

Inspection-L: A Self-Supervised GNN-Based Money Laundering Detection System for Bitcoin

Wai Weng Lo^a, Mohanad Sarhan^a, Siamak Layeghy^a, Marius Portmann^a

^aThe University of Queensland, Brisbane, Australia

Abstract

Criminals have become increasingly experienced in using cryptocurrencies, such as Bitcoin, for money laundering. The use of cryptocurrencies can hide criminal identities and transfer hundreds of millions of dollars of dirty funds through their criminal digital wallets. However, this is considered a paradox because cryptocurrencies are gold mines for open-source intelligence, allowing law enforcement agencies to have more power in conducting forensic analyses. This paper proposed Inspection-L, a graph neural network (GNN) framework based on self-supervised Deep Graph Infomax (DGI) and Graph Isomorphism Network (GIN), with supervised learning algorithms, namely Random Forest (RF) to detect illicit transactions for AML. To the best of our knowledge, our proposal is the first to apply self-supervised GNNs to the problem of AML in Bitcoin. The proposed method has been evaluated on the Elliptic dataset and shows that our approach outperforms the state-of-the-art in terms of key classification metrics, which demonstrates the potential of self-supervised GNN in cryptocurrency illicit transaction detection.

Keywords: Graph Neural Networks, Machine Learning, Anomaly Detection, Cryptocurrencies

1. Introduction

The advent of the first cryptocurrency - Bitcoin [1] has revolutionized the conventional financial ecosystem as it enables low-cost, anonymous, peer-to-peer cash transfers within and across various borders. Due to their pseudonymity, many cybercriminals, terrorists, and hackers have started to use them for illegal transactions. For example, the WannaCry ransomware attack accepted using Bitcoin [2] as the payment method due to non-traceability. The criminals received about 3.4 million (46.4 BTC) ransom only four days after WannaCry spread [2]. However, it is also a paradox, as cryptocurrencies are gold mines for open-source intelligence as transaction network data is publicly available, allowing law enforcement agencies to conduct forensic analysis of the currency flows. Therefore, effective detection of illicit transactions in Bitcoin transaction graphs is essential for preventing illegal transactions. This problem is challenging for law enforcement agencies owing to the untraceable nature of Bitcoin transactions.

Graph representation learning has shown great potential for detecting money laundering activities using cryptocurrencies. Thus, this paper proposes Inspection-L, a Graph Neural Network (GNN) framework based on Deep Graph Infomax (DGI) [3] and supervised learning algorithms such as Random Forest (RF), to detect illicit transactions for Anti Money Laundering (AML).

GNNs are tailored to applications with graph-structured data, such as social sciences, chemistry, and telecommunications, and

are able to leverage the inherent structure of the graph data by building relational inductive biases into the deep learning architecture. This provides the ability to learn, reason, and generalize from the graph data, inspired by the concept of message propagation [4].

The Bitcoin transaction flow data can naturally be represented in a graph format, where the transactions are mapped to graph nodes, and transaction flows are mapped to the graph edges. Both the topological information as well as the information contained in node features are crucial for detecting illicit transactions.

Specifically, we will investigate the Elliptic dataset [5], a realistic, partially labelled Bitcoin temporal graph-based transaction dataset consisting of real entities belonging to licit (e.g., wallet, miners), illicit entities (e.g. scams, terrorist organizations, ransomware), or unknown categories. The proposed Inspection-L framework aims to detect illegal transactions based on graph representation learning in a self-supervised manner. Current graph machine learning approaches, such as [5], generally apply supervised graph neural network approaches for the detection of illicit transactions. However, the main limitation is that it is limited to only K-layer neighbourhoods, as the extreme sparsity of the training signal causes a threat of overfitting. On the other hand, self-supervised graph neural network algorithms [6][7] allow for every node to have access to structural patterns of the entire graph, which can improve the quality of global representation and can outperform supervised graph machine learning algorithms.

In this paper, we demonstrate how the self-supervised DGI algorithm can be integrated with standard machine learning classification algorithms, i.e. Random Forest, to build an an ef-

Email addresses: w.w.lo@uq.net.au (Wai Weng Lo), m.sarhan@uq.net.au (Mohanad Sarhan), siamak.layeghy@uq.net.au (Siamak Layeghy), marius@itee.uq.edu.au (Marius Portmann)

efficient anti-money laundering detection system. We show that our Inspection-L method outperforms the state-of-the-art in terms of F1 score.

In summary, the key contributions of this paper are:

- Different from most existing works, which typically use supervised graph representation learning to generate node embeddings for illegal transaction detection, we use a self-supervised learning approach to learn the node embeddings without using any labels. This prevents the graph neural network from only capturing the K-layer neighbourhood information as the global view of the graph node information cannot be captured. The use of self-supervised learning DGI can address this limitation, as every node can access the entire graph’s structural pattern and node information.
- The proposed Inspection-L is based on a self-supervised DGI combined with the Random Forest (RF) supervised machine learning algorithms, to capture topological information and node features in the transaction graph to detect illegal transactions. To the best of our knowledge, our proposal is the first to utilize self-supervised GNNs for AML in Bitcoin.
- The comprehensive evaluation of the proposed framework using the Elliptic benchmark datasets demonstrates superior performance compared to other, supervised machine learning approaches.

2. RELATED WORKS

Mark et al. [5] created and published the Elliptic dataset, a temporal graph-based Bitcoin transaction dataset consisting of over 200K Bitcoin node transactions, 234K payment edges, and 49 transaction graphs with distinct time steps. Each of the transaction nodes has been labelled as a “licit”, “illicit”, or “unknown” entity. They evaluated the Elliptic dataset using various machine learning methods, including Logistic Regression (LR), Random Forest (RF), Multilayer Perceptrons (MLP) [8], Graph Convolutional Networks (GCNs) [9] and EvolveGCN [10]. They retrieved a recall score in the illicit category of 0.67 using RF and 0.51 using GCNs.

Yining et al. [11] collected the Bitcoin transaction graph data between July 2014 and May 2017 by running a Bitcoin client to collect the data and used an external trusted source, e.g. “Wallet Explorer” [12], a website that tracks Bitcoin wallets, to label the data. They first highlighted the differences between money laundering and regular transactions using network centrality such as PageRank, clustering coefficient [13], then applied a node2vec-based [14] classifier to classify money laundering transactions. The research also indicated that statistical information, such as in-degree/out-degree, number of weakly connected components, and sum/mean/standard deviation of the output values, could distinguish money laundering transactions from legal transactions. Vassallo et al. [15] focused on the detection of illicit cryptocurrency activities (e.g., scams,

terrorism financing, and Ponzi schemes). The proposed detection framework is based on Adaptive Stacked eXtreme Gradient Boosting (ASXGB), an enhanced variation of eXtreme Gradient Boosting (XGBoost). ASXGB is evaluated using the Elliptic dataset, and the results demonstrate its superiority at both an account and transaction level.

Chaehyeon et al. [16] applied supervised machine learning algorithms to classify illicit nodes in the Bitcoin network. They used two supervised machine learning models, namely, Random Forest (RF) and Artificial Neural Network (ANN) [8] for illegal transaction detection. First, they collected the legal and illegal Bitcoin data from the forum sites “Wallet Explorer” [12] and “Blockchain Explorer” [17]. After that, they performed feature extraction based on the characteristics of Bitcoin transactions, such as transaction fees and transaction size. The extracted features are labelled legal or illegal for supervised training. The results indicated that relatively high F1 scores could be achieved. In their experiment, ANN and RF achieved 0.89 and 0.98 F1 scores, respectively. In [18] proposed using GCNs intertwined with linear layers to classify illicit nodes of the Elliptic dataset [5]. The overall classification accuracy and recall of 97.40% and 0.67, respectively, can be achieved for illicit transaction detection. In [19], the authors used an autoencoder with graph embedding technique to detect mixing and demixing services for Bitcoin cryptocurrency. They first applied graph node embedding to generate the node representation; then, the K-means algorithm was applied to cluster the node embeddings to detect mixing and demixing services. They evaluated the model based on real-world Bitcoin datasets to evaluate the model’s effectiveness, and the results demonstrate that the proposed model can effectively perform demix/mixing service anomaly detection.

Lorenz et al. [20] proposed active learning techniques by using a minimum number of labels to achieve high detection performance of illicit transactions on the Elliptic dataset. In [21], the authors applied unsupervised learning to detect suspicious nodes in the Bitcoin transaction graph. They used various kinds of unsupervised machine learning algorithms such as K-means and Gaussian Mixture models to cluster normal and illicit nodes. However, since the Bitcoin transaction dataset they used does not consist of ground-truth labels, they simply used the internal index to validate the clustering algorithm without confirming that those nodes are actually malicious transactions. Monamo et al. [22] applied trimmed-Kmeans algorithms to detect fraud in the Bitcoin network. To perform feature extraction, they extracted various graph centrality (i.e. in degree, out-degree of the Bitcoin transactions) and currency features (i.e. the total amount sent) for performing Bitcoin transaction clustering. However, similar to [21], since the Bitcoin transaction dataset they used does not consist of ground-truth labels, they just applied clustering performance metrics such as within the sum of squares without validating that those Bitcoin transactions are actually anomalies. Shucheng et al. [23] proposed SIEGE, a self-supervised graph learning approach for Ethereum phishing scam detection. In their model, two pretext tasks have been implemented to generate node embeddings without using labels. Meanwhile, the incremental paradigm has been applied

to capture data distribution change. They collected transactions from Ethereum for about half a year for phishing detection. However, a significant limitation of this approach is it has not considered the Bitcoin context and is only limited to detecting Ethereum phishing scams. Also, they simply applied GCNs[9] in the pretext task phase, which is much less effective than the Weisfeiler-Lehman (1-WL) [24].

In contrast to related studies, our approach can detect not only phishing scams but also other illicit transactions such as terrorist organizations, ransomware and Ponzi schemes by utilizing the Elliptic dataset [5].

3. Dataset

In this paper, we adopted the Elliptic dataset [5] which is the world's largest labelled dataset of bitcoin transactions. The Elliptic dataset [5] consists of 203,769 node as transactions and 234,355 directed transaction payment flows (i.e. transaction inputs, transaction outputs). It also consists of 49 different timestep graphs uniformly spaced with a two-week interval, as illustrated in 1. Each connected component of the transaction consists of a time step that appears on the blockchain in less than three hours.

In the Elliptic dataset [5], 21% of the node entities are labelled as licit, and only 2% are labelled as illicit. The remaining node entities are unlabelled but have node features. These node entities consist of 166 features (AF features), among which the first 94 features contain local information (LF features) of the transactions, including the time step, transaction fees, and the number of inputs or outputs. The remaining 72 features are aggregated features. Those features can be obtained by aggregating transaction information from one-hop backward/forward graph nodes, such as the standard deviation, minimum, maximum, and correlation coefficients of the neighbour transactions for the same information data. More importantly, all features are obtained by using only publicly available information.

4. BACKGROUND

The main innovation of our proposed model is using DGI [25] with our proposed GIN encoder to learn the node embeddings in a self-supervised manner. The node embeddings can then be treated as the enhanced features combined with the AF/LF features for standard supervised RF machine learning algorithms for training and testing. This has a clear advantage over just using AF/LF features as inputs which overall graph-structured patterns are not captured in those features.

Consequently, the current graph-based approaches [5] [18] tried to apply a supervised GCN-based approach for capturing overall graph-structured patterns. However, the main limitation is that it can only capture neighbourhood information of the K layer, and the global view graph and node information cannot be captured, which can cause the threat of overfitting. On the other hand, our Inspection-L approach allows every node to have access to structural patterns of the entire graph, which can capture the neighbourhood information more globally. Also,

the proposed method has considered that the message passing function of [5] [18] are not powerful enough due to its injective natures. Therefore, we proposed our GIN encoder to make the message propagation function to be more robust.

4.1. Background

4.1.1. Graph Neural Networks

GNNs at a very high level can be described as a deep learning approach for graph-based data and a recent and highly promising area of machine learning. A key feature of GNNs is their ability to use the topological graph structure to their advantage through message passing. For each node in a graph, this means aggregating neighbouring node features to leverage a new representation of the current node that takes into account neighbouring information. The output of this process is known as embeddings. For nodes, embeddings are low- or n-dimensional vector representations that capture topological and node properties. This process is able to be performed for edges and graphs in a similar fashion to output edge and graph embeddings. Embeddings are then commonly used for both supervised and unsupervised evaluation tasks, e.g. node classification. The following equation shows the computational process of node embeddings in GCNs.

$$Z^{l+1} = \sigma(X^l W_0^l + \tilde{A} X^l W_1^l) \quad (1)$$

where $X^{(l)} \in \mathbb{R}^{N \times K_l}$ is the embedding at the l -th layer for all the N nodes and $X^{(0)} = X$. $W^{(l)}$ is the weight matrix that will be learnt for the downstream tasks. σ is an activation function that is usually set to be the element-wise ReLU. Let there be L layers of graph convolutions, the output $Z(L)$ is the matrix consisting of all node embeddings after L layer transformation.

4.1.2. Graph Isomorphism Network

Graph Isomorphism Network (GIN) is one of the most powerful GNN was proposed by Xu et al. [31]. The main difference between GIN and other GNNs is the message aggregation function, which is shown below:

$$h_v^{(k)} = \text{MLP}^{(k)} \left(\left(1 + \epsilon^{(k)} \right) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \right) \quad (2)$$

As the message passing function of traditional GCNs is less effective than the Weisfeiler-Lehman (1-WL) [24] algorithms due to the aggregation functions of the GNNs can be the same as the hash function of the 1-WL algorithm. Thus, message passing functions are not necessarily injective. Therefore, GIN [31] was proposed to make the passing function injective, as shown in Equation 2, where $\epsilon^{(k)}$ is a scalar parameter, and MLP stands for a multilayer perceptron. $h_v^{(k)} \in \mathbb{R}^d$ is the embedding of node v_i at the k -th layer, and $h_v^{(0)} = \mathbf{x}_v \cdot \mathcal{N}(v_i)$ is the set of neighboring nodes of node v_i . We can stack k layers to obtain the final node representation $h_v^{(k)}$.

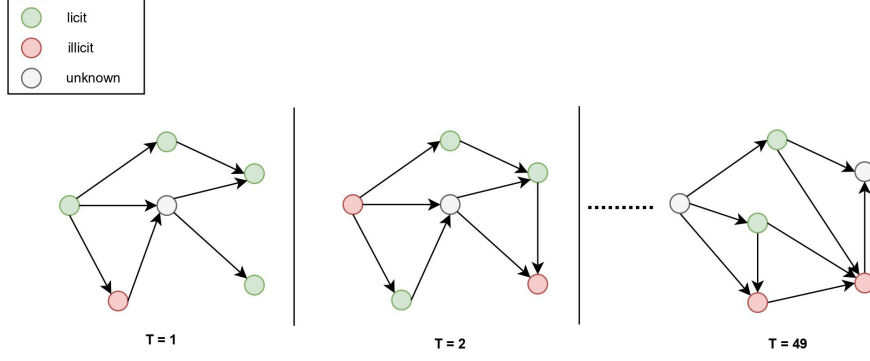


Figure 1: Overview of Elliptic Dataset

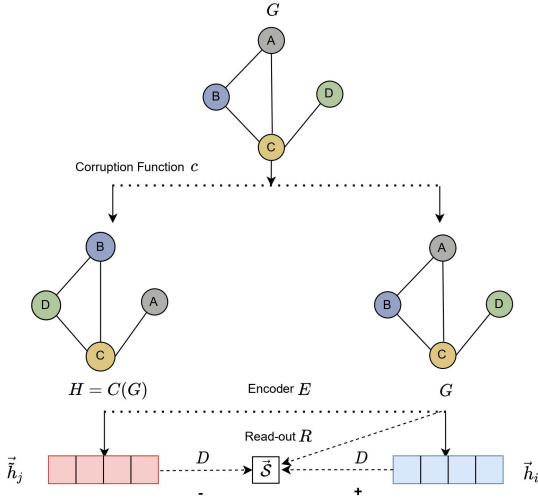


Figure 2: Overview of Deep Graph Infomax

4.1.3. Deep Graph Infomax

Deep Graph Infomax (DGI) [25] is a self-supervised graph representation learning approach. DGI relies on maximizing the mutual information between patch representations and the global graph summary. The patch representations summarise subgraphs, allowing for the preservation of similarities on the patch level. Thus, the trained encoder in DGI can be reused to generate node embeddings for downstream tasks, such as node clustering. Most of the previous works on self-supervised learning with graph representation learning approaches rely only on the random walk strategy [26][27], which is extremely computationally expensive because the number of walks depends on the number of nodes on the graph, making it unsuitable for large graphs. Moreover, the choice of hyperparameters *length of the walk*, *number of walks* can significantly impact the model performance. Overall, DGI does not require any label or random walk techniques and instead guides the model to learn how nodes are connected by simultaneously leveraging local and global information passing in a graph [25]. Therefore, it can be used to train node embeddings in a self-supervised manner.

Figure 2 shows the overall procedure of DGI. The DGI algorithm operates on the following bases:

- A true graph \mathcal{G} with the true nodes and edges that connect them, as well as the real node features associated with each node.
- A corrupted graph \mathcal{H} in which the nodes and edges have been changed using a corruption function. The paper [25] suggests that the corruption function can randomly shuffle each node feature and maintain the same edges as the true graph \mathcal{G} .

The DGI training procedure consists of four components:

- A corruption procedure C that changes the real input graph \mathcal{G} into a corrupted graph $\mathcal{H} = (C(\mathcal{G}))$. This can be done by randomly shifting the node features among the nodes in a real graph \mathcal{G} or by adding and removing an edge from the real graph \mathcal{G} .
- An encoder \mathcal{E} that computes the node embeddings of a corrupted graph and a real graph. This can be done using various graph representation methods, such as Graph Convolutional Networks (GCNs) [9], Graph Attention Networks (GATs) [28] or Graph Transformer Networks (GTNs) [29].
- The node embedding vectors for each node in the real graph are summarised into a single embed vector of the entire graph \bar{s} (global graph summary) by using a read-out function \mathcal{R} for computing the whole graph embeddings.
- A discriminator \mathcal{D} . A discriminator is a logistic non-linear sigmoid function that compares a real node embedding vector \vec{h}_i and a corrupted node embedding $\vec{\hat{h}}_i$ against the whole real graph embedding \bar{s} , and provides a score between 0 and 1. A binary cross-entropy loss objective function [25] can be applied to discriminate the embedding of the real node and the corrupted node to train the encoder \mathcal{E} .

$$\mathcal{L} = \frac{1}{N+M} \left(\sum_{i=1}^N \mathbb{E}_{(\mathbf{X}, \mathbf{A})} [\log \mathcal{D}(\vec{h}_i, \vec{s})] + \sum_{j=1}^M \mathbb{E}_{(\bar{\mathbf{X}}, \bar{\mathbf{A}})} [\log (1 - \mathcal{D}(\vec{h}_j, \vec{s}))] \right) \quad (3)$$

Overall, the pseudocode and overall procedure of our proposed algorithm are shown in Algorithm 1 and Fig. 3.

5. PROPOSED METHOD

For constructing Bitcoin transaction graphs from the dataset, we constructed 49 different Bitcoin transaction graphs (TGs) using time steps so that the nodes can be represented as node transactions and the edges can be represented as flows of Bitcoin transactions. This is a very natural way to represent Bitcoin transactions.

DGI Training. For training the proposed model, the input includes the transaction graphs \mathcal{G} with node features (AF), and the specified number of training epochs K to extract true node embeddings and corrupted node embeddings. Before this, we need to define the corruption function \mathcal{C} to generate the corrupted transaction graphs $\mathcal{C}(\mathcal{G})$ for our GIN encoder to extract the corrupted node embeddings. In this paper, we randomly shuffled the node features among the nodes in real transaction graphs \mathcal{TG} to generate the corrupted transaction graphs. This is done by shuffling the feature matrix in rows \mathbf{X} . Overall, instead of adding or removing edges from the adjacency matrix such that $(\bar{\mathbf{A}} \neq \mathbf{A})$, we use corruption function \mathcal{C} which shuffle the node features such that $(\bar{\mathbf{X}} \neq \mathbf{X})$ but retain the adjacency matrix $(\bar{\mathbf{A}} = \mathbf{A})$.

For each training epoch, in Line 3 to 4, we use our proposed GIN encoder to extract true node and corrupted node embeddings; we proposed a GIN encoder as shown in Fig. 3 with 2 layers MLP which consists of 128 hidden units with ReLU activation function with Batch normalization (as shown in Algorithm 2) [30] for extracting the true node and corrupted node embeddings.

The design of MLPs is motivated by the fundamental goal of a GNN-based model. Ideally, various types of different graph patterns should be distinguishable via the graph encoder, which means that different graph nodes should be able to map different graphs to the embedding space. This implies that it has the capability to solve the graph isomorphism problem, where non-isomorphic graphs should be mapped to different representations.

We applied a full neighbour sampling technique. We specified the size of full two-hop neighbour samples for the GIN encoder with Batch normalization, as DGI benefits from employing wider rather than deeper models [3].

For the read-out function \mathcal{R} , we applied the mean operation on all node embeddings in a real graph and then applied

a sigmoid activation function to compute the whole graph embeddings \bar{s} .

$$\bar{s} = \sigma \left(\frac{1}{n} \sum_{i=1}^n h_i^{(L)} \right) \quad (4)$$

In Algorithm 1, line 5 to 6, for the discriminator \mathcal{D} , we introduced the logistic sigmoid non-linear function which compares a node embedding vector \vec{h}_i against a real whole graph embedding \bar{s} to calculate the score of (\vec{h}_i, \bar{s}) being positive or negative.

$$\mathcal{D}(h_i, \bar{s}) = \sigma(h_i^T \mathbf{w} \bar{s}) \quad (5)$$

$$\mathcal{D}(\bar{h}_i, \bar{s}) = \sigma(\bar{h}_i^T \mathbf{w} \bar{s}) \quad (6)$$

We then applied a binary cross-entropy loss objective function to perform gradient descent as shown in Algorithm 1, line 8. To perform gradient descent, we maximize the score if the node embedding is a true node embedding \vec{h}_i and minimize the score if it is a corrupted node embedding \vec{h}_i compared to the global graph summary generated by the read-out function \mathcal{R} . As a result, we can maximize the mutual information between patch representations and the whole real graph summary. After the training process, the trained encoder can be used to generate new graph embeddings for downstream purposes, in this case, illegal transaction detection.

$$\mathcal{L}_{DGI} = -\frac{1}{2n} \sum_{i=1}^n \left(\mathbb{E}_G \log \mathcal{D}(\mathbf{h}_i^{(L)}, \bar{s}) + \mathbb{E}_{\bar{G}} \log (1 - \mathcal{D}(\bar{\mathbf{h}}_i^{(L)}, \bar{s})) \right) \quad (7)$$

In our experiments, we used all 49 different Bitcoin transaction graphs to train the DGI with the GIN encoder in a self-supervised manner without using any labels. For each training graph, we train 300 epochs using an *Adam* optimizer with a learning rate of 0.0001 to perform gradient descent to update the model weights and hyperparameters as shown in Algorithm 1, line 9.

Supervised Machine Learning Classification. After the DGI training, we can reuse the encoder to generate node embeddings as shown in Algorithm 1, line 10 to train and test the RF classifier as shown in Algorithm 1, line 11. In our experiments, we performed 70:30 splitting for 34 different Bitcoin transactions for training and the remaining 15 bitcoin transaction graphs for testing. For training and testing the RF classifier, we used 100 estimators for training and testing.

1. **Node Embeddings:** After the DGI training, we reuse the encoder to generate node embeddings for training and testing the RF classifier as mentioned above.
2. **Node Embeddings with LF features:** Similar to scenario 1, we also combine local features (i.e, first 94 raw features) with node embeddings generated by the trained encoder for training and testing the RF classifier as mentioned above.

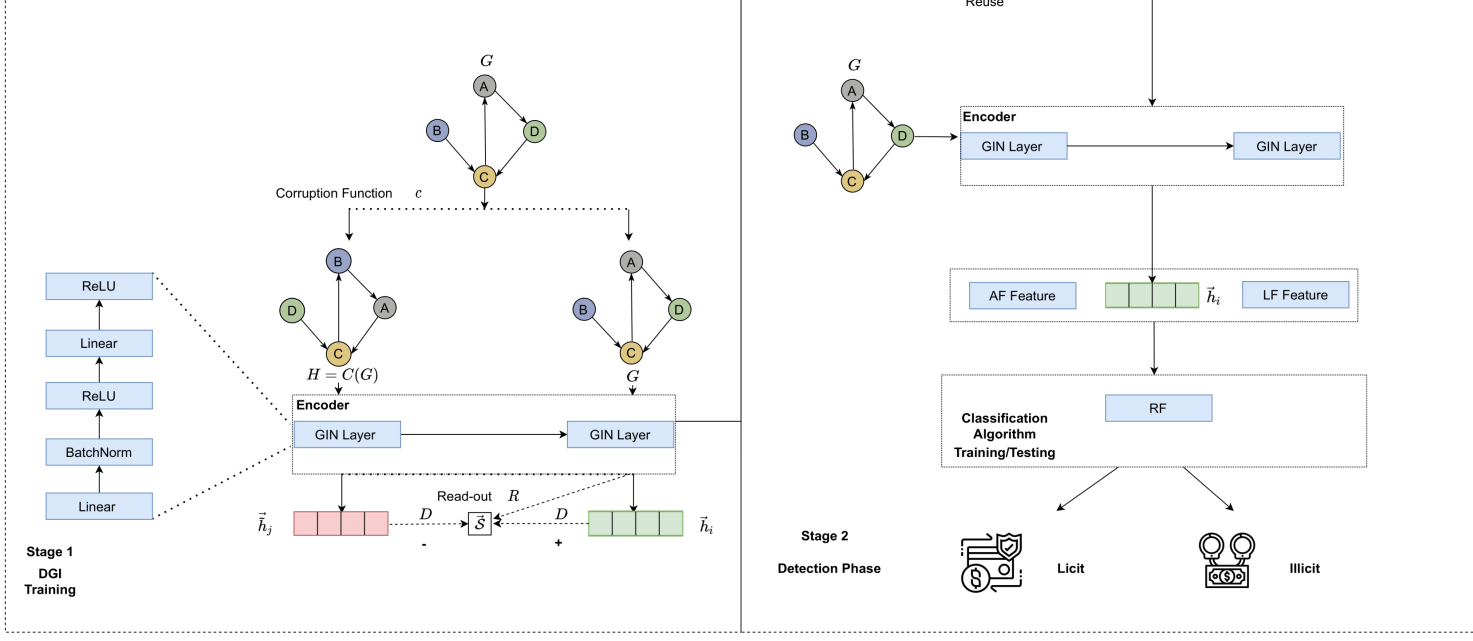


Figure 3: Proposed Method

Algorithm 1: Pseudocode for Our Proposed Algorithm

input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{A}, \mathcal{X})$;
Number of training epochs K ;
Corruption function C ;

output: Optimized GIN encoder g , Optimized RF h_R

```

1 Initialize the parameters  $\theta$  and  $\omega$  for the encoder  $g$  and the discriminator  $D$ ;
2 for  $epoch \leftarrow 1$  to  $K$  do
3    $h_i = g(G, \theta)$ 
4    $\tilde{h}_i = g(C(G), \theta)$ 
5    $\bar{s} = \sigma\left(\frac{1}{n} \sum_{i=1}^n h_i^{(L)}\right)$ 
6    $\mathcal{D}(h_i, \bar{s}) = \sigma\left(h_i^T \mathbf{w} \bar{s}\right)$ 
7    $\mathcal{D}(\tilde{h}_i, \bar{s}) = \sigma\left(\tilde{h}_i^T \mathbf{w} \bar{s}\right)$ 
8    $\mathcal{L}_{DGI} = -\frac{1}{2n} \sum_{i=1}^n \left( \mathbb{E}_G \log \mathcal{D}\left(h_i^{(L)}, \bar{s}\right) + \mathbb{E}_{\tilde{G}} \log \left(1 - \mathcal{D}\left(\tilde{h}_i^{(L)}, \bar{s}\right)\right) \right)$ 
9    $\theta, \omega \leftarrow \text{Adam}(\mathcal{L}_{DGI})$ 
10  $h_i = g(G, \theta)$ 
11  $h\_R \leftarrow \text{RF}((h_i \wedge (AF \vee LF)), y)$ 
12 return  $h\_R, g$ 

```

Algorithm 2: Batch Normalizing Transform [30]

input : Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$
Parameters can be learned: γ, β

output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

```

1  $\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$ 
2  $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$ 
3  $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ 
4  $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$ 

```

for training and testing the RF classifier as mentioned above.

5.0.1. Implementation Environments

This paper was carried out using a 2.3GHz 2-core Intel(R) Xeon(R) processor with 12 GB memory and Tesla P100 GPU on Linux operating system. The proposed approach is developed using the Python programming language with several statistical and visualization packages such as Sckit-learn, Numpy, Pandas, PyTorch Geometric, and Matplotlib. Table 1 summarizes the system configuration.

- Node Embeddings with AF Features:** Similar to scenario 1, we also combine all raw features (AF features) with node embeddings generated by the trained encoder

6. Performance Evaluation

For evaluating the performance of the proposed methods, the standard metrics listed in Table 2 are used, where TP , TN , FP and FN represent the number of True Positives, True Negatives, False Positives and False Negatives, respectively.

The proposed method is evaluated by using Precision, Recall, F1-score and Area under the receiver operating characteristics (ROC) curve. All the above metrics can be obtained using the confusion matrix (CM).

In Table 2, True positive (TP) means the total number of true positives, True negative (TN) indicates the total number of false positives, False positive (FP) is the total number of false negatives and False negative (FN) is the total number of true negatives. Based on the aforementioned terms, the evaluation metrics are calculated as follows. For a balanced test dataset, higher accuracy indicates the model is well learned, but for imbalance test dataset scenarios, in this case, only considering accuracy measures may lead to misleading conclusions since it is strongly biased to favour the licit class. Thus for this case, recall and F1-score metrics give a more reasonable explanation of the model’s performance.

Recall (also known as Detection Rate) is the total number of true positives divided by the total number of true positives and false negatives. If the recall rate is very low, that means the classifier cannot detect illicit transactions.

Precision measures the quality of the correct predictions. It is the number of true positives divided by the number of true positives and false positives. If the false positive is very high, it will cause low precision. Our goal is to maximize the precision as much as possible.

F1-score is the trade-off between precision and recall. Mathematically, it is the harmonic mean of precision and recall.

The area under the curve (AUC) computes the trade-off between sensitivity and specificity, plotted based on the trade-off between the true positive rate on the y-axis and the false positive rate on the x-axis. Our goal is to maximize the AUC score as much as possible.

Table 1: Implementation environment specification

Unit	Description
Processor	2.3 GHz 2-core Inter Xeon(R) Processor
RAM	12GB
GPU	Tesla P100 GPU 16GB
Operating System	Linux
Packages	Sckit-learn, Numpy, Pandas, PyTorch Geometric, and Matplotlib

Table 2: Evaluation metrics used in this study

Metric	Definition
Detection Rate (Recall)	$\frac{TP}{TP+FN}$
Precision	$\frac{TP}{TP+FP}$
F1-Score	$2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$
AUC-Score	$\int_0^1 \frac{TP}{TP+FN} d \frac{FP}{TN+FP}$

Table 3: Results of binary classification by Inspection-L compared to state-of-the-art. AF refers to all raw features, LF refers to the local raw features, i.e. the first 94 raw features, GNE refers to the node embeddings generated by GCN in [5] by using labels and DNE refers to the node embeddings computed by DGI without using labels.

Methods	Illicit			
	Precision	Recall	F1	AUC
Logistic Regr ^{AF} [5]	0.404	0.593	0.481	–
Logistic Regr ^{AF + GNE} [5]	0.537	0.528	0.533	–
Logistic Regr ^{LF} [5]	0.348	0.668	0.457	–
Logistic Regr ^{LF + GNE} [5]	0.518	0.571	0.543	–
RandomForest ^{AF} [5]	0.956	0.670	0.788	–
RandomForest ^{AF + GNE} [5]	0.971	0.675	0.796	–
RandomForest ^{AF} [15]	0.897	0.721	0.800	–
RandomForest ^{AF + GNE} [15]	0.958	0.715	0.819	–
XGB ^{AF} [15]	0.921	0.732	0.815	–
XGB ^{AF + GNE} [15]	0.986	0.692	0.813	–
RandomForest ^{LF} [5]	0.803	0.611	0.694	–
RandomForest ^{LF + GNE} [5]	0.878	0.668	0.759	–
MLP ^{AF} [5]	0.694	0.617	0.653	–
MLP ^{AF + GNE} [5]	0.780	0.617	0.689	–
MLP ^{LF} [5]	0.637	0.662	0.649	–
MLP ^{LF + GNE} [5]	0.681	0.578	0.625	–
GCN [5]	0.812	0.512	0.628	–
GCN [18]	0.899	0.678	0.773	–
Skip-GCN [5]	0.812	0.623	0.705	–
EvolveGCN [5]	0.850	0.624	0.720	–
Inspection-L^{AF + DNE} (RF)	0.972	0.721	0.828	0.916
Inspection-L^{LF + DNE} (RF)	0.906	0.712	0.797	0.895
Inspection-L^{DNE} (RF)	0.593	0.032	0.061	0.735

7. Experimental Results

Table 3 shows the corresponding results of our Inspection-L compared to state-of-the-art in terms of key metrics. As can be observed from the table, in regards to illicit F1-Score, Inspection-L outperforms the best-reported classifiers. We concatenated the node embeddings generated from DGI with the original raw features (AF) in this setting. The experiments achieved an F1 score and Recall of 0.828 and 0.721, respectively. Using all AF with node embeddings as features, the ML model’s performance has significantly increased with an AUC of 0.916, compared to 0.735 when only using node embeddings. The experiments demonstrate that graph information (node embeddings) was useful to enhance the transaction representation.

In the second experiment, we concatenated the node embeddings generated from DGI with the local features (LF), which can achieve an F1-score and Recall of 0.712 and 0.797, respectively. Both of the results are superior to the state-of-the-art algorithms.

These results demonstrate that the power of our self-supervised GNN-based can generate an enhanced feature set to improve the anti-money laundering detection performance. The results show that the accuracy of the model improves with the enhanced feature set for full features. The RF classifier is trained using all raw features and embedding features to improve overall performance. It can not only detect illicit transactions, but can also achieve a low false alarm rate.

Figure 5 shows the confusion matrix of the three different scenarios. As we can see, the RF classifier using only embedding features classifies a large number of illicit transactions into

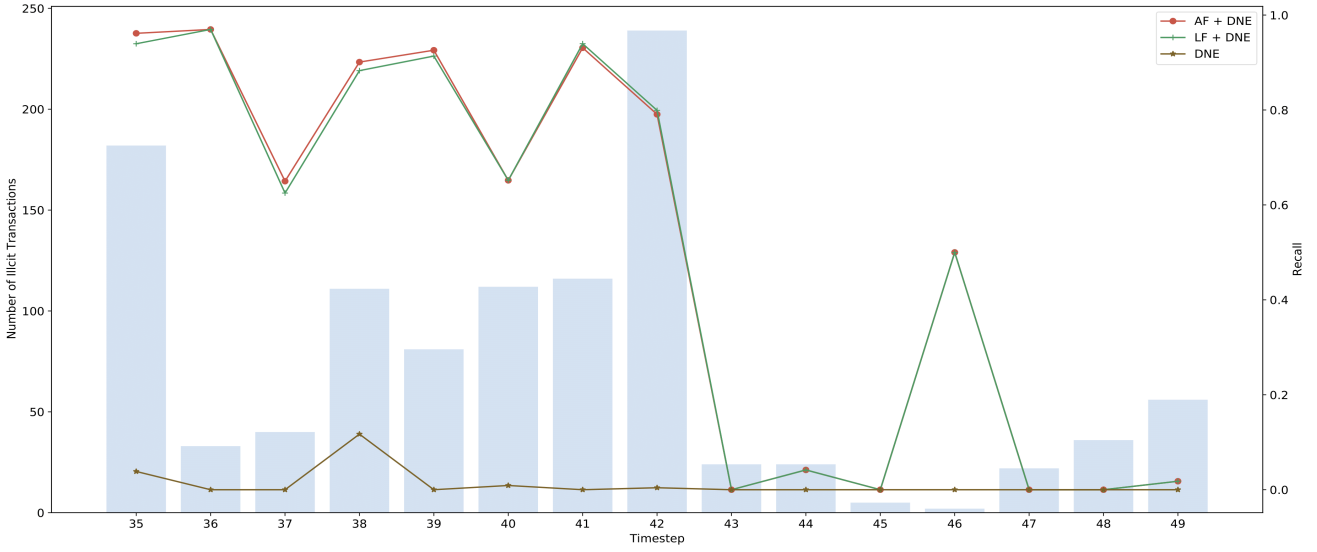


Figure 4: Illicit Recall over test timestep

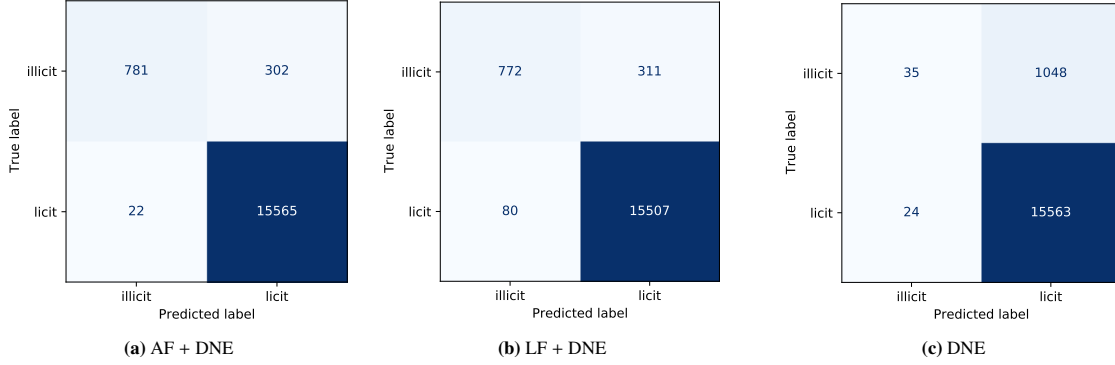


Figure 5: Confusion Matrix

licit transactions, which also have low false alarm rates. Although the classifier trained with embedding features cannot accurately detect illicit transactions, it rarely classifies licit transactions as illicit. Therefore, the false alarm rate is very low. The RF classifier trained using both raw features and embedding features has the advantage of achieving a high detection rate and a low false alarm rate.

Figure 4 shows the recall measure of three different models across various testing timesteps. Interestingly, all three models cannot detect new illicit transactions after dark market shutdown, and it was occurring at time step 43 [5]. Therefore, developing robust methods to detect illicit transactions without being affected by emerging events has become a major challenge to address.

8. Conclusions and Future Work

This paper presents a novel approach for Bitcoin illicit transaction detection based on self-supervised GNNs. We first used the DGI for generating the node embedding with raw features to train the Random Forest for detection. Our experimental evaluation indicates that our approach performs exceptionally well

and overall outperforms the state-of-the-art ML-based/Graph-based classifier. The evaluation results of our initial classifier demonstrate the potential of a self-supervised GNN-based approach for illegal transaction detection in cryptocurrencies. We hope to inspire others to work on the important challenge of using graph machine learning to perform financial forensics through this research, which is very lacking in current research. In the future, we plan to integrate with unsupervised anomaly detection algorithms to detect illegal transactions in an unsupervised manner.

References

- [1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Tech. rep., Manubot (2019).
- [2] N. Kshetri, J. Voas, Do crypto-currencies fuel ransomware?, IT professional 19 (5) (2017) 11–15.
- [3] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, R. D. Hjelm, Deep graph infomax, in: International Conference on Learning Representations, 2019. URL <https://openreview.net/forum?id=rklz9iAcKQ>
- [4] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, IEEE Transactions on Neural Networks and Learning Systems 32 (1) (2021) 4–24.

- [5] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, C. E. Leiserson, Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics, in: ACM SIGKDD International Workshop on Knowledge discovery and data mining, 2019.
- [6] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, J. Tang, Self-supervised learning: Generative or contrastive, *IEEE Transactions on Knowledge and Data Engineering*.
- [7] Y. Liu, S. Pan, M. Jin, C. Zhou, F. Xia, P. S. Yu, Graph self-supervised learning: A survey, *arXiv preprint arXiv:2103.00111*.
- [8] C. M. Bishop, N. M. Nasrabadi, *Pattern recognition and machine learning*, Vol. 4, Springer, 2006.
- [9] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, *arXiv preprint arXiv:1609.02907*.
- [10] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, C. Leiserson, Evolvegc: Evolving graph convolutional networks for dynamic graphs, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, 2020, pp. 5363–5370.
- [11] Y. Hu, S. Seneviratne, K. Thilakarathna, K. Fukuda, A. Seneviratne, Characterizing and detecting money laundering activities on the bitcoin network, *arXiv preprint arXiv:1912.12060*.
- [12] Wallet explorer.
URL <https://www.walletexplorer.com>
- [13] J. A. Bondy, U. S. R. Murty, et al., *Graph theory with applications*, Vol. 290, Macmillan London, 1976.
- [14] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.
- [15] D. Vassallo, V. Vella, J. Ellul, Application of gradient boosting algorithms for anti-money laundering in cryptocurrencies, *SN Computer Science* 2 (3) (2021) 1–15.
- [16] C. Lee, S. Maharjan, K. Ko, J. W.-K. Hong, Toward detecting illegal transactions on bitcoin using machine-learning methods, in: *International Conference on Blockchain and Trustworthy Systems*, Springer, 2019, pp. 520–533.
- [17] Blockchain explorer.
URL <https://www.blockchain.com/ko/explorer>
- [18] I. Alarab, S. Prakoonwit, M. I. Nacer, Competence of graph convolutional networks for anti-money laundering in bitcoin blockchain, in: *Proceedings of the 2020 5th International Conference on Machine Learning Technologies*, 2020, pp. 23–27.
- [19] L. Nan, D. Tao, Bitcoin mixing detection using deep autoencoder, in: *2018 IEEE Third international conference on data science in cyberspace (DSC)*, IEEE, 2018, pp. 280–287.
- [20] J. Lorenz, M. I. Silva, D. Aparício, J. T. Ascensão, P. Bizarro, Machine learning methods to detect money laundering in the bitcoin blockchain in the presence of label scarcity, in: *Proceedings of the First ACM International Conference on AI in Finance*, 2020, pp. 1–8.
- [21] T. Pham, S. Lee, Anomaly detection in bitcoin network using unsupervised learning methods, *arXiv preprint arXiv:1611.03941*.
- [22] P. Monamo, V. Marivate, B. Twala, Unsupervised learning for robust bitcoin fraud detection, in: *2016 Information Security for South Africa (ISSA)*, IEEE, 2016, pp. 129–134.
- [23] S. Li, F. Xu, R. Wang, S. Zhong, Self-supervised incremental deep graph learning for ethereum phishing scam detection, *arXiv preprint arXiv:2106.10176*.
- [24] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, K. M. Borgwardt, Weisfeiler-lehman graph kernels., *Journal of Machine Learning Research* 12 (9).
- [25] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, R. D. Hjelm, Deep graph infomax, *International Conference on Learning Representations (ICLR)*.
- [26] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, in: *Advances in Neural Information Processing Systems*, 2017. *arXiv:1706.02216*.
- [27] C. Zhang, D. Song, C. Huang, A. Swami, N. V. Chawla, Heterogeneous graph neural network, in: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 793–803.
- [28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, *International Conference on Learning Representations (ICLR)*.
- [29] S. Yun, M. Jeong, R. Kim, J. Kang, H. J. Kim, Graph transformer networks, *Advances in neural information processing systems* 32.
- [30] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: *International conference on machine learning*, PMLR, 2015, pp. 448–456.
- [31] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, *arXiv preprint arXiv:1810.00826*.