Adaptive Graph Neural Networks for Real-Time Financial Fraud Detection

London School of Economics and Political Science
Candidate Numbers: 37939, 39180, 48099, 48397

May 2025

Abstract

Financial fraud detection requires models that can capture hidden patterns and evolving relationships in transaction data. This study explores the application of Graph Neural Networks (GNNs) for detecting fraudulent financial transactions by modeling transaction networks' structural and dynamic properties. We evaluate static models (GCN, GAT), temporal GNNs (T-GCN), and heterogeneous GNNs (HeteroGNN with SAGEConv) to assess the benefits of time-aware and multi-relational modeling. Additionally, we frame fraud detection as a link prediction task to flag suspicious connections preemptively and apply graph clustering to identify coordinated fraud rings. Our experiments show that graph-based methods outperform traditional baselines, demonstrating their suitability for real-time fraud detection in financial systems.

Introduction

Financial fraud poses a major threat to global economies, with the United Kingdom alone reporting £1.17 billion in losses and nearly 3 million confirmed fraud cases in 2023 (UK Finance, 2024). As digital transactions grow in volume and complexity, so do the techniques used by fraudsters to exploit weaknesses in financial systems.

Traditional fraud detection systems often rely on machine learning models that treat transactions as isolated events, failing to capture the underlying relationships between entities. These models struggle to adapt to evolving fraud patterns and often lack the context required for early detection.

Graph Neural Networks (GNNs) offer a compelling alternative by modeling transaction data as graphs, where nodes represent entities and edges represent interactions. GNNs can learn from the structure and connectivity of the network, enabling more robust detection of anomalous behavior. This research explores the application of GNNs for financial fraud detection, with two core objectives: (1) enhancing fraud classification by modeling relational and temporal structures, and (2) anticipating emerging threats by uncovering suspicious links and fraud rings.

We present a comprehensive pipeline evaluating four GNN strategies: static models like GCN and GAT, temporal GNNs (T-GCN), heterogeneous GNNs that model inter-entity links, and unsupervised methods like link prediction and graph clustering. These models are benchmarked against traditional baselines to assess their effectiveness across different fraud detection tasks.

Literature Review

GNNs have gained traction in fraud detection for their ability to model relational structure, unlike traditional classifiers, which treat transactions as isolated rows.

Temporal GNNs (T-GCNs) extend traditional GCNs by integrating time-dependent information into the message-passing framework. Rossi et al. introduced Temporal Graph Networks (TGNs), a generic framework that uses memory modules and

time-encoding functions to maintain historical context for each node. This allows the model to learn evolving representations for entities over time, which is critical in dynamic fraud environments. Their work shows that temporal models outperform static GNNs on tasks like link prediction and node classification in dynamic graphs. (Rossi et al., 2020)

NVIDIA's fraud detection pipeline demonstrates how heterogeneous graphs, containing multiple node and edge types (e.g., users, devices, IPs), improve fraud classification by incorporating more contextual information. Their end-to-end system constructs a dynamic heterogeneous graph from transaction logs and trains a GNN model to detect both individual frauds and organized fraud rings. (NVIDIA, 2020)

Bukhori and Munir compare inductive GNN variants—GCN, GAT, and GraphSAGE—for link prediction tasks, including on fraud-like data. Their results show that GraphSAGE achieves the highest AUC due to its neighbor sampling mechanism, while GCN performs slightly worse but with significantly faster inference time. This highlights a tradeoff between performance and efficiency relevant for real-time fraud systems. (Bukhori and Munir, 2023)

Pathan and Shrivastava apply GCNs to detect fraud rings in account takeover scenarios. They leverage edge weights and node features to learn similarity-based embeddings, which are used to flag anomalous connections. The GCN approach significantly outperforms XGBoost and Louvain community detection, demonstrating GNNs' ability to generalize from limited fraud examples. (Pathan and Shrivastava, 2021)

Despite these advancements, most existing work focuses on one modeling aspect such as temporal, relational, or structural. Our work bridges these gaps by building a unified framework that compares and integrates multiple GNN strategies: temporal modeling via T-GCN, relational learning via HeteroGNN, structural reasoning via link prediction, and community detection via unsupervised clustering. By combining temporal, relational, and structural reasoning in one pipeline, our approach addresses the limitations of static or homogeneous methods while enabling real-time detection and proactive threat identification.

Research Questions

1. How do Temporal Graph Convolutional Networks (T-GCN) compare to static GNNs in terms of precision, recall, and inference latency?

2. Does modeling inter-entity relationships enhance fraud detection compared to treating transactions as independent events?

3. Can link prediction via GCNs be used to flag future suspicious connections before fraud occurs?

4. Can graph clustering help uncover hidden fraud rings by identifying behaviorally similar communities?

Problem Formation

Key Definitions

Graph: A *graph* $G = (V, E)$ consists of nodes V and edges E, where each edge $(u, v) \in E$ connects two nodes (Diestel, 2017).

Homogeneous Graph : A graph in which all nodes and all edges are of the same type (Blog, 2022).

Multi-Relational Graph: A graph $G = (V, E)$ is multi-relational when the edges are defined as tuples $e = (u, t, v)$, indicating that a particular relation $t \in T$ holds between two nodes(Hamilton, 2017).

Heterogeneous Graphs: A subset of multi-relational graphs, where nodes have different types, meaning the set of nodes can be partitioned into disjoint sets - $V = V_1 \cup V_2 \cup ... \cup V_k$ where $V_i \cap V_j = \emptyset, \forall i \neq j$ (Hamilton, 2017).

Heterogeneous Information Network (HIN): A HIN is a type of graph that contains multiple types of nodes and/or edges, each representing different entity types and relationships.(Sun et al., 2011).

Inter-Entity Relationships: Inter-entity relationships describe the connections or interactions between different entities within a system.(Ying et al., 2018).

Graph Neural Network (GNN): GNNs are deep neural networks on graph data. The key idea is to generate representations of nodes that depend on the

structure of the graph, as well as any feature information. For a given graph $G = (V, E)$ and some features $(X_v)v \in V$, we define the hidden embeddings as: $h_u^k = UPDATE(h_u^{(k-1)}, AGG(h_v^{(k-1)}; v \in N(u))$ (Gilmer et al., 2017).

Graph Convolutional Network (GCN): GCNs are a subset of GNNs that employ the symmetric normalized aggregation, as well as a self-loop update approach to learn node representations that incorporate both the features of the nodes and the structure of the graph (Kipf and Welling, 2017).

Graph Attention Network (GAT): GATs are neural networks that compute node representations using masked self-attentional layers, allowing nodes to weight the importance of their neighbors with different weights(Veličković et al., 2018a).

Temporal Graph Convolutional Network (T-GCNs): T-GCNs is a model that combines GCNs with temporal modeling to capture spatial and temporal dependencies (Zhao et al., 2019).

Adam Optimizer: Adam is an adaptive learning rate optimization algorithm that combines the advantages of AdaGrad and RMSProp, using estimates of the first and second moments of gradients (Kingma and Ba, 2015).

Link Prediction: Link prediction is estimating the likelihood that a link exists or will exist between two nodes $u, v \in V$ where $(u, v) \notin E$. (Liben-Nowell and Kleinberg, 2007).

Graph Clustering: Graph clustering groups nodes or subgraphs based on similarity, usually in an unsupervised manner. (Schaeffer, 2007).

Dataset

To analyze GNNs for financial fraud detection, we utilize the publicly available IEEE-CIS Fraud Detection dataset. The dataset has two files. The first file, "Identity", includes device type, device info, and 38 anonymized identity features. The second file, "Transaction", includes the transaction date time, amount, payment card information, address information, match information between transactions, and vesta engineered rich features.

We joined these two datasets on "transactionID", a unique identifier, resulting in a merged dataset with 590,540 rows and 434 columns. The dataset has a binary indicator "isFraud" that determines which transactions are fraudulent. This dataset is highly imbalanced, with only 3.5% of the transactions labeled as fraudulent.

Problem Setup

RQ1: Temporal and Static Graph Setup

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a transaction graph where each node $u \in \mathcal{V}$ represents a financial transaction, and each edge $(u, v) \in \mathcal{E}$ indicates a relationship between transactions. Each node u is associated with a feature vector $\mathbf{h}_u^{(0)} \in R^{15}$ consisting of:

-13 card-related features: $C1, \ldots, C13$

-The transaction amount: TransactionAmt

-The time delta from a reference point: TransactionDT

The task is binary classification: predict whether a transaction is fraudulent (isFraud $= 1$) or not (isFraud $= 0$), using only these 15 features.

We consider two types of graph models: static and temporal.

Static GCN (Kipf and Welling, 2017): A Graph Convolutional Network (GCN) with symmetric normalization performs message passing using the rule:

$$\mathbf{h}_u^{(k)} = \sigma\left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{1}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|}} \mathbf{W}^{(k)} \mathbf{h}_v^{(k-1)}\right),$$

where $\mathbf{W}^{(k)}$ is a trainable weight matrix at layer k, and σ is a non-linear activation function (ReLU).

Static GAT (Veličković et al., 2018b): The message-passing mechanism for GAT is defined as:

$$\mathbf{h}_u^{(k)} = \sigma\left(\sum_{v \in \mathcal{N}(u)} \alpha_{uv}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_v^{(k-1)}\right),$$

where the attention coefficients $\alpha_{uv}^{(k)}$ are computed using a shared attention mechanism:

$$\alpha_{uv}^{(k)} = \frac{\exp\left(\text{ReLU}\left(\mathbf{a}^\top \left[\mathbf{W}^{(k)}\mathbf{h}_u^{(k-1)} \parallel \mathbf{W}^{(k)}\mathbf{h}_v^{(k-1)}\right]\right)\right)}{\sum_{w\in\mathcal{N}(u)} \exp\left(\text{ReLU}\left(\mathbf{a}^\top \left[\mathbf{W}^{(k)}\mathbf{h}_u^{(k-1)} \parallel \mathbf{W}^{(k)}\mathbf{h}_w^{(k-1)}\right]\right)\right)}$$

Temporal Graph Networks (TGN) (Rossi et al., 2020): Let $\{\mathcal{G}_t\}_{t=1}^T$ be a sequence of graphs evolving over discrete time steps $t = 1, \ldots, T$, where each graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t)$ shares a common node set $\mathcal{V}$ but may have time-varying edge sets $\mathcal{E}_t$ and node features. Each node $u \in \mathcal{V}$ has a time-dependent feature vector $\mathbf{h}_u^{(t)} \in R^d$. A Temporal Graph Convolutional Network (T-GCN) extends static Graph Convolutional Networks (GCNs) by incorporating temporal dependencies, here we can see how we update the message and the embedding if the nodes.

$$\mathbf{m}_u^{(t,k)} = \sum_{v\in\mathcal{N}(u)\cup\{u\}} \frac{1}{\sqrt{|\mathcal{N}(u)|\cdot|\mathcal{N}(v)|}}\mathbf{W}^{(k)}\mathbf{h}_v^{(t,k-1)},$$

$$\mathbf{h}_u^{(t,k)} = \text{GRU}\left(\mathbf{h}_u^{(t-1,k)}, \sigma(\mathbf{m}_u^{(t,k)})\right),$$

where $\mathbf{m}_u^{(t,k)}$ is the message for node u at time t in the k-th layer of the T-GCN, computed as a weighted sum of neighboring node features $\mathbf{h}_v^{(t,k-1)}$. The GRU(Gated Recurrent Unit) updates this feature vector by processing the messages it receives from its neighbors and from the past state (memory).

In the case of a Temporal Graph Network (TGN), the memory of each node is initialized as a zero vector at the beginning, and it is updated upon each event involving the node, even after the model has finished training. The memory update occurs as follows:

$$\mathbf{m}_i(t) = \text{msg}_{s_i(t^-)}(s_i(t^-), s_j(t^-), \Delta t, e_{ij}(t)),$$

$$\mathbf{m}_j(t) = \text{msg}_{s_j(t^-)}(s_j(t^-), s_i(t^-), \Delta t, e_{ij}(t)),$$

where $e_{ij}(t)$ represents the interaction event between nodes i and j at time t, Δt is the time difference from the last interaction, and $s_i(t^-)$ and $s_j(t^-)$ are the memories of nodes i and j at the previous time step. The message functions msg are learnable components.

When there are multiple events for the same node within a time window, an aggregation function agg is used to combine these messages into a single message $\overline{m}_i(t)$:

$$\overline{m}_i(t) = \text{agg}(m_i(t_1), \ldots, m_i(t_b)),$$

where agg is the average of the messages across the batch.

Finally, the memory of node i is updated using:

$$s_i(t) = \text{mem}(\overline{m}_i(t), s_i(t^-)),$$

where mem represents a memory update function, which is a recurrent neural network - GRU.

To avoid memory staleness, where a node's memory becomes outdated due to inactivity, an embedding module is introduced to generate the temporal embedding $z_i(t)$ for node i:

$$z_i(t) = \text{emb}(i,t) = h\left(s_i(t), s_j(t), e_{ij}, v_i(t), v_j(t)\right),$$

where h is a learnable function that computes the embedding based on the node's memory and its interactions with neighbors.

The T-GCN can thus model long-term dependencies for each node by updating its memory and generating embeddings at each time step.

Objective: Given the three architectures (GCN, GAT, T-GCN), we aim to compare their performance on a temporal node classification task. Since fraud represents the minority class, we evaluate models using the following criteria:

- F1-score : to measure the balance between precision and recall, particularly important for imbalanced datasets.
- Training and inference time : to assess computational efficiency
- Precision-Recall (PR) Curve : to evaluate the model's ability to distinguish between fraudulent and legitimate transactions under varying thresholds.

RQ2: Inter-Entity Relationship Modeling Setup

(Wang et al., 2019)

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}_V, \mathcal{T}_E)$ be a heterogeneous graph where:

- $\mathcal{V} = \mathcal{V}_{txn} \cup \mathcal{V}_{card} \cup \mathcal{V}_{addr} \cup \mathcal{V}_{device}$ is the union of node sets corresponding to transactions, cards, addresses, and devices.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of typed edges connecting transactions to entities (e.g., $(txn, card)$).
- $\mathcal{T}_V$ and $\mathcal{T}_E$ represent the sets of node and edge types respectively.

Each transaction node $v \in \mathcal{V}_{txn}$ is associated with a label $y_v \in \{0, 1\}$ indicating whether the transaction is fraudulent. We define a binary classification task where the goal is to predict y_v using relational context.

We compare two approaches:

1. **Independent Baseline:** XGBoost is trained on transaction-level tabular features without considering relationships.
2. **Relational Modeling:** A Heterogeneous Graph Neural Network (HeteroGNN) is applied to model multi-typed edges between transactions and associated entities (cards, addresses, devices).

We use `SAGEConv` layers within a `HeteroConv` architecture to propagate information across different edge types. Each node type $t \in \mathcal{T}_V$ has its own embedding matrix $\mathbf{H}_t^{(0)} \in R^{|\mathcal{V}_t| \times d}$ initialized randomly.

At each message-passing layer l, we update node embeddings using:

$$\mathbf{h}_t^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}_t} \text{AGG}_r\left(\left\{\mathbf{W}_r \cdot \mathbf{h}_s^{(l)} \mid s \in \mathcal{N}_r(t)\right\}\right)\right)$$

Where:

- $\mathcal{N}_r(t)$ is the set of neighbors of node t under relation r
- $\mathbf{W}_r$ is a relation-specific transformation matrix
- AGG_r is a mean aggregator for relation r
- σ is the ReLU activation function

After two convolution layers, the learned embeddings $\mathbf{h}_{txn}^{(2)}$ for transaction nodes are passed through a softmax classifier to obtain fraud probabilities:

$$\hat{y}_v = \text{Softmax}(\mathbf{W}_{out} \cdot \mathbf{h}_v^{(2)})$$

Objective: We evaluate whether adding relational information through typed edges improves detection performance by comparing metrics such as precision, recall, and F1-score for fraud transactions across our 3 different models.

RQ3: Link Prediction Setup

(Zhang and Chen, 2018) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a transaction graph where each node $u \in \mathcal{V}$ represents a financial transaction, and each edge $(u, v) \in \mathcal{E}$ indicates a shared card or device. We employ a two-layer GCN for link prediction. The encoder computes node embeddings via:

$$H^{(1)} = \text{ReLU}\left(\tilde{A}XW^{(0)}\right), \quad Z = \tilde{A}H^{(1)}W^{(1)}$$

where:

- $X \in R^{|\mathcal{V}| \times d}$ is the input feature matrix
- $W^{(0)} \in R^{d \times h}$, $W^{(1)} \in R^{h \times h}$ are trainable weights
- $Z \in R^{|\mathcal{V}| \times h}$ is the final node embedding matrix

The decoder computes all the link prediction scores:

$$\text{score}(u, v) = z_u^\top z_v = \sum_{i=1}^{h} z_u^{(i)} \cdot z_v^{(i)}$$

The model is trained using a binary cross-entropy loss that encourages high scores for positive edges and low scores for negative edges. Let:

- $\mathcal{E}^+ \subseteq V \times V$: the set of positive (observed) edges
- $\mathcal{E}^- \subseteq V \times V$: the set of negative (non-existent or sampled) edges

- $\sigma(\cdot)$: the sigmoid function

The total loss is defined as:

$$\mathcal{L} = -\frac{1}{|\mathcal{E}^+|}\sum_{(u,v)\in\mathcal{E}^+}\log\sigma(z_u^\top z_v) - \frac{1}{|\mathcal{E}^-|}\sum_{(u,v)\in\mathcal{E}^-}\log\left(1-\sigma(z_u^\top z_v)\right)$$

Objective: After training the model, the goal is to compute a similarity score between each node $v \in V$ and a set of known fraudulent nodes $\mathcal{F} \subset V$. For each node v, we define the fraud similarity score as the maximum inner product between v's embedding and the embeddings of known fraudulent nodes:

$$s_v = \max_{f\in\mathcal{F}} z_v^\top z_f$$

This score is used to rank the nodes according to their proximity to known fraudulent activity in the learned embedding space.

Performance is measured using Area Under the ROC Curve (AUC), defined as:

$$\text{AUC} = \Pr\left(s_{(u,v)\in\mathcal{E}^+} > s_{(u',v')\in\mathcal{E}^-}\right)$$

RQ4: Graph Clustering Setup

(**wang2017structural**; Tsitsulin et al., 2020) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a transaction graph where each node $u \in \mathcal{V}$ represents a financial transaction, and each edge $(u, v) \in \mathcal{E}$ indicates a relationship between transactions that share cards or devices. We first extract the largest connected component $G' = (V', E') \subseteq G$ where:

$$V' = \arg\max_{C\subseteq V}|C| \quad \text{such that } C \text{ is connected}$$

We compute node embeddings $\mathbf{X} \in R^{|V'|\times d}$ for the subgraph G', using either:

- **Node2Vec**: a biased random walk method that generates embeddings via the skip-gram model:

  $$\text{Node2Vec}(G', p, q) \to \mathbf{X}$$

  where p and q control the walk bias.

- **Spectral Embedding**: using the eigenvectors of the normalized Laplacian $\mathbf{L} \in R^{|V'|\times|V'|}$:

  $$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2},$$
  $$\mathbf{X} = \text{top-}d \text{ eigenvectors of } \mathbf{L}$$

Objective: Given embeddings $\mathbf{X}$, the goal is to apply unsupervised clustering to assign labels $y_i \in Z$ to each node $i \in V'$:

$$y = \text{Cluster}(\mathbf{X})$$

where $\text{Cluster}(\cdot) \in \{\text{KMeans}, \text{DBSCAN}\}$ is a clustering algorithm.

Let $\mathcal{C}_k = \{v_i \in V' \mid y_i = k\}$ be the k-th cluster. For each cluster $\mathcal{C}_k$, we extract the corresponding set of transactions:

$$\mathcal{T}_k = \{t_i \in \mathcal{C}_k \cap \text{Transactions}\}$$

Let $\text{isFraud}(t) \in \{0, 1\}$ be a fraud label for transaction $t \in \mathcal{T}_k$. Then, the fraud rate of cluster k is:

$$\text{FraudRate}_k = \frac{1}{|\mathcal{T}_k|}\sum_{t\in\mathcal{T}_k}\text{isFraud}(t)$$

Then, our objective is to flag $\mathcal{C}_k$ as a potential fraud ring if:

$$\text{FraudRate}_k > \theta \quad \text{and} \quad |\mathcal{T}_k| \geq \tau$$

for some thresholds $\theta \in [0, 1]$, $\tau \in N$, e.g., $\theta = 0.5$, $\tau = 3$.

Proposed Solutions

RQ1: Temporal and Static Graph Solution

To evaluate whether an Adaptive GNN can effectively capture the temporal structure and dynamics of transaction networks for fraud detection, we first implemented two baseline static GNN models: a GCN and a GAT. These models serve as benchmarks to assess the added value of incorporating temporal information. We then compared their performance to a T-GCN, which explicitly models the evolution of the transaction graph over time. Moreover, we used different data splitting strategies appropriate to each

model type. For the static models (GCN and GAT), we applied a random split of the dataset into training, validation, and test sets. In contrast, for the temporal model (T-GCN), we adopted a temporal snapshot-based split, ensuring that future information is not leaked into the past. We chose Adam optimizer for our different GNNs because it is memory-efficient and adapts the learning rate dynamically for each parameter, following guidance from prior work Rossi et al., 2020.

In the context of graph-based models, hyperparameters play a crucial role in model performance. These parameters are important because:

- Optimal number of neighbors: strikes a balance between capturing local features and maintaining computational efficiency. We tried with 5 and 10 neighbors.

- Number of hidden channels: a key factor in determining the model's capacity to learn complex node representations. We chose to try 16 and 8 to see the difference.

- Learning rate: controls the step size at each iteration of the optimization process. We tried 0.01, 0.001 and 0.0001 on Adam optimizer.

A grid search allows us to explore a wide range of possible configurations and identify the best combination of hyperparameters that minimize the validation loss. After optimization, we compared all three models to assess their effectiveness in detecting fraudulent transactions.

The T-GCN's GRU-based temporal updates introduce an additional $\mathcal{O}(d^2)$ complexity per node compared to the GCN's $\mathcal{O}(|\mathcal{E}|d)$ edge-based operations, trading increased computational cost for richer temporal modeling capabilities.

RQ2: Inter-Entity Relationship Modeling Solution

To investigate whether modeling inter-entity relationships improves fraud detection, we evaluate three approaches: a baseline using XGBoost on transaction-level tabular features, a homogeneous GCN, and a Heterogeneous Graph Neural Network (HeteroGNN) with relation-aware message passing.

The XGBoost classifier serves as a non-relational baseline, treating each transaction as an independent data point. While computationally efficient, this approach fails to capture contextual information, leading to limited ability to detect complex fraud patterns involving reused entities such as cards, devices, and addresses.

We then construct a simple homogeneous graph, connecting transactions to their associated entities. A GCN is trained on this structure without distinguishing node or edge types. While this model improves recall compared to the baseline, it performs poorly in precision, likely due to its inability to differentiate among relation types, resulting in noisy aggregations and overfitting to frequent entities.

To capture richer structural and semantic information, we implement a HeteroGNN using PyTorch Geometric's `HeteroConv` module with `SAGEConv` layers. Each node type has its own embedding space, and message passing is conducted through type-specific aggregation functions. This architecture enables the model to learn from relation-specific interactions, such as repeated use of the same card across multiple fraudulent transactions.

Hyperparameter tuning was conducted across learning rates (0.001, 0.01), hidden dimensions (32, 64), and dropout values. The best results were achieved with a learning rate of 0.001 and 64 hidden units. The model was trained using cross-entropy loss and the Adam optimizer, with early stopping based on validation F1-score.

RQ3: Link Prediction Solution

To assess whether link prediction can flag suspicious connections before fraud occurs, we implement a GCN on a homogeneous transaction graph, where nodes represent transaction and edges represent shared cards or devices. The model encodes node embedding and then decodes them to predict link probabilities.

We construct the graph using PyTorch Geometric, resulting in 590,540 nodes and 89,892 edges. To reduce noise and prevent overfitting, we filtered out cards and devices that appeared in an unusually high number of transactions. We limited the edge set to 50,000 connections for computational efficiency, split-

ting them into 40,000 for training, 5,000 for validation, and 5,000 for testing. Positive examples are the real edges, while an equal number of negative edges were sampled uniformly at random, ensuring they did not overlap with existing edges.

During training, the GCN learns node embeddings layers and uses inner product to predict edge likelihoods. We optimize it with binary cross-entropy loss and sigmoid activation using the Adam optimizer. Performance is evaluated using AUC, Precision, Recall, and F1 Score.

In terms of computational complexity, the encoder is $\mathcal{O}(2 * |E| * d)$, where $|E|$ is the number of edges, d is the dimensionality of the features, and 2 is for the 2 layers. The decode computational complexity is $\mathcal{O}(M * d)$, where M is the number of edges being evaluated. Then, the complexity of the matrix multiplication between transaction embeddings and fraud embeddings is $\mathcal{O}(N * F * d)$, where N is the number of transactions, and F is the number of known fraud examples.

To identify potentially suspicious connections, we use known fraudulent transactions as anchors. We compute dot products between each transaction and the known frauds, rank the results by similarity, and return the top-scoring transactions. These can then be flagged for further investigation, enabling early fraud detection based on structural similarity in the transaction graph.

RQ4: Graph Clustering Solution

To evaluate the potential of graph clustering for detecting fraud rings, we first identify the largest connected components in the transaction graph. This opertation takes $\mathcal{O}(n+m)$ time, where n is the number of nodes, and m is the number of edges. We then generate node embeddings using the Node2Vec algorithm, which leverages random walks to capture structural information and the contextual similarity between nodes. The computational complexity of Node2Vec is $\mathcal{O}(n * \text{number of walks} * \text{length of walk})$.

These embeddings are then clustered using two different algorithms - KMeans, which is partion based and DBSCAN, which is density based. KMeans complexity is $\mathcal{O}(n * d)$, where d is the dimensions in the embedding, and DBSCANs complexity is $\mathcal{O}(n^2)$.

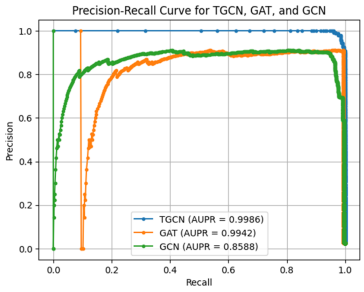

Figure 1: Precision-Recall Curves for T-GCN, GCN, GAT

Each clustering algorithm assigns a label to each node. To assess fraud concentration, we compute the fraud rate within each cluster by referencing the isFraud labels from the original dataset. Clusters exhibiting a high fraud rate (greater than 50 percent) and exhibiting shared attributes such as common cards or devices are flagged as potential fraud rings.

Numerical Experiments

RQ1: Temporal and Static Graph Results

Model	F1 Score	Training Time (s)
T-GCN	**0.8730**	14.24
GAT	0.7970	9.14
GCN	0.4129	**6.81**

Table 1: Performance comparison of T-GCN, GAT, and GCN in terms of F1 Score and training time.

After conducting an extensive grid search, we identified the optimal hyperparameters for each model, focusing on learning rate, number of neighbors, and number of hidden channels. We observed that a learning rate of 0.001 consistently provided the best trade-off between convergence speed and generalization, especially for T-GCN. A learning rate of 0.0001 led to overfitting in all models, as evidenced by the validation loss diverging from the training loss and lower F1 scores. Conversely, a learning rate of 0.01

underperformed in T-GCN, likely due to instability in learning temporal dependencies. For the number of neighbors, a value of 5 was more effective than 10 across all models, suggesting that including too many neighbors may dilute meaningful local patterns with noise. Regarding model capacity, increasing the number of hidden channels to 16 significantly improved performance, as it enabled the networks to learn more expressive node representations, especially crucial in high-dimensional, noisy fraud detection settings.

Table 1 and Figure 1 summarize the performance of the final models. T-GCN achieves the highest F1 score (0.8730) and PR-Curve, indicating strong performance in distinguishing between fraudulent and legitimate transactions. GAT performs reasonably well (F1 Score: 0.7970), leveraging attention mechanisms to weigh neighbor importance which helps to focus on more relevant interactions, but it lacks an explicit temporal modeling component, which may limit its ability to detect time-dependent fraud signals. The GCN, while it is the fastest to train (6.81s), shows the weakest performance (F1 Score: 0.4129), as it treats the graph as static and does not account for the temporal progression of transactions — an essential factor in fraud detection.

In conclusion, while GCN is computationally efficient, its lack of temporal awareness significantly reduces its effectiveness. GAT offers a good trade-off but still falls short of fully capturing temporal patterns. T-GCN, despite being slightly more computationally expensive, proves to be the most effective model, making it the ideal choice for tasks where accuracy and temporal sensitivity are crucial, such as real-time fraud detection in dynamic financial environments.

RQ2: Inter-Entity Relationship Modeling Results

We evaluated three approaches to assess the impact of modeling inter-entity relationships in fraud detection: a baseline XGBoost classifier, a homogeneous GCN, and a Heterogeneous GNN with relation-aware message passing.

Baseline (XGBoost): The model was trained on flattened tabular transaction features without any relational context. It achieved high accuracy due to class imbalance but failed to identify fraudulent transactions effectively, yielding poor recall and F1 score.

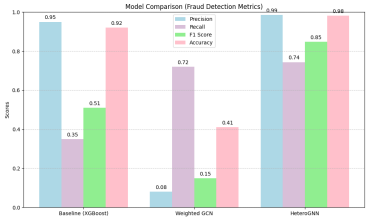

Figure 2: Comparison of Precision, Recall, F1 Score, and Accuracy across models for RQ2.

GCN (Homogeneous): A simple transaction-entity graph was constructed with transaction nodes linked to cards, addresses, and device nodes. This approach showed improvement in recall but suffered from low precision and overfitting, as the GCN was unable to distinguish node types.

HeteroGNN (SAGEConv): The best performance was observed when node types and edge semantics were explicitly modeled using PyTorch Geometric's `HeteroConv` with SAGEConv layers. The model achieved significant gains across all metrics, particularly in F1 score and fraud-class precision.

These results demonstrate that modeling inter-entity relationships (via a heterogeneous graph) leads to substantial improvements in identifying fraud. GCNs offered better recall than the baseline but lacked discriminatory power, whereas the HeteroGNN balanced high recall with precision, indicating strong generalization to minority fraud cases.

The final bar chart (Figure 2) visually summarizes the performance gap, showing that HeteroGNN significantly outperforms the other two models in detecting fraud. This supports our hypothesis that inter-entity relationships, when properly encoded, can drastically improve fraud detection in financial transaction networks.

RQ3: Link Prediction Results

To assess the effectiveness of our link prediction model for fraud detection, we monitored training dynamics and evaluated final performance on the test set. The training process showed improvement in AUC, indicating that the models ability to distinguish between positive and negative links. Early epochs demonstrated high recall but low precision, suggesting that the model was over-flagging edges as fraudulent. This behavior can often be attributed to the imbalance in the graph, where fraudulent edges are much less frequent.

As training progressed, the model's precision improved while recall slightly decreased, leading to a better overall balance in the later epochs. The final F1 score reached 0.6592, reflecting a reasonable trade-off between precision and recall.

We also evaluated the model's performance on the test set, as summarized in Table 2. The AUC of 0.7006 indicates that the model is moderately effective at ranking true fraudulent links above non-fraudulent ones.

In conclusion, although our model performs well in identifying suspicious edges, there is room for improvement in recall, which could be addressed by leveraging more sophisticated sampling techniques or by incorporating additional temporal and contextual features that capture evolving fraud patterns over time. Nonetheless, this method provides a useful tool for early fraud detection based on structural graph similarity.

Metric	Value
AUC	0.7006
Precision	0.6436
Recall	0.6756
F1 Score	0.6592

Table 2: Performance metrics for GCN-based link prediction model on the test set.

RQ4: Graph Clustering Results

To investigate whether graph clustering can reveal fraud rings, we analyzed the structural communities within the transaction network. Upon analyzing the largest connected component of the graph (5,068 nodes), we applied KMeans clustering to the node embeddings. KMeans identified 20 clusters, as shown in Figure 3, of which 7 clusters were flagged as potential fraud rings based on the following characteristics: high fraud rates (greater than 50%), shared devices labeled as "unknown", and reused cards across multiple transactions.

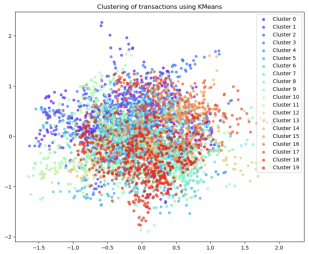

Figure 3: Graph Clustering Using KMeans

These flagged clusters are strongly indicative of coordinated fraud activity and warrant further investigation. In contrast, DBSCAN did not detect any meaningful clusters, suggesting that the density-based approach was less effective for our graph's structure. The lack of dense regions in the data highlights that fraud rings may not always exhibit high local density but rather form through relational patterns that are more effectively captured by KMeans.

In summary, the KMeans clustering algorithm proved effective in identifying fraud rings within the transaction network. The identification of shared cards, devices, and connected communities across clusters was instrumental in uncovering potential fraudulent activity. These findings suggest that graph clustering is a valuable technique for fraud detection, especially when combined with other graph-based features.

Conclusion

This paper investigates graph-based models for fraud detection in dynamic financial environments. Our experiments demonstrate that integrating temporal (T-GCN), relational (HeteroGNN), and structural (clus-

tering) modeling outperforms isolated methods, offering a more holistic solution for financial fraud detection. This synergy enables systems to not only classify fraudulent transactions but also anticipate emerging threats through link prediction and community detection..

While the results are promising, there are several avenues for future work: enhancing temporal modeling, exploring advanced sampling techniques, and combining graph-based methods with other model architectures for improved performance. Real-world evaluation and efforts to increase model interpretability will also be crucial for deploying these methods in fraud detection systems. Also, limitations include the computational overhead of T-GCNs on large graphs and the need for interpretable explanations in high-stakes fraud cases. Future work will explore scalable temporal architectures, hybrid graph-RL models, and real-world deployment with regulatory constraints. By bridging temporal, relational, and structural gaps, this work lays a foundation for next-generation fraud detection systems.

References

Liben-Nowell, D., & Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American society for information science and technology*, *58*(7), 1019–1031.

Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, *1*(1), 27–64.

Sun, Y., Han, J., Yan, X., Yu, P. S., & Wu, T. (2011). Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, *4*(11), 992–1003.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1412.6980

Diestel, R. (2017). *Graph theory* (5th). Springer.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. *Proceedings of the 34th International Conference on Machine Learning*, 1263–1272.

Hamilton, W. L. (2017). Graph representation learning [Accessed: 2025-05-05]. https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_4-Knowledge_Graphs.pdf

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018a). Graph attention networks. *International Conference on Learning Representations (ICLR)*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018b). Graph attention networks. *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1710.10903

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 974–983.

Zhang, M., & Chen, Y. (2018). Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 5165–5175.

Wang, D., Lin, J., Cui, P., Jia, Q., Wang, Z., Fang, Y., Yu, Q., Zhou, J., Yang, S., & Qi, Y. (2019). Heterogeneous graph attention networks for fraud detection. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2021–2030.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., & Lin, Y. (2019). T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, *21*(9), 3848–3858.

NVIDIA. (2020). Dynamic graph neural networks for fraud detection [Accessed: 2025-05-05].

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2020). Temporal

graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*. https://arxiv.org/abs/2006.10637

Tsitsulin, A., Palowitch, J., Perozzi, B., & Müller, E. (2020). Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*.

Pathan, S., & Shrivastava, V. (2021). Identifying linked fraudulent activities using graphconvolution network. *arXiv preprint arXiv:2106.04513*.

Blog, C. (2022, April). Knowledge graphs 101: How nodes and edges connect all the world's real estate data [Accessed: 2025-05-05]. https://blog.cherre.com/2022/04/08/knowledge-graphs-101-how-nodes-and-edges-connect-all-the-worlds-real-estate-data

Bukhori, H. A., & Munir, R. (2023). Inductive link prediction banking fraud detection system using homogeneous graph-based machine learning model. *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, 0246–0251.

UK Finance. (2024). *Annual fraud report 2024* [Accessed: 2025-05-05]. https://www.ukfinance.org.uk/policy-and-guidance/reports-and-publications/annual-fraud-report-2024

A Dataset

Link to the Kaggle IEEE Fraud Detection Dataset: https://www.kaggle.com/competitions/ieee-fraud-detection