Chapter 3

Methodology: Heterophily-aware Temporal Multi-View GNN for Fraud Detection

3.1 Introduction

3.2 Preliminaries

3.2.1 Fraud Behavior Analysis

3.2.2 Problem Formulation

We model the system as a heterogeneous temporal graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T}),$$

where $\mathcal{V}$ is the set of nodes (accounts, devices, IPs, cards or merchants), $\mathcal{R}$ is the set of relation types (login, transaction, device-association, transfer, role-change), and each edge $e = (v, u, r, t_{vu}, x_{vu}^{(r)}) \in \mathcal{E}$ connects node v to node u under relation r, occurs at timestamp $t_{vu} \in \mathcal{T}$ and carries edge attributes $x_{vu}^{(r)}$. Each node u has an attribute vector x_u. A subset $\mathcal{V}_L \subset \mathcal{V}$ is labeled, with binary labels $y_u \in \{0, 1\}$ indicating benign (0) or fraudulent (1).

The objective is node classification: learn a function $\hat{y}_u = f_\theta(u; \mathcal{G})$ that predicts y_u.

3.3 Proposed Methodology

This section presents the proposed model for fraud detection using GNN. The architecture is specifically designed to capture the unique behaviors of fraudulent entities such as

resource reuse, role switching, temporal bursts, and camouflage (heterophily). The model is built around four key components: (i) input encoding, (ii) a heterophily-aware temporal multi-view message passing framework, (iii) classification and loss, and (iv) training and inference procedure. An overview of the architecture is illustrated in Figure **??**.

3.3.1 Input Encoding

Fraud detection graphs are naturally heterogeneous, as they contain multiple types of nodes (such as users, devices, IP addresses, merchants) and multiple types of edges (login, transaction, registration). To account for this, we design a **typed attribute encoder** that generates initial embeddings for each node type. To capture the temporal dynamics, we also integrate a **temporal encoder** for the timestamps within edge features.

Typed Attribute Encoding

Let $\mathcal{V}$ denote the set of nodes and $\mathcal{T}$ the set of node types. For a node $v \in \mathcal{V}$ with type $\tau(v) \in \mathcal{T}$ and raw feature vector $\mathbf{x}_v$, its initial embedding is given by:

$$\mathbf{h}_v = MLP_{\tau(v)}(\mathbf{x}_v), \tag{3.1}$$

We use a Multi-Layer Perceptron encoder to map the input features into a shared embedding space. This ensures that heterogeneous node features are projected into a common vector space while preserving type-specific semantics.

Temporal Encoding

Fraudulent activities are inherently time-dependent (staged warm-up periods, suspicious unfrequented hours or sudden attack bursts). To capture temporal patterns, each edge (u, v, t) is associated with a timestamp t. We adopt a dual encoding strategy:

1. **Temporal Positional Encoding:** Inspired by the work proposed by Tian and Liu [45] we map each timestamp t into a periodic vector:

$$\mathbf{p}(t)_{2k} = \sin\left(\frac{t}{10000^{2k/d_t}}\right), \quad \mathbf{p}(t)_{2k+1} = \cos\left(\frac{t}{10000^{2k+1/d_t}}\right), \tag{3.2}$$

where d_t is the dimensionality of the temporal embedding. This encoding allows the model to capture relative ordering and periodic events without heavy computation.

2. **Learnable Time Embedding:**

Each timestamp is also passed through a small neural encoder $g(\cdot)$ that maps it into a learnable embedding $\mathbf{q}(t) = g(t)$ that adapt to the data distribution, so the model can discover data-specific temporal signatures.

The final temporal embedding is the concatenation:

$$\mathbf{e}_t = [\mathbf{p}(t) \,\|\, \mathbf{q}(t)]. \tag{3.3}$$

The temporal encoding $\mathbf{e}_t$ is combined with edge features to enrich the contextual information during message passing.

3.3.2 Heterophily-Aware Temporal Multi-View Message Passing Framework

The core of the model is a message passing framework that integrates semantic, structural, and temporal information while addressing heterophily (camouflage). The framework operates in four stages: (i) per-view message construction, (ii) decoupled attention aggregation, (iii) multi-view attention fusion, and (iv) embedding update.

Per-View Message Construction

The heterogeneous graph is decomposed into *views*, where each view corresponds to a relation type (for example user–device or user–merchant). For each edge $e_{u\to v}$ in view r, a message from neighbor u to node v is constructed as:

$$\mathbf{m}^{(r)}_{u\to v} = \sigma(W^{(r)}_h h_u + W^{(r)}_e e_{u\to v}) \tag{3.4}$$

To make messages relation-aware, we apply a type-specific linear transformation (where $W^{(r)}_h, W^{(r)}_e$ are learnable weight matrices for both node embeddings and edge embedding for each relation type r) followed by a non-linear activation $\sigma(\cdot)$ such as ReLU activation.

Decoupled Attention Aggregation with Neighbor Selection

To address heterophily and class imbalance (so that the majority class doesn't overwhelm the information received from the less common class and reduce the overall smoothing effect), we design a decoupled aggregation scheme that separates neighbors into three groups, inspired by [26]:

— Labeled same-class neighbors ($\mathcal{N}_v^{\text{same}}$),
— Labeled different-class neighbors ($\mathcal{N}_v^{\text{diff}}$),
— Unlabeled neighbors ($\mathcal{N}_v^{\text{unl}}$).

Each group is aggregated independently using attention scores. First, the attention score $\alpha_{uv}^{(r)}$ for neighbor u is computed as:

$$\alpha_{uv}^{(r)} = \frac{\exp\Big(\text{LeakyReLU}\Big(\mathbf{a}_r^\top [\mathbf{W}_r \mathbf{h}_u^{(l)} \,\|\, \mathbf{W}_r \mathbf{h}_v^{(l)} \,\|\, \mathbf{e}_t]\Big)\Big)}{\sum_{k \in \mathcal{N}_v} \exp\Big(\text{LeakyReLU}\Big(\mathbf{a}_r^\top [\mathbf{W}_r \mathbf{h}_k^{(l)} \,\|\, \mathbf{W}_r \mathbf{h}_v^{(l)} \,\|\, \mathbf{e}_t]\Big)\Big)}. \tag{3.5}$$

We then apply a heuristic neighbor selection, where only the top-K neighbors with the highest attention scores are retained for aggregation, thus focusing on the most informative connections.

The decoupled aggregated message for node v in view r is:

$$\mathbf{h}_{v,\text{group}}^{(r)} = \sum_{u \in \mathcal{N}_v^{\text{group}}} \alpha_{uv}^{(r)} \mathbf{m}_{u \to v}^{(r)}, \tag{3.6}$$

where group $\in \{\text{same}, \text{diff}, \text{unl}\}$.

The three group-specific messages are concatenated:

$$\mathbf{h}_v^{(r)} = [\mathbf{h}_{v,\text{same}}^{(r)} \,\|\, \mathbf{h}_{v,\text{diff}}^{(r)} \,\|\, \mathbf{h}_{v,\text{unl}}^{(r)}]. \tag{3.7}$$

Multi-View Attention Fusion

Since multiple views exist for each node, we aggregate them via attention across views:

$$\mathbf{h}_v^{\text{multi}} = \sum_{r \in \mathcal{R}} \beta_v^{(r)} \mathbf{h}_v^{(r)}, \tag{3.8}$$

where $\beta_v^{(r)}$ is a learnable attention coefficient over relation types.

Embedding and Temporal Update

To incorporate temporal evolution, we update node embeddings with a gated recurrent unit (GRU):

$$\mathbf{h}_v^{(l+1)} = \text{GRU}\Big(\mathbf{h}_v^{(l)}, \mathbf{h}_v^{\text{multi}}\Big). \tag{3.9}$$

This choice ensures that each node embedding reflects not only its most recent fused neighborhood representation but also its historical state. In fraud detection, this is critical,

as it allows the model to capture sudden shifts, like a previously normal account suddenly exhibiting burst fraud, as well as gradual behavioral changes.

3.3.3 Classification and Loss

For node classification, the final embedding $\mathbf{h}_v^{(L)}$ after L layers is passed through a prediction layer:

$$\hat{y}_v = \text{softmax}(\mathbf{W}_c \mathbf{h}_v^{(L)} + \mathbf{b}_c). \tag{3.10}$$

The loss is computed using cross-entropy over labeled nodes:

$$\mathcal{L} = -\sum_{v \in \mathcal{V}_L} y_v \cdot \log(\hat{y}_v), \tag{3.11}$$

where $\mathcal{V}_L$ is the set of labeled nodes and y_v is the ground-truth label.

3.3.4 Training and Inference

The model is trained end-to-end with backpropagation. The overall training and inference procedure is summarized in Algorithm 1.

Algorithm 1: Heterophily-Aware Temporal Multi-View GNN Training

Input: Heterogeneous graph $G = (V, E)$, node features X, edge features$Z$, timestamps T, labels Y

Output: Trained node embeddings H and classifier parameters

for *epoch = 1* **to** E_{max} **do**

  foreach *node* $v_i \in V$ **do**

    Encode node features: $\mathbf{x}_i = MLP_\tau(X_i)$;

  foreach *edge* $e_j \in E$ **do**

    Encode timestamps: $\mathbf{e}_t = [\mathbf{p}(t) \,\|\, \mathbf{q}(t)]$;

    update edge features with encoded time :$z_j \leftarrow z_j \oplus e_t$;

  foreach *layer* $l = 1 \dots L$ **do**

    foreach *view r (relation type)* **do**

      foreach *node* v_i **do**

        Construct messages: $\mathbf{m}_{u\to v}^{(r)} = \sigma(W_h^{(r)} h_u + W_e^{(r)} e_{u\to v})$;

        Compute attention weights $\alpha_{ij}^{(r)}$ over neighbors j;

        Select top-k neighbors based on $\alpha_{ij}^{(r)}$;

        foreach *group* $\in \{same, diff, unl\}$ **do**

          $\mathbf{h}_{v,\text{group}}^{(r)} = \sum_{u\in\mathcal{N}_v^{\text{group}}} \alpha_{uv}^{(r)} \mathbf{m}_{u\to v}^{(r)}$;

        decoupled Aggregation: $\mathbf{h}_v^{(r)} = [\mathbf{h}_{v,\text{same}}^{(r)} \,\|\, \mathbf{h}_{v,\text{diff}}^{(r)} \,\|\, \mathbf{h}_{v,\text{unl}}^{(r)}]$;

    Fuse views with attention: $\mathbf{h}_v^{\text{multi}} = \sum_{r\in\mathcal{R}} \beta_v^{(r)} \mathbf{h}_v^{(r)}$;

    Temporal update: $\mathbf{h}_v^{(l+1)} = \text{GRU}\big(\mathbf{h}_v^{(l)}, \mathbf{h}_v^{\text{multi}}\big)$;

  Compute predictions: $\hat{y}_i = \text{softmax}(W_c \mathbf{h}_i^{(L)} + b_c)$;

  Compute loss: $\mathcal{L} = -\sum_i y_i \log \hat{y}_i$;

  Update parameters via gradient descent;

3.4 Results

[or tests and evaluation: including the test environment, the dataset, the evaluation metrics then the results and their discussion]

3.4.1 Test Environment

(env, datasets, and benchmark models)