STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXX**, Number 1-2, 2025
DOI: 10.24193/subbi.2025.06

MONEY LAUNDERING DETECTION USING GRAPH NEURAL NETWORKS ENHANCED WITH AUTOENCODER COMPONENTS

TUDOR-IONUŢ GRAMA

Abstract. The paper addresses the topic of detecting money laundering operations in transaction data represented as graph data-structures. We propose the integration of autoencoder components in Graph Neural Networks(GNN) architectures, in order to incorporate a reconstruction step in the traditional edge classification problem and enhance model quality based upon the usage of reconstruction errors.

We show that enhancing GNNs with autoencoder components improves the predictive performance of money laundering detection, on data represented as homogeneous graphs. Additionally, the Shapley value is computed in order to gain further insight into the most important features from distinguishing normal and fraudulent activities.

1. Introduction

With the latest technological achievements, there has also been a significant increase in the complexity of fraudulent financial activities, as well as in the ability of criminal parties to render such efforts much more difficult to trace. *Money Laundering* is the process of making illicit funds hard or impossible to distinguish from those acquired by legal means, and causes damages in the billions, potentially even trillions [4]. Identifying such processes as accurately and early as possible is paramount for limiting both the resulting damages, as well as for putting a stop to the ability of organized crime to be able to acquire finances. Money Laundering is a complex process of obfuscating the origin of illicitly obtained funds by employing sequences of transactions or bank transfers, that make tracing the source of the money very difficult and

Received by the editors: 4 June 2025.

2010 *Mathematics Subject Classification.* 68T05, 68T99.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets*; E.1 [**Data**]: Data structures – *Graphs and networks*.

Key words and phrases. Money laundering, Graph neural networks, Autoencoders, Explainability.

time-consuming. Traditional approaches for money laundering detection are typically rule-based systems [22], which can be prone to a high degree of false positives [16]. Recently, machine learning and deep learning driven approaches have seen significant development [9, 1]. Of particular note are developments in *graph neural networks* (GNNs), which manage to effectively depict complex networks of financial transactions [10], due to their ability of handling relational data, represented as graphs, which are a suitable representation for financial transactions.

This work showcases GNNs with directed multigraphs, as well as various other adaptations which are either necessary for accurately depicting financial transactions or enhance predictive performance. Financial operations can encompass multiple transactions which can occur between the same entities over time, often time in both directions. The multigraph structure thus more dependably preserves the temporal and directional aspects of transactions and can serve as a better baseline for future developments. The main goal is to develop a robust manner of detecting instances of money laundering, that, at the same time is agnostic with respect to the transaction method, as well as being generalizable to a large deal of auxiliary factors, such as currency, financial institutions, or whether the transactions go beyond national boundaries. This should serve to ensure robustness for the complexities specific to the domain in real-life scenarios and improve the detection of such illicit operations, no matter the mechanisms involved.

To the best of our knowledge, there are no other studies in the literature that discuss the enhancement of GNN architectures with autoencoder components for the purpose of detecting money laundering. Our contribution consists of using autoencoder components in order to reconstruct edge attributes based on the embeddings of the nodes defining a transaction. The reconstruction error is used as part of the training loop being integrated in a binary cross entropy function which uses the edge attributes as well as the mean square error pertaining to the edge attribute reconstruction for the classification step. The output of the autoencoder is used to better differentiate transactions involved in money laundering that are not part of known or well defined patterns. Second, we integrate explainability and feature analysis using `SHapley Additive exPlanations` (SHAP) [14], in order to better understand what structural and substantive features of financial data are most telling of whether a transaction is normal or fraudulent. To conclude, three main research questions will be further investigated:

RQ1 Does integrating autoencoder components into the edge classification problem improve the predictive performance of money laundering detection?

RQ2 What is the impact of integrating autoencoder components in a GNN-based architecture for the task of detecting illicit transactions in directed heterogeneous multigraphs?

RQ3 What are the most significant features for detecting each type of transaction as provided by the SHAP explainability approach?

The rest of the paper has the following structure. Section 2 reviews existing machine learning approaches for money laundering detection. Section 3 details our methodology used in the study. The experimental results are presented and discussed in Section 4. Section 5 presents the application of SHAP values to provide interpretability for transaction classification, while the conclusions and some potential directions for future work are outlined in Section 6.

2. Literature review on machine learning approaches for money laundering detection

Weber et al. [23] employ GNNs for anti money laundering in bitcoin transactions. The researchers implement an architecture using two-layered graph convolutional networks. The authors name this Skip-GCN, incorporating a skip connection between intermediate embeddings and input features, and also experiment with EvolveGCN, a temporal model that creates separate GCN (Graph Convolutional Network) layers for each time step connected through a recurrent neural network. They show that the graph-based approaches outperform traditional methods.

GNN-based approaches have also been employed on real-life data. Johannessen and Jullum use a GNN architecture in order to detect money laundering in a network of transaction from the largest Norwegian bank. One variant explored by the researchers include a novel approach, by concatenating the node embeddings from the aggregation of messages and passing the resulting vector through a single-layer perceptron, which outputs a new representation. The model achieves impressive results on the detection of top 1% most suspicious customers, with two thirds of the ones identified by the model being actually suspicious. This is noteworthy especially due to the severe class imbalance, under 0.5% of the customers being considered suspicious [10].

Egressy et al. [5] introduce key adaptations to standard GNNs in order to render them a better fit for graphs of transactions. They implement port numbering (unique labels for each node), EgoIDs (boolean marker which allows a node to detect whether it is part of a cycle) and reverse message passing, as well as edge updates, for some of the model variants. In their experiments, reverse message passing provides the largest single performance boost for most tasks, Port numbering is shown to be essential in detecting two patterns that are common in fraudulent financial transactions. EgoIDs, coupled with the

other two adaptations are stated to improve the detection of cycle patterns, which are likewise common in fraudulent activities, up to a depth of 6.

There are other, non-GNN based approaches that have also shown promising results. Kumar et al have tested various machine learning models on data of bank loans. Support vector machines shows the best precision score, obtaining 100%, while also achieving a high accuracy score and a high f1 score, of 83% and 90% respectively. In the same study, random forest has also been shown to display competitive performance [12].

3. Methodology

This section introduces the methodology used in our study, in order to implement our proposed architectures. We describe the dataset and the preprocessing steps necessary for the data, as well as the methods used towards the implementation of the used models and the performance evaluation.

3.1. **Machine learning models used.** *Autoencoders*(AEs) are a class of neural networks that correspond to the category of self-supervised learning machine learning models which learn to reconstruct the given input data. An AE is composed of two parts which are connected by two layers. The first segment is an encoder which learns to compress the data to a latent space. If we consider the input space to have n dimensions and the desired size to be m dimensions, then we can formalize the process as such: $f : \mathbb{R}^n \to \mathbb{R}^m$. The decoder learns the reverse process $g : \mathbb{R}^m \to \mathbb{R}^n$, which teaches the model how to reconstruct the input data based on lower dimensional latent representations.

GNNs are models that are specifically adapted in order to handle data structured in a graph like format, of nodes connected by edges [26]. The neural networks typically handle particular forms of data depending on the graph structure. We can consider graphs to be either directed or undirected, depending on the presence or absence of direction at edge level, respectively homogeneous or heterogeneous. In homogeneous graphs, all the edges and nodes have the same types, where in heterogeneous graphs, multiple types can exist simultaneously.

The most common approach for developing GNNs is through the Message Passing Neural Network Framework (MPNN), first defined by Gilmer et al. [6]. Developments of GNN architectures that include autoencoder components are somewhat rare. Nevertheless, there are some notable research results. Kipf and Welling present two different approaches, one being a graph autoencoder (GAE) and the other a variational adaptation of the previous algorithm [11]. An innovative approach is proposed by Tang, namely Graph Auto-Encoder Via Neighborhood Wasserstein Reconstruction [20]. The researchers identify a shortcoming of normal autoencoders, that being their tendency to focus

on direct edges between nodes, while losing otherwise relevant topological information of the graph.

3.2. **Overview.** We aim to represent financial transactions in a graph structure, with a node being a particular account and the edges being transactions between said accounts. As there can exist multiple transactions between the same two accounts, the underlying dataset is represented as a directed multigraph. The data has been produced by the AMLWorld model from IBM [2]. The data generation is done using a multi-agent based approach, in which there are both agents that are simulating normal citizen behavior, as well as a minority of criminals, which generate the fraudulent transactions.

In the literature, multiple typical patterns have been observed. These patterns were first introduced by Suzumura and Kanezashi [19], and can be described as follows. A `fan-out` pattern (further denoted as fan_out) represents a fraudulent node being connected by outgoing edges to other non-fraudulent nodes. A `fan-in` pattern (further denoted as fan_in) is similar, but in this case the fraudulent node has incoming directed edges from normal nodes.

A message is created by every node based on the internal features of the node, the features of the edge caring the message, as well as the features corresponding to the connected nodes. Nodes aggregate incoming messages from their neighbors, using one aggregator, or multiple in conjunction. Afterwards each node updates its internal state, aggregating its initial state as well as the states of its neighbors.

The way we convert a traditional graph structure into a multigraph is using the approach defined by Egressy et al. [5], meaning we combine the use of port numbering, reverse message passing and EgoIDs. Port numbering consists in adding a label to the edges between nodes, the label being a tuple containing a number uniquely identifying the start node (the node sending the messages) and the end node (the node receiving the messages). Reverse message passing is implemented by using a separate layer for message-passing, for the incoming edges and another layer for outcoming edges. This makes nodes in MPNN able to differentiate incoming and outgoing messages. Finally, EgoIDs are used in order to ease the detection of cycles in graphs.

3.3. **Proposed architectures.** We propose two different architectures, one tailored for simpler, homogeneous graph data, as well as one with multiple adaptations and capability of handling heterogeneous data.

For the *first model*, we begin with a feature embedding layer, which projects the node and edge features into the latent space, in order to ensure the same dimensions for both set of features. The second part of the architecture is what we call the GNN Encoder. As in the base architecture of Graph Isomorphism

Network (GIN), this component is tasked with updating node embeddings using the information gathered from the local neighborhoods. We use the GINE variant of GIN (GIN with Edge features), which is an adaptation of the latter, capable of integrating edge features and has shown strong ability in differentiating graph structures. We use two layers of GINE in this component, in order to make the aggregation of information able to occur using data from neighboring nodes separated by up to two edges. The third component is a decoder that aims to reconstruct edge attributes from the node embeddings. The idea behind this approach is that the nodes would be more inclined to preserve and encode features that are more important for defining the edges in their close proximity. Second, it allows us to identify edges that display anomalous behavior, which is of interest, as we presume that fraudulent edges have some, albeit variable, specifics that ultimately make their representations deviate from expected patterns. The last component is a classifier, that combines the node embeddings, edge features and the reconstruction error for classification of transactions. The structure of the classifier is a three-layered Multi-Layer Perceptron (MLP), with two dropout layers with a considerable final dropout rate of 29%.

Below we showcase the core training loop of the model, as conducted for the results detailed in Table 3, without any other adaptations, such as those defined in Egressy et al. [5]:

The *second* architecture is similar to the first in many aspects, but there are noteworthy differences, which we will focus on further below. The feature embedding layer is enhanced in order to be able to handle both homogeneous and heterogeneous graph structures, as according to the approach in Egressy et al. [5]. The GNN Encoder has two other adaptations, besides the use of PNAConv (Principal Neighborhood Aggregation Convolution) instead of GINEConv, namely, compatibility with reverse message passing, by aggregating information both from incoming and outgoing messages, as well as optional edge updates. Otherwise, we also use two layers of convolutions, in order to keep the same depth when aggregating node features. The decoder structure is slightly altered, in the sense that the architecture is now symmetric, with two processing layers. The classifier respects the original structure. Below we showcase the training loop of the PNA-based, autoencoder-enhanced GNN. For the purposes of the results shown in the Table 3, we have implemented all the adaptations described in Egressy et al. [5], as well as the edge updates mechanism. The baseline architecture is called Multi-PNA, which refers to a PNA architecture enhanced with EgoIDs, reverse message passing and port numbering.

Input: $G = (X, E, A, Y)$ - graph with node features, edges, edge features, and labels; T - number of epochs; L - number of GNN layers
for *epoch* $= 1$ ***to*** T **do**
  for *batch* (X_b, E_b, A_b, Y_b) ***in*** G **do**
    $A_{orig} \leftarrow A_b$.clone() `// original edge attributes`
    $H \leftarrow \text{NodeEmbed}(X_b)$ `// node embedding`
    $A_{emb} \leftarrow \text{EdgeEmbed}(A_b)$ `// edge embedding`
    `// GIN Encoder with residual connections`
    for *layer* $i = 1$ ***to*** L **do**
      $H \leftarrow (H + \text{ReLU}(\text{BN}_i(\text{GINEConv}_i(H, E_b, A_{emb}))))/2$ `// batch normalized computations`
    end
    $Z \leftarrow \text{Linear}_{enc}(H)$ `// project to latent space`
    $Z_{src}, Z_{dst} \leftarrow Z[E_b[0]], Z[E_b[1]]$ `// source/dest embeddings`
    `// decoding edge attributes`
    $\hat{A} \leftarrow \text{DecoderMLP}([Z_{src} \| Z_{dst}])$
    `// Per-edge reconstruction error`
    $\epsilon \leftarrow \text{mean}((\hat{A} - A_{orig})^2, \text{dim} = 1)$
    `// features augmented with reconstruction error`
    $F \leftarrow [Z_{src} \| Z_{dst} \| A_{orig} \| \epsilon]$
    $\hat{Y} \leftarrow \text{Classifier}(F)$
    $\mathcal{L} \leftarrow \text{CrossEntropy}(\hat{Y}, Y_b)$
    Backpropagate $\mathcal{L}$ and update model parameters
  end
end

Algorithm 1: GIN-based GNN with autoencoder enhancement

3.4. **Training and testing.** The models were trained on the training portion of the `Small HI` dataset, more precisely, the first 60% of the timestamps, validated on the next 20%, and tested on the final 20%. The segmentation was done based on the temporal ordering of the transactions. For all cases, the training has been conducted using a batch size of 8192, over 100 epochs.

As mentioned in Section 3.3, both models use two GNNs layers, we further employ [100, 100] neighbor sampling for both created models, in order to keep our approach as similar as the one of Egressy et al. [5] Both models utilize a weighted binary cross-entropy loss, with a weight of 1 for the normal transactions and 7.1 for the laundering transactions, for the GNN component. We have used the Adam optimizer and set learning rate to 0.0006 for both the GIN based architecture as well as the PNA based architecture. We have opted for a lower learning rate in order to mitigate the risk of the models overfitting on the patterns identified in the training data, which would be exacerbated by the use of reconstruction error. In addition, we use hidden sizes of 64 for the GIN based architecture, and 20 for the PNA based one, as well as latent dimensions of 32 and 12, respectively. For both architectures, the decoders also use a dropout rate of 0.08.

Input: $G = (X, E, A, Y)$ - graph with node features, edges, edge attributes, labels; T - number of epochs; L - number of GNN layers

for *epoch* $= 1$ ***to*** T **do**
  for *batch* (X_b, E_b, A_b, Y_b) ***in*** G **do**
    $E_{fwd} \leftarrow E_b$ `// forward edges`
    $E_{rev} \leftarrow E_b$.flip(0) `// reverse edges`
    $A_{orig} \leftarrow A_b$.clone() `// original attributes`
    $H \leftarrow \text{NodeEmbed}(X_b)$ `// node embeddings`
    $A_{fwd} \leftarrow \text{EdgeEmbed}(A_b)$ `// forward edge embeddings`
    $A_{rev} \leftarrow \text{EdgeEmbed}(A_b)$ `// reverse edge embeddings`
    for *layer* $i = 1$ ***to*** L **do**
      $H_{fwd} \leftarrow \text{ReLU}(\text{BN}_i^{fwd}(\text{PNA}_i^{fwd}(H, E_{fwd}, A_{fwd})))$ `// batch normalized computations`
      $H_{rev} \leftarrow \text{ReLU}(\text{BN}_i^{rev}(\text{PNA}_i^{rev}(H, E_{rev}, A_{rev})))$
      $H \leftarrow (H + H_{fwd} + H_{rev})/3$ `// residual aggregation`
      $A_{fwd} \leftarrow A_{fwd} + \text{EdgeMLP}_i([H[E_b[0]] \| H[E_b[1]] \| A_{fwd}])/2$
      $A_{rev} \leftarrow A_{rev} + \text{EdgeMLP}_i([H[E_{rev}[0]] \| H[E_{rev}[1]] \| A_{rev}])/2$
    end
    `// encoding to latent space`
    $Z \leftarrow \text{Linear}_{enc}(H)$
    $Z_{src}, Z_{dst} \leftarrow Z[E_b[0]], Z[E_b[1]]$
    `// decoding edge attributes`
    $D \leftarrow \text{ReLU}(\text{Linear}_{expand}([Z_{src} \| Z_{dst}]))$
    for *layer* $j = 1$ ***to*** L **do**
      $D \leftarrow \text{Dropout}(\text{BN}_j(\text{ReLU}(\text{Linear}_j(D))))$
    end
    $\hat{A} \leftarrow \text{Linear}_{final}(D)$ `// reconstructed edge attributes`
    `// Reconstruction error as feature`
    $\epsilon \leftarrow \text{mean}((\hat{A} - A_{orig})^2, \text{dim} = 1)$ `// per-edge MSE`
    $F \leftarrow [Z_{src} \| Z_{dst} \| A_{orig} \| \epsilon]$
    $\hat{Y} \leftarrow \text{Classifier}(F)$ `// features augmented with reconstruction error`
    $\mathcal{L} \leftarrow \text{CrossEntropy}(\hat{Y}, Y_b)$
    Backpropagate $\mathcal{L}$ and update parameters
  end
end

Algorithm 2: Multi-PNA Autoencoder for Anti-Money Laundering Detection

The AE components, in both cases, provide a reconstruction error calculated using mean square error. The edge attributes are reconstructed based upon the node embeddings. The reconstruction error is calculated per feature and then it is averaged across features. For the Multi-PNA architecture, we also take into consideration the node embeddings coming from the reverse message passing layers when conducting reconstruction. In the end, besides the node and edge attributes, we also use the reconstruction error as part of the classification step

In order to assess model performance, we measure the `F1`-score of the minority class at the end of the last training epoch.

4. Experimental evaluation

This section presents the experiments enacted in order to answer the aforementioned research questions, namely whether autoencoder components are beneficial for detecting fraudulent transactions in GNN architectures and if these effects remain valid for a heterogeneous graph. The experimental evaluation has been carried out using the methodology presented in Section 3.

We mention that part of the baseline models from the Egressy et al. [5] paper have been tested during this study, in order to validate their performance and determine where the defined approach is valid and a potentially worthwhile starting point of further study.

4.1. **Dataset.** We have chosen the `Small HI` (Higher Illicit) transaction subset of the dataset. We consider the Higher Illicit transaction set to be more suitable in comparison to the Lower Illicit set. While this subset is less representative of real-life scenarios, the ratio of transactions involved in laundering being just slightly over 0.1, we consider that less illicit transactions would considerably hinder the capabilities of the model to learn the patterns associated with the minority class. This subset of data contains information for a primary period of 10 simulated days, with around 5 million transactions, realized by 515 thousand bank accounts.

Out of all the transactions in the main period, only approximately 3600 are illicit. Table 1 presents the statistics for the small cohorts of the data, both higher and lower illicit sets. The dataset includes a secondary period of 8 days, succeeding the primary period, with a much smaller number of accounts and transactions.

Statistics	Higher Illicit	Lower Illicit
Number of Days Spanned	10 + 8	10 +7
Number of Bank Accounts	515K +716	705K + 201k
Number of Transactions	5M + 1.1K	7M +283K
Number of Laundering Transactions	3.6K +655	4.0K + 435
Laundering Rate (1 per N Trans)	981	1942

Table 1. Statistics for the 'Small' subset of the dataset.

The initial features of the dataset are explained in the Table 2

When the data is processed for training and inference, the timestamp column, payment currency, receiving currency and payment format columns are encoded using label encoding. The columns 'to bank' and 'to account' are

Feature	Description
Timestamp	Timestamp of the transaction in YYYY/MM/DD HH:MM Format
From Bank	Numeric ID of the bank the transaction is sent from
Account	Hexadecimal ID of the account on the sending end of a transaction (The initial feature name is identical to the other Account name, later gets changed to 'From Account')
To Bank	Numeric ID of the bank the transaction is sent to
Account	Hexadecimal ID of the account on the receiving end of a transaction (Later changed to 'To Account')
Amount Received	Monetary amount received by 'To account'
Receiving Currency	Currency of the amount received by 'To Account'
Amount Paid	Monetary amount send to the 'To account'
Payment Currency	Currency of the amount sent by 'From Account'
Payment Format	How the transaction was conducted (Wire transfer, cheque)
Is Laundering	Boolean value of whether the transaction is money laundering or not

TABLE 2. Feature explanation

turned into a singular column, by appending the value of the latter to the former. This is similarly done for the 'from bank' and 'from account' columns. Finally, an EdgeID is created, which is an integer value that uniquely identifies every transaction.

The data that is inputted into the model, in the case of the two architectures, is identical, both in the case of the models which implement directed homogeneous multigraphs and those which implement directed heterogeneous multigraphs. The features used are the following:'Timestamp', 'Amount Received', 'Received Currency', 'Payment Format'. The only changes between heterogeneous and homogeneous multigraphs are those pertaining to the edge index, which needs to be flipped when handled by the reverse message passing layer, as well as the port numbers, if port numbering is enabled. The main differentiator between the layers is which messages are handled by which layer, namely homogeneous implementations only use the data from incoming messages, the edge features of incoming edges, in order to update a node's internal representation, while in heterogeneous graphs, the outgoing messages are also used, together with the features of the outgoing edges and handled by the reverse message passing layer. This implementation is used in order to remain consistent with the implementations of Egressy et. al.[5]. For example, given the following example transaction:

2019/01/01 00:22,800319940,8004ED620,808519790,872ABC810,120.92,US Dollar,120.92,US Dollar,Credit Card,0

The final features kept would be the Timestamp (2019/01/01 00:22), Amount Received (120.92),Received Currency (US Dollar) and the Payment Format (Credit Card). The Is_Laundering (0) value is used as the target variables.

If port numberings is enabled, unique port numbers are added as edge features. We shall consider the starting node to be 800319940-8004ED620, encoded as '0' (zero), and the receiving node to be 808519790-872ABC810, encoded as '1'. Therefore, the port numbering for this edge would be the tuple (0, 1). The edge index is likewise represented as a tuple, respecting the encoded format (0, 1). For node '1', in an homogeneous graph implementation, the edge features and internal node representation for node '0' would be used, as '1' is a receiving node and the transaction (0, 1) represents an incoming transaction for node '1'. Node '0' would not use this transaction to update its internal state, as it represents the sending node, of this transaction.

When we activate reverse message passing, the original message passing layer would keep the behavior of a directed homogeneous multigraph. The reverse message passing layer needs to reverse the edge index and port numbering tuple, thus becoming (1, 0). In the case of the reverse message passing layer, node '1' acts as the sender, and '0' as the receiver. By enabling reverse message passing, now both nodes are able to use all the receiving and outgoing transactions and messages, in order to update their internal state. The resulting structure represents the directed heterogeneous multigraphs.

4.2. **Results and discussion.** Table 3 comparatively presents the performance of baselines and our proposed architectures. The most extensive results in the literature, specifically on the used dataset, using part of the adaptations discussed are presented in the work of Egressy et al. [5]. We consider these models as baselines for our architectures, which are written as GIN+AE (GIN+ AutoEncoder) and Multi-PNA+EU (Edge Updates)+AE respectively. The experiments using the models GIN + ReverseMP + Ports, R-GCN and all the experiments prefixed with 'Multi-' handle the dataset as a directed heterogeneous multigraph, the others as a directed homogeneous multigraph. The tree-based model baseline use the dataset augmented with graph features. As Anti-Money Laundering systems are prone to false positive rates approaching 90%[15], due to the resulting negative effect on precision, we can assess that F1-scores are likely to remain reduced. To our knowledge, Multi-PNA+EU+AE represents the state of the art for money laundering detection on directed heterogeneous multigraphs and GIN+EgoIDs represents the state of the art on directed homogeneous multigraphs.

From Table 3, which consists of the average across five runs with different seeds, plus-minus the standard deviation, in case of the baselines, we observe that the architectures that exhibit all three of the adaptations mentioned

TABLE 3. Performance of baselines and proposed architectures (F1-score %).

Model	**AML Small HI**
LightGBM+GFs [2]	62.86 ± 0.25
XGBoost+GFs [2]	63.23 ± 0.17
GIN [24, 7])	28.70 ± 1.13
GIN+AE	34.98
PNA [21]	56.77 ± 2.41
GIN+EU [3]	47.73 ± 7.56
R-GCN [18]	41.78 ± 0.48
GIN+EgoIDs [25]	39.65 ± 4.73
GIN+Ports [17]	54.85 ± 0.89
GIN+ReverseMP [8] +Ports	46.79 ± 4.97
GIN+Ports	56.85 ± 2.64
+EgoIDs (Multi-GIN)	57.15 ± 4.99
Multi-GIN+EU	64.79 ± 1.22
Multi-PNA	64.59 ± 3.60
Multi-PNA+EU	68.16 ± 2.65
Multi-PNA+EU+AE	50.92

Egressy et al. [5], namely the EgoIDs, port numbering and Reverse Message Passing display the best performance. For the graph isomorphism network, port numbering shows the biggest performance impact, with an increase from the minority class `F1` score from 28.70 ± 1.13 to 54.85 ± 0.89. The second biggest impact comes from Edge updates, followed by Reverse Message Passing and EgoIDs, with the lowest impact. It is shown that combining all these adaptations render the comparably simpler GIN architecture to be very close in performance with the Multi-PNA Model.

Adding an autoencoder component on the normal GIN architecture has a positive impact on performance, with an increase of the evaluation metric to 34.98%, which is greater than the values exhibited at the tall end of a confidence interval given by the researchers. This tells us that autoencoder components can be beneficial in directed, homogeneous graphs.

However, in the case of heterogeneous graphs, we have observed a large drop in performance when compared to the equivalent baseline implementation (Multi-PNA+EU), obtaining a minority class `F1` of 50.92% We theorize that this might be due to the increased complexity of the resulting node embeddings, coupled with the higher volume and diversity of edges and their corresponding features, which might necessitate special adaptations in the reconstruction process, for stable performance. One approach that might result in potential improvements is to consider the inbound and outbound messages as two separate homogeneous graphs, strictly for the purpose of calculating reconstruction errors. Then, we calculate the reconstruction error separately

for each instance and add them both in the final classifier. This could lead to improved performance if the boost associated with the autoencoder component remains consistent across multiple homogeneous graphs. For the purpose of testing of our implementations, due to time and computational constraints, we had ran the experiments only once.

Therefore, the increase in performance exhibited in the case of the first model, when compared to the baseline renders, corroborated with the lower performance exhibited by the second model renders us able to answer **RQ1**: the performance improves by adding autoencoder components in the case of homogeneous graphs, and decays in the case of heterogeneous graphs. As an answer to **RQ2**, we may may that the performance decays when integrating autoencoder components, when compared to the baseline, at least in the case of our architecture. Nevertheless, further research and tests are required for definitive validation. We note that there are no other graph autoencoder architectures in the literature that deal with the problem of money laundering detection as an edge labeling task

5. Explainability

For answering **RQ3** we further investigate the application of SHAP values for providing interpretability for transaction classification.

SHAP values are based upon shapley values and provide a standardized approach for estimating the effect and the associated magnitude of the input features on the model output. The SHAP values are computed using explainer models and can be used in order to infer feature importance for all predictions as a whole, as well as individually. In the first case, they can provide an aggregate overview of the feature importance for the global predictive behavior of the model, more precisely showing the effect of the feature on the given predictions. This depiction can be rendered even more informative, with the use of beeswarm plots, which show how values of the feature influence the importance of the feature in prediction, or scatter plots, which can model interaction effects between feature values and their associated SHAP values. Second, SHAP values can be used to infer feature importance for individual predictions made by the model. This is relevant for explaining how the model reaches a result for a particular observation, and, likewise, we can use either waterfall plots or force plots for this end.

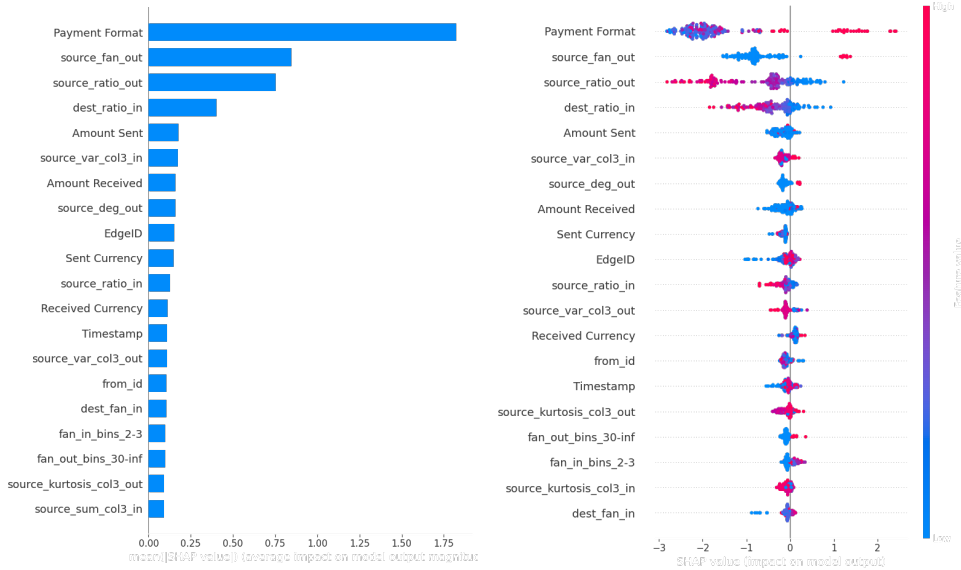
Due to the complexity of computing SHAP values for graph structure data, particularly for edges, we will first process the data in order to extract meaningful structural features, which will render the data more usable by more traditional classifiers. To this end, we will employ the Graph Feature Preprocessor module from the SNAPML library. The preprocessor computes two kinds of graph based features, which the researchers label as graph patterns

and graph vertex statistics. The first set of features consists of identifying for each transaction the kind of patterns it is implicated in. The second set of features is represented by features like the number of neighboring edges, both incoming and outgoing as well as the `fan`/`degree` ratio (the `fan` is the number of neighboring nodes and the `degree` is the number of incident edges), as well as `average`, `sum`, `minimum`, `maximum`, `median`, `var`, `skew`, and `kurtosis` of the selected raw features of the aforementioned three features. All the features shown in Figure 1 and Figure 2, written in snake-case represent augmentations computed with SNAPML. The total final number of features is 100.

The feature explanation has been computed by wrapping an XGBoost model using a `TreeExplainer` model from SHAP [13], which is an architecture that is adapted for SHAP values computation for tree-based models. XGBoost is an efficient implementation of gradient boosting machines that has seen widespread adoption in machine learning competitions.

The global and beeswarm feature importances are shown in Figure 1. Looking at the global feature importance (Figure 1(A)), we can see that four features show the greatest influence in the final result: (1) the most important predictor on whether a transaction is fraudulent or not is `Payment Format`; (2) the second feature refers to the number of neighboring nodes of a source node (`source_fan_out`); (3) the third feature represents the ratio between the `fan` and the *degree* for a starting node of the outgoing transactions; (4) the fourth feature is similar, but it instead is computed on the destination node.

We can see that, the number of the neighboring nodes of a source node is a strong predictor for fraudulent transactions. The correlation between different amounts sent and fraud does not seem to be very clearly defined, but low transaction amounts have a small tendency of lowering the risk of a transaction being fraudulent. Another insight we can gain from the plot is by looking at the EdgeID feature. The edges in the dataset are labeled using numeric ids, which are incremented by 1 for every subsequent transaction that is simulated. As the edges with low feature values (a lower ID) are shown to decrease the chance a feature is actually fraudulent, we can stipulate that the researchers have first created the normal transactions, and only afterwards created the bulk of fraudulent transactions. As the transactions with medium-valued EdgeIDs are shown to increase the probability of being a money laundering transaction (in this instance, meaning they are actually fraudulent), we can assume with reasonable certainty that the fraudulent transactions were simulated in a later step of the simulation, and late transactions act as filler, likely necessary for increased veracity. This is further corroborated by the `Timestamp` features, where we can see lower timestamps mostly corresponding to normal transactions.



(A) Global Feature Importance. (B) Beeswarm feature importance.

FIGURE 1. Feature importance.

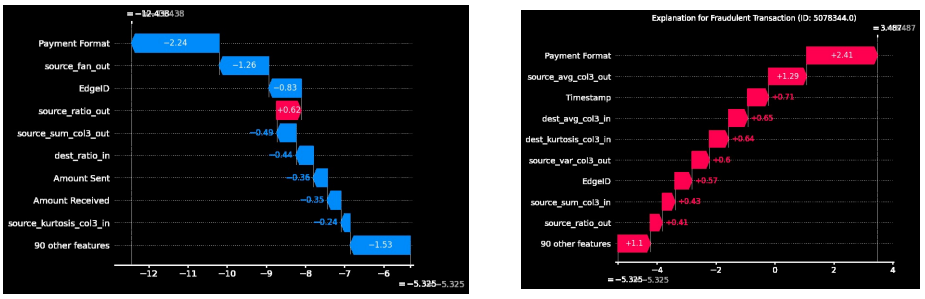
We can also stipulate that the transactions were generated in batches, as even for a small subset of samples (the first two hundred), the bee swarm plot and the influence of feature values remain mostly consistent with the previous case. The beeswarm plot illustrated in Figure 1(B) showcases this behavior.

TABLE 4. Transaction statistics per payment format

Payment	Not Laund.	Laund.	Total	Fraud %
ACH	596,314	4,483	600,797	0.75
Bitcoin	146,035	56	146,091	0.04
Cash	490,783	108	490,891	0.02
Cheque	1,864,007	324	1,864,331	0.02
Credit Card	1,323,118	206	1,323,324	0.02
Reinvestment	481,056	0	481,056	0.00
Wire	171,855	0	171,855	0.00

For a better comprehension of the importance of the features, we consider two sample transactions depicted in Figure 2 (a normal one, depicted in Figure 2(A) and a fraudulent one depicted in Figure 2(B)).

When looking at an individual **normal transaction** (Figure 2(A), we can see that most features make the sample less likely to be fraudulent, in this case, with the exception of the `fan/degree` ratio at the source node.



(A) Normal transaction - 298998. (B) Fraudulent transaction - 5078334

FIGURE 2. Sample transactions.

In order to better understand why these features have this particular impact, for this sample, on the model predictions, we analyzed the distributions of the nine most important features, according to global feature importance. We observed that the sample corresponds to a transaction conducted using reinvestments, that has a small, but above the mean `source_ratio_out`. We also observed that the transaction sent a large amount of money, well beyond the median but far below the mean, and the same holds for the amount received. Of note is the difference between the mean of the two, showing that the average received amount is much higher than the sent amount, signaling that monetary conversions into a weaker currency, than the starting one is particularly common. We can also see that the starting node of the transaction has 7 neighboring nodes, which is above the median, but well below the mean.

For the fraudulent transaction (Figure 2(B)), all the main features push the transaction towards being fraudulent. We observed that five of the nine most important features refer to statistical features of the timestamps (`col3` corresponds to the timestamp column). We note that the transaction was conducted using the ACH (Automated Clearing House) format, which was shown to have the highest percentage of fraudulent transactions 4. The transaction is also shown to have a small ratio of `fan/degree` for its starting node (has comparatively few transactions when compared to the number of implicated accounts.) The `source_fan_out` also shows us that the number of accounts connected to the source account is above the median, but well below the mean. For the end node, we can see the ratio is much closer to the mean and the median. Of note is also that the amount sent is only slightly larger than the median, but much lower than the mean. This suggests that transactions that are not particularly large, but still larger than most, are most likely to be part of fraudulent activity.

6. Conclusions

We have shown in this paper that enhancing GNN architectures with autoencoder components improves the predictive performance, when it comes to the money laundering detection tasks, on data represented as homogeneous graphs. This finding does not hold for heterogeneous graphs, at least when we consider the multi-PNA architecture, where we encountered a decrease in model performance. We theorize that this could indeed be due to the data differences, whose impact could be mitigated by utilizing different autoencoders for each type of data.

We can therefore state that, in regard to RQ1, autoencoder components can enhance the performance for homogeneous graph data. For heterogeneous data the performance decreases, more advanced implementations likely being necessary. This renders us able to answer RQ2, namely that the Multi-PNA architecture does not benefit from autoencoder components as implemented in our iteration. Regarding RQ3, we have shown that payment formats and the ratio of transactions to the participating accounts have a significant predictive capability in determining transactions involved in money laundering.

As the current research does not discuss the implementation of autoencoder components for all the model architectures discussed in the main paper, analyzing the impact of such adaptations to the other model variants will likely lead to valuable research results. Finally, in regards to explainability, developing SHAP adaptations for explainability on edge labeling tasks, directly applicable to GNNs would be a major improvement towards model interpretability. Otherwise, integrating new statistical features of transaction amounts, both for sent and received, as was done for the timestamps would likely also lead to valuable insights.

References

[1] Abal, R., Peña, L., and Soria Quijaite, J. Machine learning models for money laundering detection in financial institutions. a systematic literature review. In *Proceedings of the 22nd LACCEI International Multi-Conference for Engineering, Education, and Technology* (2024), pp. 1–10.

[2] Altman, E. R., Egressy, B., et al. Realistic Synthetic Financial Transactions for Anti-Money Laundering Models. In *Proceedings of NeurIPS 2023* (2023), pp. 1–24.

[3] Battaglia, P. W., Hamrick, J. B., et al. Relational inductive biases, deep learning, and graph networks. *CoRR abs/1806.01261* (2018), 1–40.

[4] Cassara, J. A. Countering international money laundering: Total failure is "only a decimal point away". Tech. rep., The FACT Coalition, Washington, DC, August 2017.

[5] Egressy, B., von Niederhäusern, L., et al. Provably Powerful GNNs for Directed Multigraphs. In *Proceedings of (AAAI-24)* (2024), AAAI Press, pp. 11838–11846.

[6] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *CoRR abs/1704.01212* (2017), 1–14.

[7] Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V. S., and Leskovec, J. Pre-training graph neural networks. *CoRR abs/1905.12265* (2019), 1–22.
[8] Jaume, G., Nguyen, A., et al. edGNN: a Simple and Powerful GNN for Directed Labeled Graphs. *CoRR abs/1904.08745* (2019), 1–9.
[9] Jensen, R. I. T., and Iosifidis, A. Fighting money laundering with statistics and machine learning. *IEEE Access 11* (2023), 8889–8903.
[10] Johannessen, F., and Jullum, M. Finding money launderers using heterogeneous graph neural networks. *CoRR abs/2307.13499* (2023), 1–20.
[11] Kipf, T. N., and Welling, M. Variational graph auto-encoders. *CoRR abs/1611.07308* (2016), 1–3.
[12] Kumar, S., Ahmed, R., et al. Exploitation of machine learning algorithms for detecting financial crimes based on customers' behavior. *Sustainability 14*, 21 (2022), 1–24.
[13] Lundberg, S. M., Erion, G., et al. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence 2*, 1 (2020), 2522–5839.
[14] Lundberg, S. M., and Lee, S.-I. A Unified Approach to Interpreting Model Predictions. In *Adv Neural Inf Process Syst 30*. Curran Associates, Inc., 2017, pp. 4765–4774.
[15] Murphy, A., Robu, K., and Steinert, M. The investigator-centered approach to financial crime: Doing what matters. *McKinsey and Company* (June 2020). Accessed: March 8, 2025.
[16] Richardson, D., Williams, and Mikkelsen, D. Network analytics and the fight against money laundering. McKinsey and Company, 2019.
[17] Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. *CoRR abs/1905.10261* (2019), 1–15.
[18] Schlichtkrull, M. S., Kipf, T. N., et al. Modeling Relational Data with Graph Convolutional Networks. In *Proceedings of ESWC 2018,* (2018), vol. 10843 of *Lecture Notes in Computer Science*, Springer, pp. 593–607.
[19] Suzumura, T., and Kanezashi, H. Anti-Money Laundering Datasets: InPlusLab anti-money laundering datadatasets. `http://github.com/IBM/AMLSim/`, 2021.
[20] Tang, M., Yang, C., and Li, P. Graph auto-encoder via neighborhood wasserstein reconstruction. *CoRR abs/2202.09025* (2022), 1–17.
[21] Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. *CoRR abs/1809.10341* (2018).
[22] Verhage, A. Supply and demand: anti-money laundering by the compliance industry. *Journal of Money Laundering Control 12* (10 2009), 371–391.
[23] Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., and Leiserson, C. E. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *CoRR abs/1908.02591* (2019), 1–7.
[24] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *CoRR abs/1810.00826* (2018), 1–17.
[25] You, J., Gomes-Selman, J. M., Ying, R., and Leskovec, J. Identity-aware graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence 35*, 12 (May 2021), 10737–10745.
[26] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *CoRR abs/1812.08434* (2018), 57–81.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
Email address: `tudor.grama@stud.ubbcluj.ro`