Amatriciana: Exploiting Temporal GNNs for Robust and Efficient Money Laundering Detection

Marco Di Gennaro, Francesco Panebianco, Marco Pianta, Stefano Zanero, Michele Carminati
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano
Milan, Italy
{marco.digennaro, francesco.panebianco, stefano.zanero, michele.carminati}@polimi.it
marco.pianta@mail.polimi.it

***Abstract*—Money laundering is a financial crime that poses a serious threat to financial integrity and social security. The growing number of transactions makes it necessary to use automatic tools that help law enforcement agencies detect such criminal activity. In this work, we present Amatriciana, a novel approach based on Graph Neural Networks to detect money launderers inside a graph of transactions by considering temporal information. Amatriciana uses the whole graph of transactions without splitting it into several time-based subgraphs, exploiting all relational information in the dataset. Our experiments on a public dataset reveal that the model can learn from a limited amount of data. Furthermore, when more data is available, the model outperforms other State-of-the-art approaches; in particular, Amatriciana decreases the number of False Positives (FPs) while detecting many launderers. In summary, Amatriciana achieves an F1 score of 0.76. In addition, it lowers the FPs by 55% with respect to other State-of-the-art models.**

***Index Terms*—Anti Money Laundering, Fraud Detection, Graph Neural Networks.**

I. Introduction

Money laundering schemes have long assisted organized crime in moving their financial assets while covering their tracks. This process can be defined as the conversion of illegally acquired currency into apparently legitimate assets, allowing perpetrators to evade detection and legal prosecution. When laundering is successful, criminals can reinvest the money to perform other criminal operations, expand their activities, and evolve their behavior, making detection much more complex. The process typically involves three stages: *placement*, *layering*, and *integration* [1]. At the *placement* stage, illicit funds are introduced into the financial system through deposits, asset purchases, or business investments. Then, during *layering*, complex transactions are conducted to obscure the origin of the funds, making them harder to trace. Finally, during *integration*, laundered money is reintroduced into the economy with an appearance of legitimacy.

In 2020, the total amount of non-cash payments in the euro area was €167.3 trillion, making it impossible to investigate each transaction manually [2]. Furthermore, transaction tracking is increasing in complexity due to the combined usage of different financial channels, such as cryptocurrencies and complicit foreign institutions.

Automatic fraud detection systems monitor and analyze the continuous stream of transactions to find suspicious activities. Classic rule-based detection efficacy is augmented by expert knowledge [3]. However, the static nature of these techniques enables criminals to learn and adapt their habits to remain undetected. More recently, Machine Learning (ML) was proven to be a more reliable choice [3]–[14]. Deep Learning (DL) techniques can identify complex patterns that are more challenging to bypass or evade. Among these, there are approaches based on Graph Neural Networks (GNNs) [15]–[18], [18], [19], i.e., DL architectures that extract features from connected graphs. Indeed, the advantage of using a graph representation is that it captures the relational information, which can be useful for correlating financial transactions.

While State-of-the-art (SOTA) GNNs capture relational information, they have limitations. Transductive methods require retraining for new samples, making them impractical for real-time monitoring. Inductive methods like GraphSAGE [20] handle unseen nodes but ignore temporal information, missing evolving patterns. EvolveGCN [21] addresses time dynamics but alters the graph topology, potentially obscuring key relationships. Additionally, these models often ignore edge features and struggle with memory issues on large graphs. In Anti-money laundering (AML), the lack of real-world datasets due to privacy concerns forces reliance on synthetic data generators. These generators provide fully labeled data and maintain a realistic class imbalance, making it difficult for classifiers to balance False Positives and False Negatives.

In this work, we propose Amatriciana: a graph-based framework to detect money laundering. Our approach improves the performance of SOTA detectors by introducing relevant features in the embedding extraction process. Specifically, it uses edge features and temporal information about the transaction to enrich the latent representation. Temporal information is incorporated into the embedding using a Long Short Term Memory (LSTM) [22] aggregator, which removes the need for numerous time-based subgraphs.

We further improve the training procedure by proposing a novel loss function based on the Matthews Correlation Coefficient (MCC). This objective focuses the update step on low-performing samples (e.g., FPs), using a tunable weighting scheme. Finally, we present a memory-efficient training procedure that creates batches for the nodes after computing the embedding. This enables training the LSTM aggregator and

the classificator simultaneously. In our evaluation, performed on the IT-AML generated dataset, Amatriciana outperforms all baselines with an Area Under the ROC Curve (AUC) of 0.82. Amatriciana also significantly improves the Precision of SOTA approaches, reaching 0.81. The baseline GraphSAGE, for comparison, is limited to only 0.67.

We summarize our contributions in what follows:

- We introduce Amatriciana, a novel framework for money laundering detection based on GNNs. It is based on GraphSAGE, with the addition of temporal information and other edge features.
- We propose a new loss function derived from the MCC. It tunes the weights to account for class imbalance and reduce FPs.
- We introduce a novel training framework designed to reduce the memory requirements for processing large graphs, eliminating the need for graph fragmentation.

II. Graph Machine Learning Primer

Graphs provide a way to model real-world entities and their relationship. More formally, a graph is defined as a set of nodes (also called vertices) $\boldsymbol{V}$ and a set of edges $\boldsymbol{E}$ that connect the nodes. The connections within a graph can be represented with a matrix called *adjacency matrix*. In such a matrix, each element corresponds to the weight of the edge between node i (row) and node j (column). Additionally, the graphs are enriched with features on both the nodes and the edges [23].

Traditional Deep Learning (DL) methods struggle in handling graph-structured data due to the absence of a fixed spatial structure. With the rise of DL, Graph Neural Networks (GNNs) provided a reliable architecture to solve several graph-related tasks. Leveraging the principles of Convolutional Neural Networks (CNNs), GNNs generalize the concept of convolution to graph domains, enabling the extraction of local features (also called *embeddings*) while preserving the graph's inherent topology. Recent advances in GNN architectures, such as Graph Convolutional Networks (GCNs) [24], Graph Attention Networks (GATs) [25], evolveGCN [21], and GraphSAGE [20], enable the solution of graph-related tasks like node classification, link prediction, and graph classification across various domains [23]. EvolveGCN is a GCN architecture that uses an RNN to extract a dynamic representation of the graph. Specifically, EvolveGCN creates time-based subgraphs and returns node classifications for each of them. On the other hand, GraphSAGE is a general inductive framework that leverages node features to efficiently generate node embeddings for never-seen examples. To compute the embedding of a node, it samples and aggregates the features of nodes connected to it. At the sampling stage, given a node, the algorithm selects some neighbors. At the aggregation stage, the algorithm aggregates the features of selected nodes using an aggregation function (e.g., mean, sum, or a function computed by a Neural Network). GraphSAGE is a *message-passing* architecture, wherein the sampled neighborhood features act as messages transmitted to the nodes for which embeddings are being computed.

III. Related works on AML

Deep Learning. Paula et al. [7] perform anomaly detection for money laundering with unsupervised DL. Their approach uses an *autoencoder* network, a model trained to reproduce the samples in input. The paper considers samples that are difficult to recreate as anomalies, and thus potential laundering transactions. Zhou et al. [8] and Zhang and Trubey [9] use traditional Machine Learning (ML) models like decision trees, Random Forest (RF), and Support Vector Machine (SVM) to detect suspicious money laundering activities. Jullum et al. [10] adopt a supervised learning approach to detect financial activities that are likely to be reported. Li et al. [11] change the task to community discovery, using a temporal-directed *Louvain* algorithm to detect communities according to relevant AML patterns. Kannan and Somasundaram [12] model the data as a time series and use autoregressive algorithms to detect outliers. Labanca et al. [13] propose *Amaretto*, an active learning framework that combines supervised and unsupervised learning to achieve better detection performance and reduce the cost of monitoring transactions for financial institutions. Finally, Savage et al. [14] analyze group behavior using network analysis and supervised learning.

Graph Neural Network. Weber et al. [26] show the promising results that GCNs can achieve on the money laundering detection task. Weber et al. [17] present an evaluation of various ML methods (including GCNs and, more specifically, evolveGCN [21]) for financial forensics with cryptocurrency transactions. Anomaly detection datasets are often imbalanced towards the legitimate class. To address this issue, Humranan and Supratid [15] propose using a Focal Loss function when training a GCN model. Karim et al. [18] approach the task by employing a semi-supervised graph learning technique. Cardoso et al. [19] propose LaundroGraph, a fully self-supervised approach that leverages GNNs to encode the representations of customers and transactions within the context of AML. The network of financial interactions is modeled as a directed bipartite graph, with the GNN trained for link prediction between pairs of customer and transaction nodes.

IV. Motivation

Rule-based money laundering detection struggles with emerging patterns, as criminals adapt to evade static rules. Dynamic systems like Machine Learning (ML) can identify new patterns but fall short without graph-based structures, missing critical relationships between entities. These relationships are crucial for detecting the complex transactions that define money laundering. As for State-of-the-art (SOTA) approaches, GNNs have recently been employed to capture relational information in financial transactions. GNNs have enhanced performance on the task, but current solutions come with their limitations. Many existing approaches are *transductive*, meaning that each new sample necessitates retraining - an effort that is impractical and costly. This limitation also prevents their application to real-time monitoring systems. In contrast, GraphSAGE [20] introduces an *inductive* approach,

enabling predictions on previously unseen nodes. However, it does not integrate temporal information into its embeddings, excluding patterns that develop over time. Alternatively, evolveGCN [21] addresses temporal variations by dividing the original graph into multiple time-based subgraphs. Although this method accounts for temporal dynamics, it alters the graph's topology, obscuring important relationships and patterns. Moreover, none of the aforementioned approaches incorporate edge features in the embedding process, which may restrict the model's ability to perform reliable predictions. Lastly, large graphs pose significant challenges in terms of memory usage, and current methods have yet to provide an efficient solution to this problem.

In the field of AML, a significant challenge is the lack of real-world datasets. Financial institutions typically do not share such data publicly due to the sensitive nature of financial transactions and the need to protect customer privacy. The few publicly available datasets often lack detailed feature descriptions. To develop effective money laundering detection systems, researchers rely on dataset generators that simulate realistic financial transactions, including both legitimate and illicit activities. These generators can fully track funds and label transactions. The advantage of using synthetic datasets is that data is complete and fully labeled. Additionally, it is important to note that money laundering is a rare activity, making these datasets highly imbalanced. This can lead to trained classifiers having difficulties balancing False Positives and False Negatives.

V. Amatriciana

Amatriciana is an Anti-money laundering (AML) approach that works with graph-structured transactions introducing a custom Graph Neural Network (GNN).

Amatriciana preprocesses raw transactions to generate a graph representing account interaction. As a supervised ML approach, it employs a GNN for a node classification task, distinguishing between two classes: launderer and lawful user. Thus, labeled nodes are required during the training phase. Indeed, when creating the graph from raw transactions, each node is labeled as either a launderer or a lawful user.

The GNN Amatriciana adopts is inspired by GraphSAGE [20], an inductive GML algorithm that learns to represent unseen nodes in a graph. Unlike GraphSAGE, Amatriciana incorporates edge features and transaction temporal information in addition to node features. The graph is split into training and validation subgraphs, each enriched with graph-dependent features to enhance model performance.

The model architecture consists of three key components: the *Node Encoder*, *Temporal Encoder*, and *Classifier*. The first one generates initial embeddings by aggregating node and edge features. The *Temporal Encoder* captures temporal patterns using a LSTM, grouping features by time steps. The *Classifier* combines these embeddings to classify nodes, using skip connections to retain intermediate information.

To handle the large graph size, we employ a memory-efficient training algorithm, divided into two phases: training the LSTM and training the Node Encoder and Classifier. Based on the MCC, the custom loss function addresses class imbalance by weighting metrics to enhance sensitivity to incorrect classifications.

In this section, we describe the preprocessing pipeline, the Amatriciana model architecture, and the detailed training workflow, including the custom loss function.

A. Data Preprocessing

Amatriciana needs to transform a raw dataset of transactions into a graph. Since datasets do not have a fixed structure, it is not feasible to directly transform any dataset into a graph with a fixed approach; therefore, we require an intermediate representation. During this phase, we create a list of transactions containing information such as sender and receiver, amount sent, transaction type, and other typical transaction-related data. All dates are converted into discrete hourly time steps, which are then used as temporal information in the Graph Neural Network (GNN). Additionally, Amatriciana maintains a list of accounts that have performed at least one transaction. These accounts are supplemented with their available information and a label classifying the account as either a launderer or a lawful user. For the training phase, the dataset must contain labels for each account or each transaction path. In the latter case, any account involved in at least one laundering path is considered a launderer. Due to the varying information contained in different datasets, Amatriciana can be configured to customize the information (edge and node features) to be taken into account.

Once Amatriciana obtains the dataset's intermediate representation (the list of transactions and accounts), generating a graph becomes straightforward. Each account in the list generates a node, while a transaction from one account to another generates an edge. Each node and edge is enriched by the attributes in the respective lists mentioned earlier. Finally, we use a multidimensional adjacency matrix (instead of a simple matrix) to represent the connections between nodes. This enables storing all the different time steps at which transactions between the same pair of nodes occur.

As previously mentioned, the key element of the Amatriciana approach is a Graph Neural Network (GNN) inspired by the well-known GraphSAGE framework. Since Amatriciana learns an inductive representation (i.e., it learns how to represent new, unseen graphs) and we do have hyperparameters to tune, we need to split the dataset into training and validation sets. However, instead of a traditional dataset, we are working with a graph. Therefore, Amatriciana splits the graph obtained during the preprocessing phase into two non-overlapping subgraphs. To achieve this, we select a label-based stratified subset of nodes (accounts) for the validation set. The subgraph containing these nodes and the edges between them is then used as the validation graph.

Finally, Amatriciana's preprocessing phase enriches the training and validation graphs with additional features computed separately for each graph. This approach ensures that the two graphs are not contaminated with features derived from

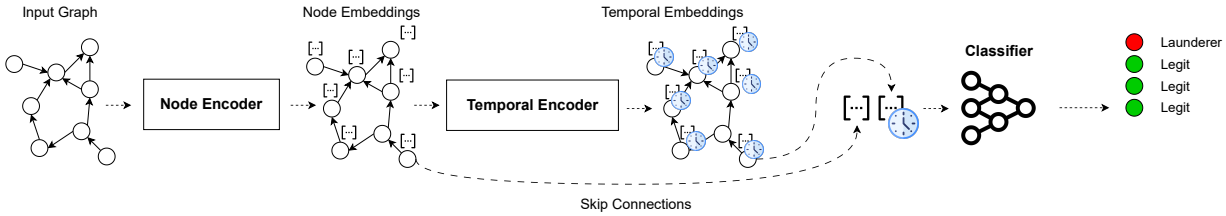


Figure 1: High-level Overview of Amatriciana Architecture

the other graph's information. These features, all related to nodes, can be classified into four categories:

Transactions-related features. Features derived from all the transactions in which the account is involved within the graph (*average transactions per step, minimum amount sent, minimum amount received, maximum amount sent, maximum amount received, variance of received amounts, variance of sent amounts*).

Topological features. Features added to enhance the model's understanding of the graph's topology (*incoming edges degree, outgoing edges degree, degree centrality, closeness centrality, eigenvector centrality, average neighbor degree*).

Clustering features. Features computed to provide information about how closely a node tends to cluster with other nodes and form a community, or how isolated a node is from the rest of the network. Communities and clusters can be crucial in identifying potential criminal activities (*clustering coefficient*).

Ranking features. We use PageRank [27] to assign more importance to nodes with a high number of edges or nodes connected to highly ranked nodes (*PageRank ranking*).

B. Model Architecture

After preprocessing raw transactions, we develop and train a Graph Neural Network (GNN) model to detect launderer nodes in a financial network. The model internally extracts node representations, namely *embeddings*, and uses them to classify the nodes. As previously mentioned, the GNN Amatriciana adopts is inspired by the GraphSAGE architecture. However, we consider edge features (transaction information) when aggregating neighborhood features, whereas the original GraphSAGE does not. To further improve the model's effectiveness, we introduce a custom sampling and aggregation strategy that groups node features based on time steps, using a Long Short Term Memory (LSTM) to capture temporal dependencies. Money laundering operations often span multiple transactions over time, so incorporating temporal patterns enables the detection of more complex laundering schemes. As depicted in Figure 1, the model comprises three key components. The first one is the *Node Encoder* which generates node embeddings by considering both node and edge features. Then, the *Temporal Encoder* incorporates temporal information to produce time-aware embeddings. Finally, a *Classifier* takes in input the embeddings from the two previous components to determine if a node is engaged in money laundering. The first two components may be used more than once in the network depending on how deep the network should be.

Let us explain the three components and their sampling and aggregation logic.

Node Encoder. This component generates initial node embeddings by processing graph data, including an adjacency matrix, node features, and edge features. The sampling process in the Node Encoder involves collecting messages that consist of node features from connected nodes and edge features from incoming edges. These messages are organized into separate lists for nodes and edges, which are then aggregated. During aggregation, the Node Encoder computes embeddings by averaging the collected messages and applying a dot product with a weight matrix. This process produces embeddings that encapsulate both node and edge information.

Temporal Encoder. The Temporal Encoder introduces temporal dynamics into the node embeddings, crucial for detecting patterns in time-evolving money laundering. The sampling process in the Temporal Encoder groups node and edge messages based on the time steps. These time-step-specific messages are then aggregated by averaging, resulting in a sequence of time-aware aggregated messages. This sequence is fed into a Long Short Term Memory (LSTM) layer, which further aggregates the messages to add temporal knowledge to the embeddings. This approach enables the model to capture complex temporal patterns that are characteristic of sophisticated laundering schemes.

Classifier. The final stage of the model is the classifier, which is a fully connected Neural Network (NN). It receives the concatenated embeddings generated by the Node Encoder and Temporal Encoder, along with any intermediate embeddings preserved through *skip connections* [28]. The classifier then predicts whether each node is involved in money laundering or not. This component synthesizes the comprehensive structural and temporal features captured by the preceding encoders to make accurate classifications.

As the model progresses through multiple layers, it generates several intermediate embeddings. The Node Encoder and Temporal Encoder each produce embeddings, and due to the depth of the network, we use *skip connections* to preserve valuable information. These skip connections concatenate all intermediate embeddings and feed them into the final classifier. Such a technique ensures important features from earlier stages are not lost, improving the accuracy of the final node classification.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP \cdot w_{TP} + FP \cdot w_{FP}) \cdot (TP \cdot w_{TP} + FN \cdot w_{FN}) \cdot (TN \cdot w_{TN} + FP \cdot w_{FP}) \cdot (TN \cdot w_{TN} + FN \cdot w_{FN})}} \tag{1}$$

C. Training Pipeline

Due to the large size of the graph, we employ a custom memory-efficient training algorithm. We first train the LSTM component of the detector (the Temporal Encoder) and then the Node Encoder and the Classifier.

For each epoch, we first compute an initial node embedding using the Node Encoder. These embeddings are then fed into the Temporal Encoder, which generates time-aware node embeddings. To address memory constraints during training, the sampling results from the Temporal Encoder are split into batches. For each batch, we apply the aggregation step and then perform classification using the detector's classifier component. The resulting classification is used to calculate our custom loss function based on the Matthews Correlation Coefficient (MCC) (more details in Subsection V-D), which updates only the LSTM's trainable weights.

After updating the LSTM weights, they are frozen for the remainder of the epoch. We then proceed with a standard forward pass using the input data, during which the loss is computed, gradients are calculated, and model weights are updated, except the frozen LSTM weights. The pseudocode of the training pipeline in Algorithm 1

Algorithm 1 Memory-Efficient Training Algorithm

Input: x (node features), a (adjacency matrix), e (edge features)

Initialize model parameters
for each epoch **do**
  // Phase 1: Train the RNN
  $x\_embed, e\_embed \leftarrow$ NodeEncoder(x, a, e)
  $timed\_nodes \leftarrow$ Sample(x_embed, a, e_embed)
  $batches \leftarrow$ SplitIntoBatches$(timed_nodes)$
  for each batch in batches **do**
    $lstm\_node \leftarrow$ TemporalEncoder$(batch)$
    $classification \leftarrow$ Classify$(lstm_node)$
    $lstm\_loss \leftarrow$ ComputeLoss$(classification)$
    UpdateWeights$(lstm_loss,$ target $=$ 'LSTM')
  end for
  FreezeWeights(target = 'LSTM')
  // Phase 2: Train Embedder and Classifier
  $classification \leftarrow$ ForwardPass(x, a, e)
  $loss \leftarrow$ ComputeLoss$(classification)$
  UpdateWeights$(loss)$
  UnfreezeWeights(target = 'LSTM')
end for

D. MCC Custom Loss Function

We propose a custom training loss to address dataset imbalance without resampling. Specifically, we assign higher weights to minority class samples, ensuring that negative and positive samples contribute equally to loss calculation. Our loss function is based on an adapted version of the Matthews Correlation Coefficient (MCC) metric. Typically, MCC is a balanced metric that evaluates a model's prediction performance, giving equal importance to True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). Our custom loss function weights these metrics to emphasize specific classification tasks and enhance sensitivity to class imbalance during training. These weights, treated as hyperparameters, reduce erroneous classifications.

In summary, we define a weighted version of the original MCC to make the model more focused on incorrect classifications. We define the loss in Equation 1 where w_{metric} represents the weight for each metric.

VI. Experimental Validation

We conducted two experiments to evaluate Amatriciana against baseline models on the HI_SMALL [29] synthetic dataset and demonstrate the enhanced classification ability and reduced False Positives (FPs) due to our improved embedding algorithm and custom loss function.

The first experiment compares Amatriciana with State-of-the-art (SOTA) models. The goal is to demonstrate that with sufficient training data, Amatriciana achieves fewer FPs and better launderer detection than other approaches. In particular, we compare Amatriciana against three baselines: *EvolveGCN*, *GCN with Focal Loss*, and *GraphSAGE*. *EvolveGCN*, similarly to Amatriciana, effectively captures temporal dynamics [18], [21], allowing us to evaluate the detection of complex temporal patterns. *GCN with Focal Loss* [15] addresses the class imbalance in money laundering datasets, allowing us to evaluate the proposed custom loss against the Focal Loss. Finally, *GraphSAGE* [18], serves as a baseline to assess the improvements our extensions bring to the original GraphSAGE algorithm. Except for *EvolveGCN*, where default parameters are used due to their optimal performance, all models are configured comparably to Amatriciana. We use mixed-precision training (float16 for training, float32 for model variables) to accelerate computation and reduce memory usage, while the other models use standard single-precision. Tests confirm that mixed precision does not significantly affect results, so it is used only for Amatriciana due to its memory demands.

The second experiment highlights the effectiveness of our custom loss function in reducing false positives. The goal is to show that the custom loss achieves results comparable to *Cross Entropy* loss, with the benefit of tunable weights to address data imbalance or focus on specific classification tasks.

A. Experimental Settings

IT-AML Dataset Generator. In this work, we use a HI_SMALL [29] dataset generated with the IT-AML dataset

Table I: Amatriciana vs State-of-the-art Models on IT-AML.

	F1	Accuracy	Precision	Recall
evolveGCN	0.6061	0.5487	0.5377	0.6947
GCN (FL)	0.6365	0.5417	0.5274	**0.8028**
GraphSAGE	0.7103	0.6923	0.6710	0.7547
Amatriciana	**0.7600**	**0.7734**	**0.8093**	0.7153

generator [30]. The dataset contains 10 days' worth of transactions, for a total of 5078345 transactions. The number of accounts in the dataset is 515080, of which 511910 are legitimate users and 3170 (0.61%) are launderers. The IT-AML Dataset Generator is a money laundering dataset generator developed by IBM, designed to overcome limitations in previous generators like AMLSim [31]. It uses an agent-based financial simulator to generate complex transaction data, including temporal patterns. The generator provides a list of transactions and corresponding labels, indicating whether each transaction is related to money laundering. The generator tracks laundered funds throughout their lifecycle, from initial placement to eventual use by unwitting non-launderers. Such a level of detail is difficult to achieve with real-world datasets due to the vast number of legitimate and criminal transactions involved. The IT-AML generator simulates transactions across different banks and countries, using multiple currencies and transaction types. Although it does not generate specific account data, it focuses on creating realistic transaction records.

The attributes the generator creates for each transaction are: *Timestamp, From Bank, Account, To Bank, Account.1, Amount Received, Receiving Currency, Amount Paid, Payment Currency, Payment Format, Is Laundering*.

We preprocess data from the IT-AML dataset following the pipeline explained in Subsection V-A. We thereby create a graph and the associated ground truth. In particular, each node is labeled as a launderer or legit account. Using the same split logic explained in Subsection V-A we split the graph to obtain a test graph on which we extract all results we show in this section. Then, the training set is further split into training and validation graphs where the validation graph is used to tune the hyperparameters of our Graph Neural Network (GNN) architecture (details in Subsection V-B).

Performance Metrics. To evaluate and compare the models, we use several key metrics. In particular, we use classification metrics like *Accuracy*, *Precision*, *Recall*, *F1-score*, and the *Receiver Operating Characteristic (ROC) curve* [32].

All metrics are adjusted for class imbalance in the dataset. Additionally, metrics for evolveGCN are scaled by averaging True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) values across test subgraph, then normalizing them to match the adjusted sample sizes used for other models.

B. Comparison with State-of-the-art Models

The dataset used for this experiment spans 10 days and is converted into 384 discrete time steps. We applied early stopping (20-epoch patience) and a learning rate 0.001 to train

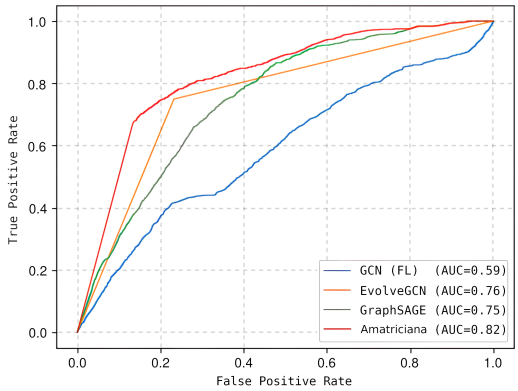


Figure 2: ROC Curves of Amatriciana vs State-of the-art Models on IT-AML Test Dataset.

all models. Our detector uses two `Node Encoder` layers with 32 channels and one `Time Encoder` layer comprising two LSTM units of 32 each. For GraphSAGE and Graph Convolutional Network (GCN) models, we utilized 32 channels.

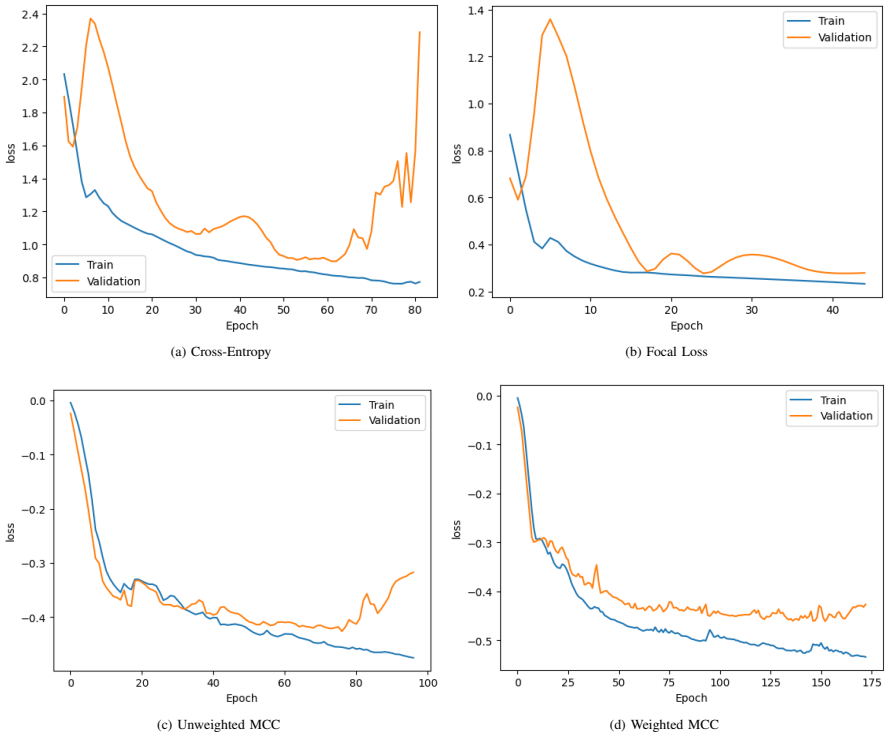
As previously stated, the test graph is disjoint from both the training and validation graphs. For GraphSAGE we follow the same graph splits, despite its adjacency matrix representing the sum of transaction values between nodes. For the GCN model with Focal Loss (FL), we employed a transductive approach. Indeed, we use the entire graph during all training, validation, and testing phases. However, sample weights mask nodes not belonging to the current set, ensuring appropriate training and evaluation. Finally, For EvolveGCN we divide the dataset into time-based subgraphs, following the proportions outlined by the model authors [21].

We conduct model *cross-validation* [33], selecting models with the best F1-score to evaluate on the test set. Table I summarizes the comparative metrics for each model. Notably, the Amatriciana model outperforms others, particularly in Precision. The Precision reflects the proportion of predicted positive samples that are truly positive, helping assess the model's ability to avoid false positives. Thus, the result demonstrates that Amatriciana is the model with fewer FPs.

As shown in Table I, the GraphSAGE baseline provides competitive results but tends to produce more False Positives indicated by the lower precision compared to the Amatriciana detector. Our model performs better by using both relational and temporal information. This is achieved by processing a full graph without creating more subgraphs for each time step, preserving important time-series relationships essential for detecting laundering patterns.

In summary, our approach captures money laundering patterns effectively, requiring only 10 days' worth of transaction data, which aligns with realistic anti-money laundering system constraints. Figure 2 illustrates the superior performance of our model in terms of ROC curves, with the Amatriciana detector demonstrating the highest Area Under the ROC Curve (AUC).

As shown in Table I, the Cross-Entropy and Weighted MCC losses achieve the best results, exhibiting the highest MCC,



(a) Cross-Entropy (b) Focal Loss

(c) Unweighted MCC (d) Weighted MCC

Figure 3: Training and Validation Loss Comparison.

F1-score, and Precision. These models successfully minimize False Positives while maintaining a strong True Positive Rate.

Figure 3 illustrates the training and validation losses for the different models. Both Focal Loss and Cross-Entropy Loss take several epochs to achieve comparable validation and training losses. In contrast, MCC-based losses enable the model to achieve a close match between validation and training losses earlier in the training process, leading to better generalization.

C. Loss Function Evaluation

This experiment demonstrates the potential of custom MCC-based loss functions for learning money laundering patterns in the presence of class imbalance. Furthermore, the flexibility in assigning weight values enables models to be fine-tuned for specific classification tasks, such as minimizing FPs.

This experiment aims to demonstrate that the proposed loss functions can achieve SOTA results while also addressing the challenges of class imbalance. The primary focus is to minimize False Positives. Thus, we compare three loss functions: Cross-Entropy, Focal Loss, and a novel MCC-derived loss. The results indicate that both Cross-Entropy and MCC-based losses perform well.

We experiment with two configurations of MCC-derived loss: one using equal weights (Unweighted MCC) and another that emphasizes False Positives and False Negatives through weight adjustments (Weighted MCC). The latter configuration helps reduce misclassifications. A key advantage of the Weighted MCC-based loss is the ability to adjust weights, allowing the model to focus on specific tasks, such as minimizing FPs.

Table II: Losses Comparison on IT-AML.

	F1	Accuracy	Precision	Recall
Cross-Entropy	0.7638	0.7806	0.8269	0.7097
Focal Loss	0.7185	0.7222	0.7283	0.7090
UW-MCC	0.7627	0.7559	0.7421	**0.7847**
W-MCC	**0.7640**	**0.7819**	**0.8325**	0.7058

Legend: UW-MCC: Unweighted MCC; W-MCC: Weighted MCC.

In this experiment, as in the previous one, we apply early stopping (20-epoch patience) and a learning rate of 0.001 to train all models. Our detector uses two `Node Encoder` layers with 32 channels and one `Time Encoder` layer comprising two LSTM units of 32 each. For GraphSAGE and Graph Convolutional Network (GCN) models, we utilized 32 channels. Furthermore, we conduct model cross-validation, selecting models with the best F1-score for the evaluation.

Table II compares the metrics across the different models trained with various losses. As depicted in the table, the Weighted MCC loss delivers the best performance, although Cross-Entropy also performs well.

VII. Conclusion

To address money laundering, we proposed Amatriciana: A GNN framework augmented by temporal features. It detects launderer accounts in financial transaction networks by reducing the task to node classification. Amatriciana takes

inspiration from GraphSAGE but overcomes its limitations by (1) embedding creation using both node and edge information, and (2) incorporating temporal information with an LSTM to detect patterns in the observed time window. Both these factors enrich the feature representation provided to the model, thereby improving its predictive capabilities. Our detector outperformedSOTA models, raising the AUC to 0.82, and reducing FPs by 55%. Using the IT-AML dataset, we demonstrated robust detection capabilities with fewer FPs than the baselines. We have also shown that our custom loss function matches the performance of existing standard loss functions while empowering the tradeoff between FPs and FNs. A limitation of this work, shared by most of the literature, is the lack of real-world datasets, which hinders a comprehensive evaluation of the model. Experiments were performed on synthetic data instead of actual transactions. Future work should focus on further reducing FPs and enhancing detection through a Human-In-The-Loop (HITL) paradigm. Additionally, we plan to study how timestep discretization influences the results and robustness. Finally, using real-world data will be crucial for achieving reliable evaluations.

ACKNOWLEDGMENT

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

REFERENCES

[1] B. Buchanan, "Money laundering—a global obstacle," *Research in International Business and Finance*, vol. 18, no. 1, pp. 115–127, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0275531904000029

[2] "Payments statistics: 2020," https://www.ecb.europa.eu/press/pr/stats/paysec/html/ecb.pis2020~5d0ea9dfa5.en.html, accessed: 2023-08-07.

[3] N. A. Le Khac and M.-T. Kechadi, "Application of data mining for anti-money laundering detection: A case study," in *2010 IEEE International Conference on Data Mining Workshops*, 2010, pp. 577–584.

[4] C. Lawrencia and W. Ce, "Fraud detection decision support system for indonesian financial institution," in *2019 International Conference on Information Management and Technology (ICIMTech)*, vol. 1, 2019, pp. 389–394.

[5] R. Soltani, U. T. Nguyen, Y. Yang, M. Faghani, A. Yagoub, and A. An, "A new algorithm for money laundering detection based on structural similarity," in *2016 IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2016, pp. 1–7.

[6] S. Yang and L. Wei, "Detecting money laundering using filtering techniques: a multiple-criteria index," *Journal of Economic Policy Reform*, vol. 13, no. 2, pp. 159–178, 2010. [Online]. Available: https://doi.org/10.1080/17487871003700796

[7] E. L. Paula, M. Ladeira, R. N. Carvalho, and T. Marzagão, "Deep learning anomaly detection as support fraud investigation in brazilian exports and anti-money laundering," in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2016, pp. 954–960.

[8] Y. Zhou, X. Wang, J. Zhang, P. Zhang, L. Liu, H. Jin, and H. Jin, "Analyzing and detecting money-laundering accounts in online social networks," *IEEE Network*, vol. 32, no. 3, pp. 115–121, 2018.

[9] Z. Yan and T. Peter, "Machine learning and sampling scheme: An empirical study of money laundering detection," *Computational Economics*, vol. 54, no. 3, pp. 1043–1063, Oct 2019. [Online]. Available: https://doi.org/10.1007/s10614-018-9864-z

[10] M. Jullum, A. Løland, R. B. Huseby, G. Ånonsen, and J. Lorentzen, "Detecting money laundering transactions with machine learning," *Journal of Money Laundering Control*, vol. 23, no. 1, pp. 173–186, Jan 2020. [Online]. Available: https://doi.org/10.1108/JMLC-07-2019-0055

[11] X. Li, X. Cao, X. Qiu, J. Zhao, and J. Zheng, "Intelligent anti-money laundering solution based upon novel community detection in massive transaction networks on spark," in *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, 2017, pp. 176–181.

[12] K. S. and S. K., "Autoregressive-based outlier algorithm to detect money laundering activities," *Journal of Money Laundering Control*, vol. 20, no. 2, pp. 190–202, Jan 2017. [Online]. Available: https://doi.org/10.1108/JMLC-07-2016-0031

[13] D. Labanca, L. Primerano, M. Markland-Montgomery, M. Polino, M. Carminati, and S. Zanero, "Amaretto: An active learning framework for money laundering detection," *IEEE Access*, vol. 10, pp. 41 720–41 739, 2022.

[14] D. Savage, Q. Wang, P. Chou, X. Zhang, and X. Yu, "Detection of money laundering groups using supervised learning in networks," 2016.

[15] P. Humranan and S. Supratid, "A study on gcn using focal loss on class-imbalanced bitcoin transaction for anti-money laundering detection," in *2023 International Electrical Engineering Congress (iEECON)*, 2023, pp. 101–104.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[17] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson, "Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics," 2019.

[18] M. R. Karim, F. Hermsen, S. A. Chala, P. de Perthuis, and A. Mandal, "Catch me if you can: Semi-supervised graph learning for spotting money laundering," 2023.

[19] M. Cardoso, P. Saleiro, and P. Bizarro, "Laundrograph: Self-supervised graph representation learning for anti-money laundering," 2022.

[20] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[21] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," 2019.

[22] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[23] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[24] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[26] M. Weber, J. Chen, T. Suzumura, A. Pareja, T. Ma, H. Kanezashi, T. Kaler, C. E. Leiserson, and T. B. Schardl, "Scalable graph learning for anti-money laundering: A first look," 2018.

[27] L. P. Sergey Brin, "The anatomy of a large-scale hypertextual web search engine," 1999.

[28] K. Xu, M. Zhang, S. Jegelka, and K. Kawaguchi, "Optimization of graph neural networks: Implicit acceleration by skip connections and more depth," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 11 592–11 602. [Online]. Available: https://proceedings.mlr.press/v139/xu21k.html

[29] "It-aml dataset, kaggle url," https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml, accessed: 2023-08-07.

[30] E. Altman, B. Egressy, J. Blanuša, and K. Atasu, "Realistic synthetic financial transactions for anti-money laundering models," 2023.

[31] T. Suzumura and H. Kanezashi, "Anti-Money Laundering Datasets: InPlusLab anti-money laundering datadatasets," http://github.com/IBM/AMLSim/, 2021.

[32] M. J. Zaki and W. Meira Jr, *Data mining and machine learning: fundamental concepts and algorithms*. Cambridge University Press, 2020.

[33] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 1137–1143.