-L, *e.g.*, 15.4% precision improvement over the best competitor SigTran as shown in Table 2. The reason is that DIAM with Edge2Seq is able to take advantage of the abundant edge attributes in the multigraph of Bitcoin-L, to automatically extract informative representations for accurate detection of illicit accounts. Existing solutions, such as SigTran, require manual feature engineering, and thus, could not effectively leverage the large-scale data to preserve the intrinsic transaction patterns of accounts. In Section 5.3, we conduct an evaluation to further reveal the effectiveness of Edge2Seq, compared with handcrafted features. We conclude that DIAM achieves superior performance for illicit account detection on cryptocurrencies.

## 5.3 Study on MGD and Edge2Seg

**MGD Evaluation.** As we have discussed in Section 4.2, our MGD is different from existing work. To test the effectiveness of MGD in DIAM and answer RQ2, we replace MGD with existing GNN layers, namely, Sage layer [14], GAT layer [28], GDN layer in AEGIS [9], and FRA layer in FRAUDRE [39], and compare their performance. Figure 3 presents the F1 and AUC results for DIAM across all datasets, using each of the five different GNN layers. Observe that DIAM with MGD always achieves the highest F1 and AUC scores on all datasets, and outperforms GDN, Sage, GAT, and FRA layers. The results demonstrate the effectiveness of our MGD to preserve the differentiable representations of both illicit and benign nodes with the consideration of the discrepancies when conducting message passing over the multigraph topology. In particular, given a target node $v$, Sage and GAT layers do not consider discrepancies, GDN layer only passes and aggregates the representation differences of its neighbors to it. Compared with the FRA layer, our MGD employs sum pooling, layer-wise learnable transformations, and an attention mechanism to flexibly pass and aggregate both incoming and outgoing neighbor discrepancies and neighbor representations. Moreover, among existing GNN layers, GDN layer performs better than Sage, GAT, and FRA layers on Ethereum-P in Figure 3(b), while being inferior on the other three datasets. This indicates that it is also important to propagate and aggregate neighbor representations to target nodes in the input multigraph, rather than only considering node representation differences, for effective illicit account detection.

**Edge2Seq Evaluation.** To answer RQ3, we demonstrate the power of Edge2Seq by interchanging it with the handcrafted features as the input of Sage, GAT, and our MGD, and report the evaluation results on Bitcoin-L in Table 3. Specifically, in Table 3, Manual indicates to have the handcrafted node features introduced in Section 5.1 as the initial input of node representations for training, while Edge2Seq

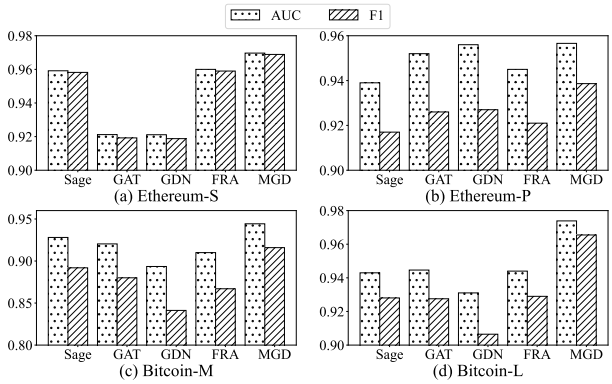



**Figure 3: Compare the MGD module with other GNN layers.**

**Table 3: Manual features v.s., learned representations by Edge2Seq on Bitcoin-L (in percentage %). Relative improvements of Edge2Seq over Manual are in brackets.**

| GNN Layer | Variant | F1 | AUC |
|---|---|---|---|
| Sage | Manual | 83.92 | 89.93 |
| | Edge2Seq | 92.80 (+10.6%) | 94.30 (+4.9%) |
| GAT | Manual | 71.80 | 80.44 |
| | Edge2Seq | 92.75 (+29.2%) | 94.46 (+17.4%) |
| MGD | Manual | 85.29 | 92.39 |
| | Edge2Seq | 96.55 (+13.2%) | 97.39 (+5.4%) |

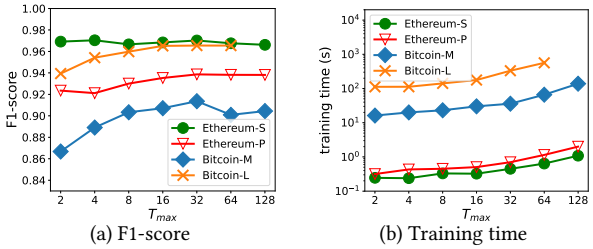
automatically learns node representations by applying GRUs over the incoming and outgoing edge sequences of nodes. As shown in Table 3, comparing against Sage (resp. GAT) with manual features, Sage (resp. GAT) with Edge2Seq always achieves higher F1 and AUC by a significant margin. For instance, GAT with Edge2Seq improves GAT with manual features by a significant margin of 29.2%. The results indicate the superiority of Edge2Seq, compared with manual feature engineering. Further, the result of our MGD with manual features in Table 3 (*i.e.*, DIAM without Edge2Seq) also indicates that Edge2Seq is important for the problem studied in this paper. Our method DIAM assembling Edge2Seq and MGD together obtains the best performance, as shown in Table 3.

## 5.4 Training Efficiency

To answer RQ4, Table 4 reports the average training time per epoch of DIAM and the competitors in seconds on all datasets. First, observe that anomaly detection methods (GDN, CARE-GNN, DCI, PC-GNN, and FRAUDRE) and our method DIAM are generally slower than the common GNN models listed in the first group of Table 4, *e.g.*, GCN and Sage, which is because of the unique designs for illicit/anomaly detection in these methods. However, as reported in Section 5.2, compared with DIAM, common GNN models yield inferior accuracy since they are not dedicated to the task of illicit account detection. Second, DIAM is faster than most graph-based anomaly detection methods. Specifically, on Ethereum-S and Ethereum-P, DIAM is faster than CARE-GNN, DCI, PC-GNN, and FRAUDRE. On Bitcoin-M and Bitcoin-L, DIAM is faster than DCI, PC-GNN, and FRAUDRE. In addition, although EdgeProp is fast, it is not as accurate as DIAM as shown in Section 5.2. The training

**Table 4: Training time per epoch (Seconds)**

| Method | Ethereum-S | Ethereum-P | Bitcoin-M | Bitcoin-L |
|---|---|---|---|---|
| GCN | 0.29 | 0.65 | 14.47 | 274.20 |
| Sage | 0.31 | 0.67 | 13.76 | 270.80 |
| GAT | 0.75 | 1.14 | 18.25 | 307.70 |
| GATE | 0.57 | 0.92 | 13.42 | 130.08 |
| GINE | 0.13 | 0.39 | 8.92 | 104.23 |
| TransConv | 0.22 | 0.66 | 14.72 | 181.82 |
| EdgeProp | 0.17 | 0.39 | 9.73 | 105.22 |
| GDN | 0.39 | 0.74 | 18.47 | 294.64 |
| CARE-GNN | 0.62 | 1.80 | 29.19 | 257.45 |
| DCI | 1.06 | 1.37 | 46.27 | 972.52 |
| PC-GNN | 1.62 | 6.10 | 90.70 | 6525.68 |
| FRAUDRE | 1.34 | 2.58 | 75.40 | 884.20 |
| BERT4ETH | 2.57 | 3.69 | 286.32 | 4107.06 |
| **DIAM** | 0.45 | 0.70 | 35.92 | 330.73 |




(a) F1-score (b) Training time

**Figure 4: Performance Comparison with Varying $T_{max}$**

**Table 5: Ablation Study (in percentage %)**

| Methods | Ethereum-S | | Ethereum-P | | Bitcoin-M | | Bitcoin-L | |
|---|---|---|---|---|---|---|---|---|
| | F1 | AUC | F1 | AUC | F1 | AUC | F1 | AUC |
| **DIAM \MGD** | 95.17 | 95.26 | 82.23 | 88.66 | 69.81 | 81.87 | 72.96 | 81.77 |
| **DIAM \A** | 96.75 | 96.83 | 93.28 | 95.62 | 89.94 | 92.93 | 95.36 | 96.65 |
| **DIAM** | **97.11** | **96.97** | **93.86** | **95.66** | **91.59** | **94.43** | **97.72** | **97.39** |

time per epoch in Table 4 does not include SigTran and Pdetector, since they are not trained in an epoch manner. Considering together the training efficiency in Table 4 and the effectiveness in Table 2, we can conclude that DIAM has superior accuracy for illicit account detection, while being reasonably efficient, on large-scale cryptocurrency datasets.

## 5.5 Sensitivity Analysis

We conduct experiments for sensitivity analysis to answer RQ5.

**Varying the maximum sequence length $T_{max}$.** We vary $T_{max}$ in Edge2Seq from 2 to 128 and report the performance of DIAM and average training time per epoch (seconds) in Figure 4. The result of $T_{max} = 128$ on Bitcoin-L is not reported due to out of GPU memory. In Figure 4a, observe that as $T_{max}$ increases, F1 score on Ethereum-S is relatively stable, F1 score on Ethereum-P and Bitcoin-L increases first and then becomes stable, and F1 score on Bitcoin-M increases first and then decreases after $T_{max}$ is beyond 32. As discussed in [20], the decrease in Bitcoin-M may be caused by the noise introduced among distant elements when considering very long sequences in sequence models. Therefore, we choose $T_{max} = 32$ as default in experiments. In terms of training time per epoch in Figure 4b, when $T_{max}$ increases, it takes more time for training on all datasets, since there are longer sequences to be handled by Edge2Seq. The increasing trend of training time is consistent with the time complexity analysis in Section 4.3.

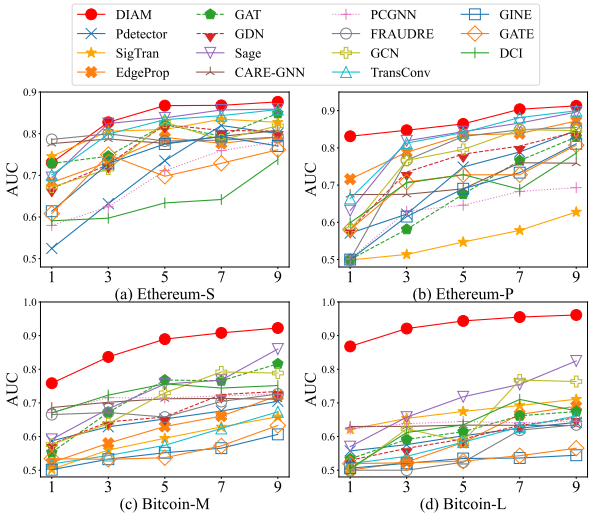



**Figure 5: Varying illicit ratio (%) on all datasets.**

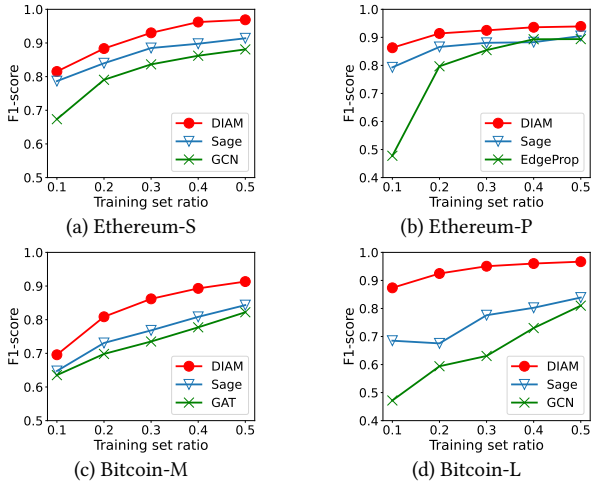



**Figure 6: Varying training set size ratio**

**Ablation Study.** To validate the effectiveness of every component in DIAM, we conduct extra ablation study by evaluating DIAM without MGD in Section 4.2 (denoted as DIAM \MGD), and DIAM without the attention mechanism in Eq. (5) (*i.e.*, set $\alpha_{v,1} = \alpha_{v,2} = \alpha_{v,3} = 1$ in Eq. (4)), denoted as DIAM \A. Table 5 presents their performance compared with the complete version DIAM. First, observe that the performance on all four datasets increases as we add more techniques, validating the effectiveness of the proposed MGD and attention mechanism. Further, note that essentially DIAM \MGD is only with Edge2Seq (*i.e.*, only considering a node's local transaction features), and thus, it has inferior performance as shown in Table 5. This observation indicates the importance of incorporating the multigraph topology for illicit account detection.

**Varying illicit ratio.** As shown in Table 1, the number of illicit accounts is relatively high compared with normal nodes, particularly on Ethereum-S and Ethereum-P datasets. In order to stress test DIAM and the baselines when the illicit node labels are scarce, we have conducted experiments to vary the illicit ratio from 1% to 9%, by randomly sampling a subset of illicit nodes in training on every dataset. The illicit ratio is the proportion of illicit nodes in all labeled training nodes. Figure 5 reports the performance of all methods on all datasets. The overall observation is that DIAM outperforms existing methods under most illicit ratios, except the AUC at 1% on Ethereum-S. As the illicit ratio decreases, the performance of all methods drops on all datasets, since all methods would be under-trained with limited labels. Further, the superiority of DIAM is more obvious on larger datasets. The reason is that our method can better leverage the abundant data to automatically extract meaningful features via Edge2Seq and MGD in DIAM. The results in Figure 5 demonstrate the effectiveness of the proposed DIAM when labels are scarce.

**Varying training data ratio.** To compare the performance of DIAM with baselines under the situation with insufficient training data, we vary the percentage of training data from 10% to 50%. The F1 results on all datasets are reported in Figure 6, where DIAM and the top-2 best baselines per dataset are evaluated. The overall observation is that the F1 scores of all methods decrease as the amount of training data decrease; meanwhile, DIAM keeps achieving the highest effectiveness. For instance, on Ethereum-S in Figure 6a, we compare DIAM with the top-2 baselines Sage and GCN of the dataset (see Table 2). For different sizes of training data, DIAM keeps outperforming the baselines. Similar trends are observed in the other three datasets. Another observation is that the performance of DIAM is relatively stable on the largest Bitcoin-L. Compared to training with 50% of the data, training with 10% of the data only resulted in a 9.6% decrease in model performance. While the two other competitors decreased 18.3% (Sage) and 41.7% (GCN), respectively, which validates the capability of DIAM to leverage abundant data to obtain expressive representations.

## 6 Conclusion

We present DIAM, an effective discrepancy-aware multigraph neural network for the problem of illicit account detection on cryptocurrency transaction networks. The core techniques in DIAM include Edge2Seq that leverages sequence models to automatically learn node representations capturing both incoming and outgoing transaction patterns, and a new Multigraph Discrepancy module MGD, which is able to learn high-quality representations to distinguish the discrepancies between illicit and normal nodes. We conduct extensive experiments on 4 large cryptocurrency datasets, and compare DIAM against 15 existing solutions. The comprehensive experimental results show that DIAM consistently achieves superior performance. Note that the multigraph model in this paper can also describe other transaction networks besides cryptocurrencies, such as online payment data by tech firms, *e.g.*, AliPay and PayPal. Hence, in the future, in addition to cryptocurrency transaction networks, we plan to apply our method to other types of transaction networks to further validate its effectiveness.

## Acknowledgments

The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU25201221, PolyU15205224). The work is supported by NSFC No. 62202404; P0036831; Tencent Technology Co., Ltd. P0048511; P0048213.

## References


[1] Cuneyt Gurcan Akcora, Yitao Li, Yulia R. Gel, and Murat Kantarcioglu. 2020. BitcoinHeist: Topological Data Analysis for Ransomware Prediction on the Bitcoin Blockchain. In *IJCAI*. 4439–4445.

[2] Liang Chen, Jiaying Peng, Yang Liu, Jintang Li, Fenfang Xie, and Zibin Zheng. 2020. Phishing scams detection in ethereum transaction network. *ACM TOIT* 21, 1 (2020), 1–16.

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *ACM SIGKDD*. 785–794.

[4] Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. 2020. Understanding ethereum via graph analysis. *ACM TOIT* 20, 2 (2020), 1–32.

[5] Weili Chen, Xiongfeng Guo, Zhiguang Chen, Zibin Zheng, and Yutong Lu. 2020. Phishing Scam Detection on Ethereum: Towards Financial Security for Blockchain Ecosystem.. In *IJCAI*. 4506–4512.

[6] Weili Chen, Zibin Zheng, Jiahui Cui, Edith Ngai, Peilin Zheng, and Yuren Zhou. 2018. Detecting ponzi schemes on ethereum: Towards healthier blockchain technology. In *WWW*. 1409–1418.

[7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[8] CoinMarketCap. 2023. Cryptocurrency prices, charts and market capitalizations. https://coinmarketcap.com/

[9] Kaize Ding, Jundong Li, Nitin Agarwal, and Huan Liu. 2021. Inductive anomaly detection on attributed networks. In *IJCAI*. 1288–1294.

[10] Kaize Ding, Qinghai Zhou, Hanghang Tong, and Huan Liu. 2021. Few-shot Network Anomaly Detection via Cross-network Meta-learning. In *Proceedings of the Web Conference 2021*. 2448–2456.

[11] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. 2020. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *CIKM*. 315–324.

[12] Etherscan. 2023. Etherscan.io - Ethereum (ETH) Blockchain Explorer. https://etherscan.io/labelcloud

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *ACM SIGKDD*. 855–864.

[14] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1025–1035.

[15] Sihao Hu, Zhen Zhang, Bingqiao Luo, Shengliang Lu, Bingsheng He, and Ling Liu. 2023. BERT4ETH: a pre-trained transformer for ethereum fraud detection. In *Proceedings of the ACM Web Conference 2023*. 2189–2197.

[16] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for Pre-training Graph Neural Networks. In *ICLR*.

[17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *NeurIPS* 30 (2017), 3146–3154.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[19] Shucheng Li, Fengyuan Xu, Runchuan Wang, and Sheng Zhong. 2021. Self-supervised Incremental Deep Graph Learning for Ethereum Phishing Scam Detection. *arXiv preprint arXiv:2106.10176* (2021).

[20] Can Liu, Li Sun, Xiang Ao, Jinghua Feng, Qing He, and Hao Yang. 2021. Intention-aware heterogeneous graph attention networks for fraud transactions detection. In *ACM SIGKDD*. 3280–3288.

[21] Jie Liu, Mengting He, Xuequn Shang, Jieming Shi, Bin Cui, and Hongzhi Yin. 2024. BOURNE: Bootstrapped Self-supervised Learning Framework for Unified Graph Anomaly Detection. In *IEEE ICDE*.

[22] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and Choose: A GNN-based Imbalanced Learning Approach for Fraud Detection. In *Proceedings of the Web Conference 2021*. 3168–3177.

[23] Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. 2020. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *SIGIR*. 1569–1572.

[24] Xuewei Ma, Geng Qin, Zhiyang Qiu, Mingxin Zheng, and Zhe Wang. 2019. RiWalk: Fast structural node embedding via role identification. In *ICDM*. 478–487.

[25] Farimah Poursafaei, Reihaneh Rabbany, and Zeljko Zilic. 2021. SigTran: Signature Vectors for Detecting Illicit Activities in Blockchain Transaction Networks.. In *PAKDD*. 27–39.

[26] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509* (2020).

[27] Da Sun Handason Tam, Wing Cheong Lau, Bin Hu, Qiu Fang Ying, Dah Ming Chiu, and Hong Liu. 2019. Identifying Illicit Accounts in Large Scale E-payment Networks–A Graph Representation Learning Approach. *arXiv:1906.05546* (2019).

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.

[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[30] WalletExplorer. 2023. WalletExplorer.com: Smart bitcoin block explorer. https://www.walletexplorer.com/

[31] Yanling Wang, Jing Zhang, Shasha Guo, Hongzhi Yin, Cuiping Li, and Hong Chen. 2021. Decoupling representation learning and classification for gnn-based anomaly detection. In *SIGIR*. 1239–1248.

[32] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[33] Jiajing Wu, Jieli Liu, Weili Chen, Huawei Huang, Zibin Zheng, and Yan Zhang. 2021. Detecting mixing services via mining bitcoin transaction network with hybrid motifs. *IEEE Transactions on SMC: Systems* (2021).

[34] Jiajing Wu, Jieli Liu, Yijing Zhao, and Zibin Zheng. 2021. Analysis of cryptocurrency transactions from a network perspective: An overview. *Journal of Network and Computer Applications* (2021), 103139.

[35] Jiajing Wu, Qi Yuan, Dan Lin, Wei You, Weili Chen, Chuan Chen, and Zibin Zheng. 2020. Who are the phishers? phishing scam detection on ethereum via network embedding. *IEEE Transactions on SMC: Systems* (2020).

[36] Lei Wu, Yufeng Hu, Yajin Zhou, Haoyu Wang, Xiapu Luo, Zhi Wang, Fan Zhang, and Kui Ren. 2021. Towards Understanding and Demystifying Bitcoin Mixing Services. In *Proceedings of the Web Conference*. 33–44.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *ICLR*.

[38] Zihao Yuan, Qi Yuan, and Jiajing Wu. 2020. Phishing detection on ethereum via learning representation of transaction subgraphs. In *International Conference on Blockchain and Trustworthy Systems*. Springer, 178–191.

[39] Ge Zhang, Jia Wu, Jian Yang, Amin Beheshti, Shan Xue, Chuan Zhou, and Quan Z Sheng. 2021. FRAUDRE: Fraud Detection Dual-Resistant to Graph Inconsistency and Imbalance. In *ICDM*. 867–876.

[40] Tong Zhao, Chuchen Deng, Kaifeng Yu, Tianwen Jiang, Daheng Wang, and Meng Jiang. 2020. Error-Bounded Graph Anomaly Loss for GNNs. In *CIKM*. 1873–1882.

[41] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. 2017. An overview of blockchain technology: Architecture, consensus, and future trends. In *IEEE international congress on big data*. 557–564.

[42] Shuang Zhou, Xiao Huang, Ninghao Liu, Qiaoyu Tan, and Fu-Lai Chung. 2022. Unseen anomaly detection on networks via multi-hypersphere learning. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*. SIAM, 262–270.

[43] Shuang Zhou, Qiaoyu Tan, Zhiming Xu, Xiao Huang, and Fu-lai Chung. 2021. Subtractive aggregation for attributed network anomaly detection. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3672–3676.

[44] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *arXiv preprint arXiv:2006.11468* (2020).