


Article

Disentangled Prototypical Graph Convolutional Network for Phishing Scam Detection in Cryptocurrency Transactions

Seok-Jun Buu ¹  and Hae-Jung Kim ^{2,*}

¹ Department of Computer Science, Gyeongsang National University, Jinju-si 52828, Republic of Korea; sj.buu@gnu.ac.kr

² Department of Computer Science, Kyungil University, Gyeongsan-si 38428, Republic of Korea

* Correspondence: hjkim325@kiu.kr

Abstract: Blockchain technology has generated an influx of transaction data and complex interactions, posing significant challenges for traditional machine learning methods, which struggle to capture high-dimensional patterns in transaction networks. In this paper, we present the disentangled prototypical graph convolutional network (DP-GCN), an innovative approach to account classification in Ethereum transaction records. Our method employs a unique disentanglement mechanism that isolates relevant features, enhancing pattern recognition within the network. Additionally, we apply prototyping to disentangled representations, to classify scam nodes robustly, despite extreme class imbalances. We further employ a joint learning strategy, combining triplet loss and prototypical loss with a gamma coefficient, achieving an effective balance between the two. Experiments on real Ethereum data showcase the success of our approach, as the DP-GCN attained an F1 score improvement of 32.54%p over the previous best-performing GCN model and an area under the ROC curve (AUC) improvement of 4.28%p by incorporating our novel disentangled prototyping concept. Our research highlights the importance of advanced techniques in detecting malicious activities within large-scale real-world cryptocurrency transactions.



Citation: Buu, S.-J.; Kim, H.-J.

Disentangled Prototypical Graph Convolutional Network for Phishing Scam Detection in Cryptocurrency Transactions. *Electronics* **2023**, *12*, 4390. <https://doi.org/10.3390/electronics12214390>

Academic Editors: Mikolaj Karpinski, Oleksandr O. Kuznetsov and Roman Oliynykov

Received: 5 October 2023

Revised: 19 October 2023

Accepted: 21 October 2023

Published: 24 October 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: scam detection; node classification; graph neural network; representation learning; blockchain; cryptocurrency transaction network

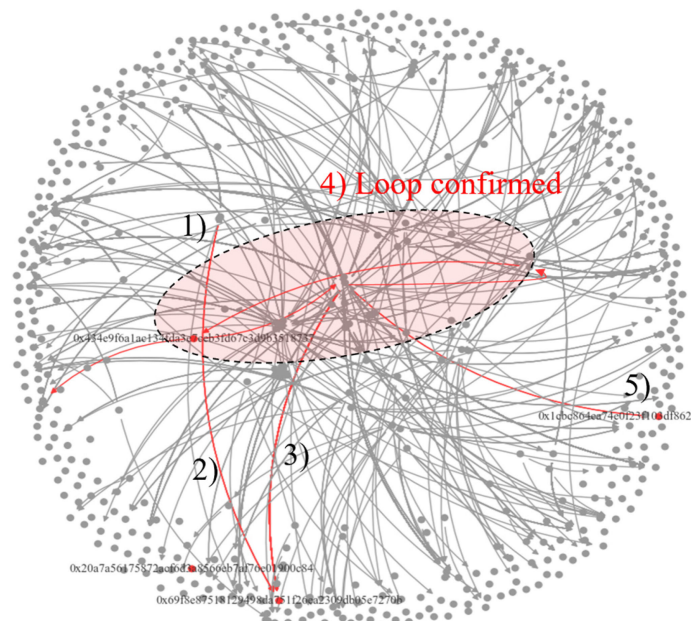
1. Introduction

Blockchain technology has revolutionized the digital transaction landscape, offering decentralized ledger systems that record transactions across multiple computers, to ensure data integrity and transparency. Initially conceptualized for cryptocurrency transactions, blockchain technology has seen rapid adoption across various industries, resulting in an explosion of transaction data and complex interactions [1]. Challenges of efficiently processing and analyzing the massive volume of data generated within blockchain networks accompany this growth.

One such blockchain network is Ethereum, a platform allowing the creation of customizable, self-executing contracts (smart contracts) and decentralized applications (DApps). As Ethereum gains traction, the vast and intricate web of transactions between accounts (nodes) becomes increasingly complex [2,3]. Ethereum accounts, either externally owned or controlled by smart contract code, interact through transactions, forming a dynamic and complex network structure that constantly evolves with the execution of the smart contracts.

The Ethereum transaction network poses unique challenges due to its sheer size and intricate interactions. The network generates massive amounts of data from high transaction volumes [4], with its ledger containing hundreds of millions of transactions. Extreme class imbalance in the network makes it difficult to accurately identify malicious activities, as certain classes, such as scam nodes, are underrepresented [4]. Traditional machine learning techniques struggle to capture high-dimensional topological and transactional patterns [5].

A concrete illustration of this challenge is shown in Figure 1. This figure visualizes the Ethereum transaction graph, highlighting a suspicious transaction pattern. The transaction sequence marked in red depicts a possible case of wash trading, in which an NFT is minted and this is followed by a series of sales where the final sale that may be artificially inflated, including a loop indicating potential wash trading. This example underscores the need for advanced techniques capable of identifying subtle malicious activities within the complex transaction network of Ethereum.



- 1) NFT Minted → 2) First sell → 3) Second sell
 → 4) Third sell: **Loop indicates potential wash trading**
 → 5) Fourth sell: Potentially inflated

Figure 1. Ethereum transaction graph showcasing a suspicious transaction pattern, highlighted in red, indicative of potential wash trading and an artificially inflated sale.

Understanding and addressing these challenges is crucial for ensuring Ethereum's security and integrity and for promoting trust and confidence in blockchain technology adoption. In the context of extreme class imbalance and massive, noisy real-world graph structures, there is a pressing need for advanced methods to model the Ethereum transaction network, disentangle transaction features, and tackle account classification and fraud detection challenges.

In this research, we introduce the disentangled prototypical graph convolutional network (DP-GCN) for account classification in Ethereum transaction records. Our method employs a novel approach, which we term disentangled prototyping, to enhance pattern recognition in Ethereum transaction networks. The contributions of our work can be summarized as follows:

- **Disentangled Prototyping:** We propose a new approach that uses disentangled prototyping to effectively recognize patterns in Ethereum transaction networks by isolating relevant features and leveraging prototypical networks, to enhance account classification.
- **Joint Learning:** We incorporate a joint learning strategy that combines curriculum learning, triplet loss, and prototypical loss with an adjustable gamma coefficient for optimal performance.
- **Empirical Success:** Our method achieved significant improvements over existing methods in experiments on real Ethereum transaction data, demonstrating its practical effectiveness in analyzing large-scale cryptocurrency transactions.

2. Related Works

The analysis of blockchain transaction graphs has received significant attention in the literature, with various methods employed to achieve different objectives. Table 1 provides an overview of the relevant literature, detailing the approach, objective, method, transaction network, and performance.

Earlier research explored graph traversing techniques for analyzing blockchain transaction data. For instance, Lin et al. utilized a graph representation method that captures time-dependent patterns in Ethereum transaction networks by encoding temporal information into walk strategies [3]. This approach provides a robust representation of time-evolving blockchain transaction graphs [6]. Similarly, Ofori-Boateng et al. employed a topological analysis technique to detect anomalous transaction patterns in Ethereum and Ripple networks [5]. Their method highlights the importance of topological features in blockchain anomaly detection. Further advancing the field, Bai et al. adopted temporal graphs to capture dynamics in the Ethereum transaction network [6]. By incorporating temporal information, their approach provides a comprehensive analysis of blockchain networks.

The use of graph embedding techniques has become increasingly prevalent in recent research. Jin et al. combined a graph convolutional network (GCN) with hierarchical feature augmentation (HFAug) to detect Ponzi schemes in blockchain networks [7]. Their approach underscores the effectiveness of augmenting node features for improved performance in graph-based anomaly detection. Expanding upon this idea, Liu et al. proposed a feature augmentation-based graph neural network (FA-GNN) for account classification in Ethereum transaction data [8]. Their method emphasizes the significance of feature augmentation in achieving accurate blockchain account classification.

Meanwhile, Xia et al. introduced an ego-graph embedding approach coupled with a skip-gram model for phishing detection [9]. Their method demonstrated enhanced pattern recognition capabilities in blockchain transaction graphs. Building on the concept of graph embedding, Huang et al. developed an edge heterogeneous graph convolutional network (EH-GCN) for account classification in Ethereum transaction data [2]. Their approach highlights the importance of modeling edge heterogeneity for accurate node classification in blockchain networks. Lastly, Zhou et al. designed Ethident for de-anonymization across multiple Ethereum transaction datasets [10]. Their method showcased the benefits of a unified approach to de-anonymization across various types of transactions.

In this study, we propose a unique disentanglement and prototyping process, employing a custom loss function that enables the separation of transaction features into interpretable and meaningful representations. By isolating distinct transactional behaviors and interactions within the network, our method allows for the creation of accurate and informative prototypes.

Table 1. Summary of approaches and methods for transaction graph modeling in cryptocurrency networks.

Approach	Objective	Method	Transaction Network	Performance
Traversing	Graph representation	Temporal Walk Strategies [3]	Ethereum	AUC 0.9383
	Anomaly Detection	Clique Persistent Homology [5]	Ethereum, Ripple	Acc. 0.9540
Traversing Embedding	Ponzi Scheme Detection	GCN with HFAug [7]	Ethereum	Success rate 0.8405
	Account Classification	FA-GNN [8]	Ethereum	F1-Score 0.8880
Embedding	Phishing Detection	Ego-Graph Embedding, Skip-gram Model [9]	Ethereum	F1-Score 0.8199
	Account Classification	EH-GCN [2]	Ethereum	Acc. 0.8620
	De-Anonymization	Ethident [10]	ETH-Mining, ETH-Exchange, ETH-Phish	F1-Score 0.9798

Unlike the existing approaches, our method combines the power of unsupervised learning with a carefully designed loss function, which allows the model to capture non-trivial,

high-level patterns in the transaction graphs. This process enhances the interpretability of the embeddings, making it easier to discern relationships and anomalies within the data.

3. Proposed Method

3.1. Overview of the Proposed Method

Our proposed method, the disentangled prototypical graph convolutional network (DPGCN), is a novel approach to detecting fraudulent transactions in cryptocurrency transaction networks. It integrates disentangled representation learning and prototypical networks within a graph convolutional framework, aiming to provide a robust and interpretable mechanism for classifying nodes as either scam or benign. Figure 2 illustrates the structure of the proposed DPGCN, showcasing the flow of information from the transaction network to the disentangled prototypes.

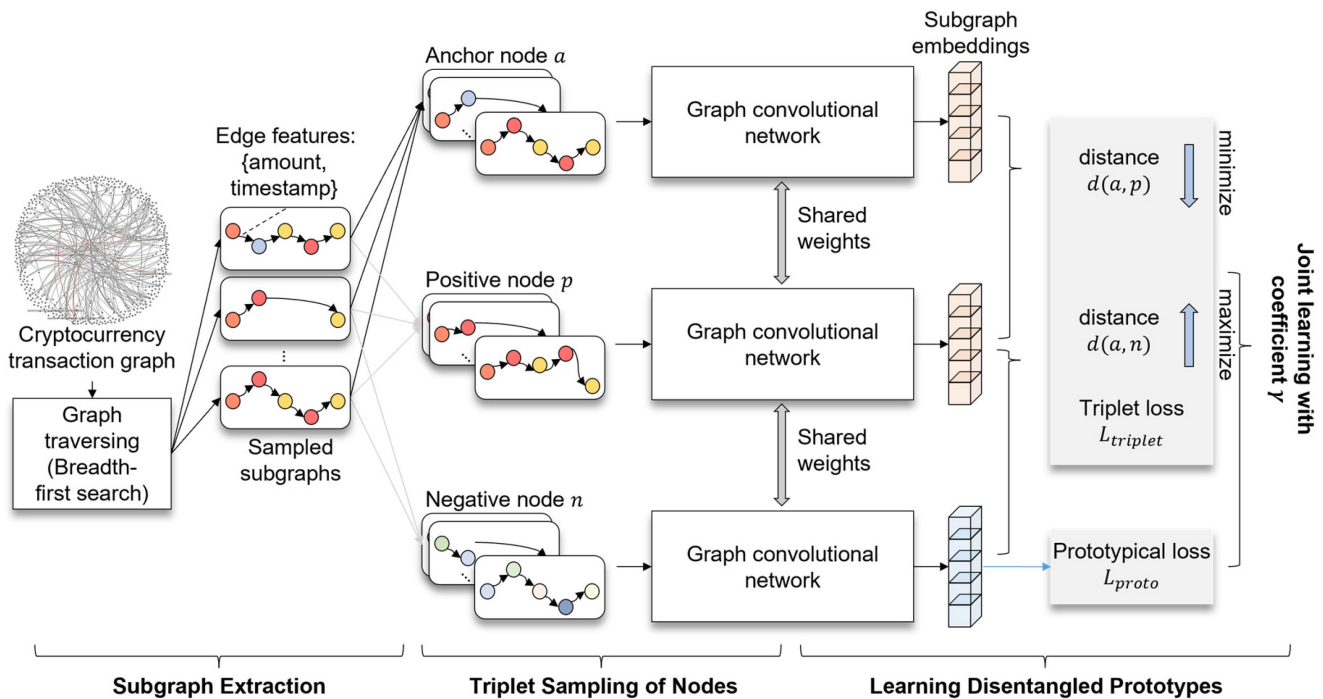


Figure 2. Overview of the disentangled prototypical graph convolutional network (DPGCN) architecture.

The DPGCN model operates in three main stages: subgraph extraction, triplet sampling of nodes, and learning disentangled prototypes. We utilize a graph convolutional network (GCN) to learn node embeddings [11,12], which captures both transactional and topological features from the Ethereum transaction network. Given a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, the GCN layer computes the embeddings for each node using the following equation:

$$H^{(l+1)} = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right) \quad (1)$$

where $H^{(l)}$ is the node feature matrix at layer l , \tilde{A} is the adjacency matrix with added self connections, \tilde{D} is the degree matrix of \tilde{A} , $W^{(l)}$ is the weight matrix at layer l , and σ is the activation function. This formula allows the DPGCN model to capture complex patterns within the transaction graph by aggregating information from neighboring nodes and transforming the node features through multiple GCN layers.

3.2. Disentangled Prototyping of a Transaction Network

The disentanglement mechanism within our disentangled prototypical graph convolutional network (DPGCN) is achieved by employing a triplet loss function. Given an anchor node a , a positive node p sharing similar characteristics with a , and a negative node n that is dissimilar to a , the triplet loss function aims to ensure that the embeddings of similar (same class) nodes are closer in the latent space than the embeddings of dissimilar nodes, by at least a margin of α :

$$\mathcal{L}_{\text{triplet}} = \max\left(0, \|e_a - e_p\|_2^2 - \|e_a - e_n\|_2^2 + \alpha\right) \quad (2)$$

where e_a , e_p , and e_n are the embeddings of the anchor, positive, and negative samples, respectively, and α is the margin parameter.

Once the disentangled embeddings are learned, our next step is to prototype these embeddings for the task of scam node detection. Prototyping involves learning a representative embedding for each class (i.e., scam or benign), which serves as a “prototype” in the embedding space. The prototype P_c of class c is computed as the mean of the embeddings of the samples in that class:

$$P_c = \frac{1}{N_c} \sum_{i=1}^{N_c} e_i \quad (3)$$

where e_i is the embedding of the i th sample in class c and N_c is the number of samples in class c .

To classify a node, we measure the distance between its embedding and the prototypes of all classes, assigning it to the class with the closest prototype:

$$y_i = \arg \min_c \|e_i - P_c\|_2^2 \quad (4)$$

where y_i is the predicted class of node i .

During training, we minimize the prototypical loss:

$$\mathcal{L}_{\text{proto}} = \frac{1}{N} \sum_{i=1}^N -\log \left(\frac{\exp(-\|e_i - P_{y_i}\|_2^2)}{\sum_{c=1}^K \exp(-\|e_i - P_c\|_2^2)} \right) \quad (5)$$

where $\mathcal{L}_{\text{proto}}$ is the prototypical loss, N is the total number of samples, and K is the number of classes.

To put the disentanglement and prototyping process into practice, we employed Algorithm 1, which trained our DPGCN on the Ethereum transaction graph (ETG) with disentangled embeddings. This algorithm utilized curriculum weighting to gradually shift the emphasis between the triplet loss and the prototypical loss over the training epochs.

3.3. Joint Learning with Gamma Coefficient and Curriculum Learning

The overall loss of our model L is a linear combination of the prototypical loss $\mathcal{L}_{\text{proto}}$ and the triplet loss $\mathcal{L}_{\text{triplet}}$. The coefficient γ is introduced to modulate the weight between the two losses, where a higher γ emphasizes prototypical loss and a lower γ emphasizes triplet loss:

$$\mathcal{L} = \gamma \mathcal{L}_{\text{proto}} + (1 - \gamma) \mathcal{L}_{\text{triplet}} \quad (6)$$

To smoothly transition the emphasis between the two losses, we adopted a sigmoidal curriculum weighting scheme to adjust the weights at different training epochs:

$$w(t) = \frac{1}{1 + \exp(-\kappa(t - \tau))} \quad (7)$$

where $w(t)$ is the curriculum weight at epoch t , κ is the sigmoid scaling factor, and τ is the sigmoid shift parameter.

Algorithm 1: Disentangled Prototypical Graph Convolutional Network (DPGCN) Training

Training a DPGCN on the Ethereum transaction graph with disentangled embeddings, utilizing curriculum weighting to balance emphasis between triplet and prototypical losses during the training epochs.

Input:

- ETG: Ethereum Transaction Graph
- transaction_features: Features for each transaction in ETG
- num_classes: Number of different classes (or types) of transactions
- num_epochs: Number of training epochs
- α : Margin for triplet loss
- γ_{init} : Initial value for dynamic gamma adjustment
- γ_{final} : Final value for dynamic gamma adjustment
- τ : Epoch threshold for aggressive γ adjustment

Output:

- dpgcn_model: Trained DPGCN model for disentangled embeddings

Initialization

- 1: function train_DPGCN(ETG, transaction_features, num_classes, num_epochs, α , γ_{init} , γ_{final} , τ)
- 2: Initialize DPGCN model parameters with disentangling mechanisms specific to transaction characteristics.
- 3: Initialize class_prototypes as zero vectors of embedding dimension.

Main Training Loop

- 4: for epoch = 1 to num_epochs do

Dynamic Gamma Adjustment

- 5: if epoch < τ then
- 6: $\gamma = \gamma_{init} + (\text{epoch}/\tau) * (0 - \gamma_{init})$
- 7: else
- 8: $\gamma = 0 + ((\text{epoch} - \tau)/(\text{num_epochs} - \tau)) * (\gamma_{final} - 0)$
- 9: end if

Disentangled Prototyping

- 10: embeddings = []
- 11: triplets = sample_triplets(ETG, transaction_features, num_classes)
- 12: for triplet in triplets do
- 13: anchor, positive, negative = triplet
- 14: // Compute disentangled embeddings for each node in triplet and extend embeddings list
- 15: embeddings.extend([DPGCN(node, transaction_features) for node in [anchor, positive, negative]])
- 16: end for
- 17: // Compute prototypes for each class by averaging embeddings
- 18: for c = 1 to num_classes do
- 19: class_members = get_transactions_of_class(ETG, c)
- 20: class_prototype[c] = average(embeddings[class_members])
- 21: end for

Loss Computation and Model Update

- 22: // Compute triplet loss
 - 23: triplet_loss = compute_triplet_loss(embeddings, α)
 - 24: // Compute prototypical loss
 - 25: prototype_loss = compute_prototypical_loss(embeddings, class_prototypes)
 - 26: // Total DPGCN loss
 - 27: loss = $((1 - \gamma) * \text{triplet_loss} + (1 + \gamma) * \text{prototype_loss})$
 - 28: Update DPGCN model parameters using backpropagation with the computed loss.
 - 29: end for
 - 30: return dpgcn_model
-

The sigmoidally weighted triplet loss $L_{triplet}$ and prototypical loss L_{proto} are defined as below, respectively:

$$\mathcal{L}_{triplet}^w = w(t)\mathcal{L}_{triplet} \quad (8)$$

$$\mathcal{L}_{\text{proto}}^w = (1 - w(t))\mathcal{L}_{\text{proto}} \quad (9)$$

The curriculum weight $w(t)$ dynamically determines the contribution of each loss to the overall training objective. The gamma coefficient γ serves as a hyperparameter that regulates the balance between the two loss components, where a higher value γ corresponds to greater emphasis on prototypical loss, and vice versa.

Algorithm 2 illustrates the process of detecting potentially fraudulent transactions in an Ethereum transaction graph (ETG) using disentangled embeddings from the trained DPGCN model. By comparing transaction embeddings to class prototypes, the algorithm classifies each transaction as a scam or benign.

Algorithm 2: Transaction Scam Detection using Disentangled Prototypical Graph Convolutional Network

Algorithm 2 outlines the process of detecting potentially fraudulent transactions in the Ethereum transaction graph using disentangled embeddings from DPGCN. By comparing transaction embeddings to class prototypes, the algorithm classifies each transaction as scam or benign.

Input:

- ETG: Ethereum Transaction Graph
- transaction_features: Features for each transaction in ETG
- dpgcn_model: Pre-trained Disentangled Prototypical Graph Convolutional Network model

Output:

```

- scam_labels: Predicted scam/benign labels for transactions in ETG
1: function ScamDetection(ETG, transaction_features, dpgcn_model)
2:   //Obtain disentangled embeddings for all transactions
3:   embeddings = dpgcn_model.get_embeddings(ETG, transaction_features)
4:   //Calculate class prototypes
5:   class_prototypes = dpgcn_model.compute_class_prototypes(embeddings)
6:   scam_labels = []
7:   for each transaction in ETG do
8:     //Derive its disentangled embedding
9:     transaction_embedding = embeddings[transaction]
10:    //Determine its class by the nearest prototype
11:    nearest_class = find_nearest_prototype(transaction_embedding, class_prototypes)
12:    if nearest_class is scam:
13:      scam_labels.append('scam')
14:    else:
15:      scam_labels.append('benign')
16:    end if
17:  end for
18:  return scam_labels
19: end function

```

4. Experimental Results

4.1. Dataset and Preprocessing

We conducted our experiments on the Ethereum transaction history, focusing on a large connected component of the transaction graph. The dataset utilized in this study was obtained from the research conducted by Chen et al. [13], which offered an in-depth description of the data collection and preprocessing techniques. This connected component was extracted through initiating random walks from 1165 source nodes, resulting in a subgraph that encompassed 2,973,382 nodes and 13,551,214 edges. Among these nodes, 1157 were labeled as phishing nodes.

In the context of this study, when we refer to the graph size, we are specifically alluding to the number of nodes in the graph. While both the node and edge counts provide vital insights into the structure and characteristics of a graph, we chose the number of nodes to represent the graph's size for clarity and simplicity.

The average degree of a node provides another essential metric. This signifies the average number of edges connected to a node in a graph. For a directed graph with $|V|$ nodes and $|E|$ edges, the average degree D can be mathematically expressed as $D = \frac{|E|}{|V|}$. Within our Ethereum transaction dataset, this metric offers insight into the average transactional connectivity of an entity in the network.

To ensure a comprehensive and consistent evaluation of our models, we selected the largest connected component of the transaction graph. We deliberately excluded other connected components, to mitigate potential biases that could arise from analyzing smaller, disconnected transaction patterns. The expansive nature of this component allows it to aptly capture the intricate relationships embedded within the Ethereum transaction network.

To establish experimental samples, we embarked on random walks, starting each from a unique node, with the aim of generating subgraphs of varying sizes. For this research, we zeroed in on subgraphs encompassing 30,000, 40,000, and 50,000 nodes, aligning with the comparative methods adopted. Such a methodological approach facilitated an exploration into the performance of our proposed model against a backdrop of diverse graph sizes, each reflecting differing complexity levels of the Ethereum transaction history. Table 2 offers a summarized view of our dataset specifications and the subgraph sizes we engaged in our experiments.

Table 2. Specifications of the Ethereum transaction dataset used in the experiments, detailing the dataset collection period and graph sizes.

# Nodes (Scam)	# Edges	Average Degree
Ethereum transaction history (7 August 2015–19 January 2019)		
2,973,382 (1165)	13,551,214	9.1147
Subgraph [13]		
30,000 (113)	774,379	51.6252
40,000 (134)	994,410	49.7205
50,000 (172)	1,388,156	55.5262

4.2. Implementation Details and Evaluation Metrics

For the implementation of our experiments, we used the Python deep learning library PyTorch (version 2.0.1) in conjunction with the graph deep learning library Spektral (version 1.3.0), TensorFlow-gpu (version 2.9.0), and Scikit-learn (version 1.3.0) for preprocessing and evaluation purposes. We conducted our experiments on NVIDIA Tesla V100 GPUs.

For the node features, we opted to use an identity matrix, due to the lack of available node data other than class labels. Edge features, however, were constructed using the transaction amount and timestamp associated with each transaction.

As for the hyperparameters, we set the margin parameter α for the triplet loss to 0.2. The embedding vector used for calculating the prototypes was set to a dimension of 16. To ensure a robust evaluation, all experiments were conducted with 5-fold cross-validation, with 20% of the data reserved for testing in each fold. In each experiment, the number of triplets was fixed at 2000. Given the high class imbalance in our dataset, we used precision, recall, and F1 score as the primary evaluation metrics.

Considering the significant class imbalance inherent in our dataset, we anchored our evaluation around precision, recall, and the F1 score. It is pertinent to highlight that the F1 score serves as the harmonic mean of the precision and recall, acting as a balanced metric between the two, particularly in scenarios with an imbalanced class distribution. The formula for the F1 score is given by $F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$.

AUC offers insight into the classifier's ability to discern between the classes, indicating the probability that a randomly selected positive instance is ranked higher than a randomly selected negative one. This metric becomes especially invaluable when dealing with imbalanced datasets.

4.3. Performance Comparison

Table 3 presents a comparison of precision, recall, and F1 scores for different graph sizes (30,000, 40,000, and 50,000). We compared our proposed DPGCN method with Deep Walk, Node2Vec, and LINE for graph traversing and embedding, as well as GNN, GAT, and GCN. Additionally, we assessed variants of our proposed method, such as DPGCN w/o prototyping and DPGCN w/o disentanglement.

Table 3. Precision, recall, and F1 score comparison of the different methods for graphs of sizes 30,000, 40,000, and 50,000.

Model	Graph Size = 30,000			Graph Size = 40,000			Graph Size = 50,000		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Graph traversing/embedding									
Deep Walk [14]	0.1251	0.7108	0.2049	0.1453	0.5754	0.2227	0.1575	0.5945	0.2426
Node2Vec [15]	0.1094	0.6956	0.1832	0.1424	0.6689	0.2267	0.1554	0.6475	0.2426
LINE [16]	0.1409	0.5352	0.2163	0.1332	0.5597	0.2087	0.1726	0.5222	0.2538
Comparatives									
GNN	0.8447	0.5536	0.5658	0.6382	0.6267	0.6117	0.6458	0.6284	0.6079
GAT	0.8483	0.5662	0.5818	0.6629	0.6485	0.6381	0.6717	0.6392	0.6338
GCN [2]	0.8748	0.5714	0.5949	0.6419	0.6520	0.6466	0.6494	0.6449	0.6471
Ours									
DPGCN	0.9637	0.8898	0.9203	0.9666	0.8958	0.9250	0.9394	0.9410	0.9402
w/o prototyping	0.9060	0.5984	0.6384	0.9022	0.6028	0.6433	0.8825	0.6073	0.6535
w/o disentanglement	0.9168	0.6712	0.7293	0.9037	0.6753	0.7339	0.6882	0.9004	0.7801

Several state-of-the-art graph embedding and traversing methods served as benchmarks in our comparative analysis against DPGCN. Deep Walk [14] is a popular approach that generates embeddings by simulating random walks across a graph, allowing a deep representation of vertex sequences. Node2Vec [15], an extension of Deep Walk, provides a more flexible and generalized random walk, offering enhanced node homophily and structural equivalence. LINE [15], on the other hand, focuses on large-scale information network embeddings, striving to preserve both local and global network structures. On the graph neural network (GNN) front, GAT and GCN [2] stand out as prominent models. GAT introduces attention mechanisms, enabling nodes to weigh their neighbors' features, while GCN focuses on creating a layered propagation model to effectively represent graph-structured data. Our proposed DPGCN aims to advance beyond these methods by introducing disentangled prototyping, enabling refined embeddings, particularly for a Ethereum transaction graph.

From Table 3, several insights can be drawn. First, the overall performance improved as the graph size increased, with the highest performance achieved when the graph size was 50,000. This was likely due to the extremely sparse and class-imbalanced nature of the dataset. Furthermore, the graph-traversing or embedding methods such as Deep Walk, Node2Vec, and LINE exhibited relatively low performance, likely due to their inability to effectively model edge information (transaction amount and timestamp), which is crucial in this dataset. In contrast, the graph neural networks such as GNN, GAT, and GCN displayed significantly better performance, as they effectively model edge information.

The proposed DPGCN model, along with its variants (DPGCN w/o prototyping and DPGCN w/o disentanglement), consistently achieved the highest F1 scores across all graph sizes compared to the conventional GCN and other methods. Specifically, our method experimentally demonstrated the validity of the disentangled prototyping strategy for this problem, achieving a maximum F1 score of 0.9402, in comparison to the GCN method, which scored 0.6471 F1. This result highlights the effectiveness of disentangled prototyping in classifying transactions in the Ethereum transaction graph.

4.4. Effects of Disentangled Prototyping

To further investigate the effectiveness of our disentangled prototyping approach, we visualized the feature space using the t-SNE technique and compared the AUC scores for different models.

Figure 3 provides a visualization of the feature space for (a) the input space, (b) the GCN embedded space, and (c) our DPGCN model, using the t-SNE technique. The visualization highlights the differences in the feature spaces across the three cases. In the input space, the features are likely to be scattered randomly with no discernible pattern. In the GCN-embedded space, while there may be some clustering, the features are still likely to be entangled, making it challenging to distinguish between classes. In contrast, the DPGCN model produces a feature space that is clearly distinguishable and less entangled. This visualization emphasizes the advantages of our disentangled prototyping approach, which can help in creating more informative and less entangled feature spaces for better classification.

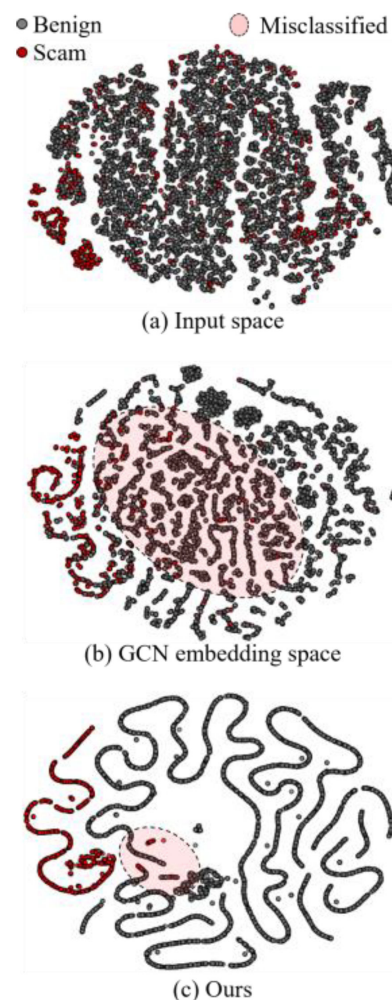


Figure 3. Feature space visualization using the t-SNE technique for (a) input space, (b) GCN[2]-embedded space, and (c) our DPGCN model.

Figure 4 compares the area under the ROC curve (AUC) for DPGCN and DPGCN w/o disentangled prototyping. A higher AUC indicates a better model, as it measures the model's ability to distinguish between positive and negative classes. As shown in the figure, DPGCN achieved a higher AUC compared to DPGCN w/o disentangled prototyping. This result further validates the effectiveness of the disentangled prototyping in our proposed model. By disentangling the feature space, our DPGCN model could more effectively differentiate between the classes, leading to an improved classification performance.

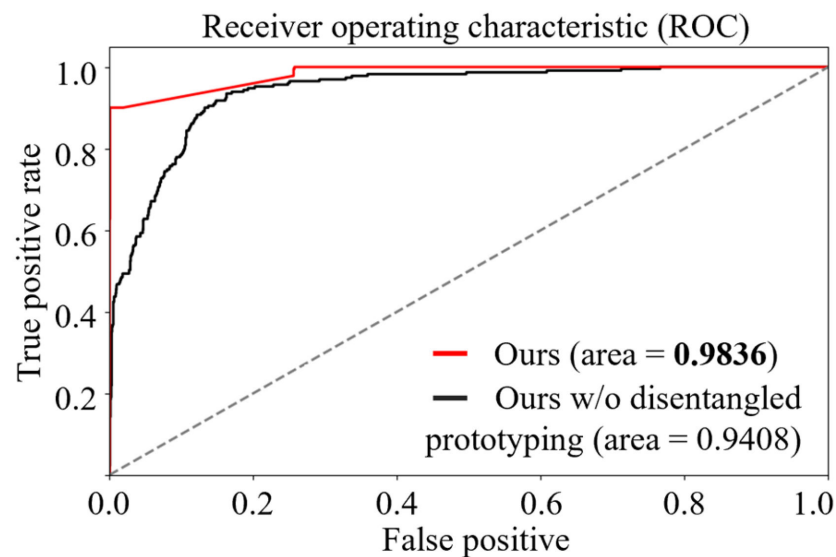


Figure 4. Comparison of area under the ROC curve (AUC) for DPGCN and w/o disentangled prototyping.

4.5. Discussion

We consider the implications of our experiments by performing case analyses on transactions with differing classifications between the GCN model and our DPGCN model, as presented in Table 4.

Table 4. Case analysis of transactions, showing differences in classification between GCN and our DPGCN method, providing node name, block number, from and to destination, and transaction value information.

	Node Name	Reported as	Block	from	To	Value (ETH)
Case 1 (Misclassified by GCN but correct in Ours)	0 ×	Phishing scam	4200269	Self	0 ×	372.999
	950bb8abd2419da2c867		4186015	Enigma presale	7965...F29A	373.000
	97a23d43bbb2da067848	Fraud	7795229	Self	Spindle Token	0.000
	0 ×		7795217	Self	Playgame Token	0.000
	8760d59d64fc8082d278		7042857	Self	Exchange	0.000
	8f1e17e844f4e47230fe				(Malaysia)	
Case 2 (Both misclassified)			5986398	Self	Other scam node	4.144
	0 ×	Phishing scam		Exchange		
	9844f5c5f9aa7146a74f		5985936	(Bittrex, Seattle, WA, USA)	Self	0.084
	fc7b9227742acfa71dea		5976675	Other scam node	Self	1.000

- Case 1: DPGCN effectively detected scams associated with presale, low-quality tokens, and transfers to small exchanges. Despite limited information (amount and timestamp), our method captured and differentiated scam and benign transactions, successfully classifying such cases. This ability is crucial for robust fraud detection.
- Case 2: A scam node transaction transferred to the large exchange Bittrex was falsely detected. This highlights the limitations of our model in the absence of destination information. Incorporating additional transfer destination data could improve classification for such cases.

In summary, our DPGCN model offers advantages over the GCN method for detecting fraudulent Ethereum transactions, demonstrated by its ability to effectively separate scam and benign transaction histories. Incorporating additional features could further enhance its performance.

In the pursuit of refining our DPGCN model and enhancing Ethereum transaction security, we must also be cognizant of broader implications. Ethical considerations come to the forefront, especially as cryptocurrency transactions bear significant legal and moral weight. The repercussions of false positives, which could unjustly tarnish legitimate entities, are as concerning as the dangers posed by false negatives, which may let malicious activities go unchecked. Moreover, while our model aims to distinguish transaction patterns, there is an inherent risk of infringing upon the privacy rights of users, even if indirectly. Ensuring network integrity is vital, but it should not come at the cost of the very principles of privacy and fairness that underpin the cryptocurrency world.

5. Conclusions

In this study, we introduced the disentangled prototypical graph convolutional network (DPGCN) for identifying fraudulent Ethereum transactions. Our approach combines the strengths of prototypical networks, disentangled representations, and graph convolutional networks for effective transaction network modeling and enhanced fraud detection. Using a real-world Ethereum dataset, we demonstrated the superiority of our model over conventional graph traversal methods and comparatives, such as GNN, GAT, and GCN. Our results highlighted the ability of our method to accurately distinguish between scam and benign transaction histories, showcasing the potential of disentangled prototypical representations.

As we look ahead, several avenues emerge to refine our model further. The potential of integrating richer destination information is evident; however, this comes with challenges, such as assessing the authenticity of these addresses. While our results advocate for the inclusion of more intricate features into the graph for improved performance, it is pivotal to weigh the benefits of adding meta-information, such as transaction frequency or associated notes. This could enhance the model's precision, albeit with the task of filtering potential data noise. Additionally, as we aspire to synergize enhanced graph features, harmonization with the disentangled prototypical loss framework necessitates careful evaluation. Moving forward, we are driven to develop more advanced graph neural networks that can handle vast and intricately complex graphs, beyond only Ethereum transaction analysis, and to unlock further insights in the realm of large-scale network data.

Author Contributions: Conceptualization, S.-J.B.; Formal analysis, H.-J.K.; Funding acquisition, H.-J.K.; Investigation, S.-J.B.; Methodology, S.-J.B. and H.-J.K.; Visualization, S.-J.B.; Writing—review and editing, H.-J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No. NRF-2021R1F1A1063085).

Data Availability Statement: Xblock datasets (<http://xblock.pro/#/search?types=datasets>).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wang, Z.; Huang, B.; Tu, S.; Zhang, K.; Xu, L. DeepTrader: A deep reinforcement learning approach for risk-return balanced portfolio management with market conditions Embedding. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 643–650. [CrossRef]
2. Huang, T.; Lin, D.; Wu, J. Ethereum account classification based on graph convolutional network. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 2528–2532. [CrossRef]
3. Lin, D.; Wu, J.; Yuan, Q.; Zheng, Z. Modeling and understanding ethereum transaction records via a complex network approach. *IEEE Trans. Circuits Syst. II: Express Briefs* **2020**, *67*, 2737–2741. [CrossRef]
4. Martin, K.; Rahouti, M.; Ayyash, M.; Alsmadi, I. Anomaly detection in blockchain using network representation and machine learning. *Secur. Priv.* **2022**, *5*, e192. [CrossRef]

5. Ofori-Boateng, D.; Dominguez, I.S.; Akcora, C.; Kantarcioglu, M.; Gel, Y.R. Topological anomaly detection in dynamic multilayer blockchain networks. In Proceedings of the Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, 13–17 September 2021; Proceedings, Part I 21; pp. 788–804.
6. Bai, Q.; Zhang, C.; Liu, N.; Chen, X.; Xu, Y.; Wang, X. Evolution of transaction pattern in Ethereum: A temporal graph perspective. *IEEE Trans. Comput. Soc. Syst.* **2021**, *9*, 851–866. [[CrossRef](#)]
7. Jin, C.; Jin, J.; Zhou, J.; Wu, J.; Xuan, Q. Heterogeneous feature augmentation for ponzi detection in ethereum. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 3919–3923. [[CrossRef](#)]
8. Liu, J.; Zheng, J.; Wu, J.; Zheng, Z. FA-GNN: Filter and augment graph neural networks for account classification in ethereum. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 2579–2588. [[CrossRef](#)]
9. Xia, Y.; Liu, J.; Wu, J. Phishing detection on ethereum via attributed ego-graph embedding. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 2538–2542. [[CrossRef](#)]
10. Zhou, J.; Hu, C.; Chi, J.; Wu, J.; Shen, M.; Xuan, Q. Behavior-aware account de-anonymization on ethereum interaction graph. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 3433–3448. [[CrossRef](#)]
11. Liu, X.; Tang, Z.; Li, P.; Guo, S.; Fan, X.; Zhang, J. A graph learning based approach for identity inference in dapp platform blockchain. *IEEE Trans. Emerg. Top. Comput.* **2020**, *10*, 438–449. [[CrossRef](#)]
12. Yu, L.; Zhang, F.; Ma, J.; Yang, L.; Yang, Y.; Jia, W. Who Are the Money Launderers? Money Laundering Detection on Blockchain via Mutual Learning-Based Graph Neural Network. In Proceedings of the 2023 International Joint Conference on Neural Networks (IJCNN), Gold Coast, Australia, 18–23 June 2023; pp. 1–8.
13. Chen, L.; Peng, J.; Liu, Y.; Li, J.; Xie, F.; Zheng, Z. Phishing scams detection in ethereum transaction network. *ACM Trans. Internet Technol. (TOIT)* **2020**, *21*, 1–16. [[CrossRef](#)]
14. Huo, X.; Li, M.; Zhou, Z.-H. Control flow graph embedding based on multi-instance decomposition for bug localization. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 4223–4230. [[CrossRef](#)]
15. Grover, A.; Leskovec, J. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 855–864.
16. Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; Mei, Q. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 1067–1077.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.