CAPTURING HIGHER-ORDER FRAUD: MULTI-VIEW CONTINUAL LEARNING FOR STREAMING FRAUD DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

High-order fraud is increasingly prevalent in transaction networks, where fraudulent behaviors only emerge after aggregating weak signals across distant users. Existing graph-based methods typically rely on stacking multiple aggregation layers to capture such high-order patterns. However, in Streaming Fraud Detection (SFD), our theoretical analysis reveals that multi-layer aggregation can amplify the embedding drift bound under continuous concept drift, potentially leading to unstable model updates and degraded fraud detection performance. To address this issue, we propose MC-SFD, a Multi-view hypergraph Continual learning model for Streaming Fraud Detection, which reduces embedding drift through shallow hypergraph aggregation and effectively captures evolving high-order fraud. MC-SFD incorporates an RL-based Adaptive Hypergraph Constructor (RL-AHC) to dynamically evolve hypergraph structures and a Multi-view Personalized Prompt Generator (MPPG) for efficient adaptation with minimal parameter updates. In addition, both RL-AHC and MPPG integrate LLMs for policy initialization and semantic enhancement, respectively. Experiments on four SFD datasets show that MC-SFD consistently outperforms 23 state-of-the-art baselines by 1.63% to 3.71% in AUC.

1 INTRODUCTION

Emerging fraud patterns are increasingly higher-order and collaborative (Motie & Raahemi, 2024; Hafez et al., 2025), where a fraudster often cannot be reliably identified from local signals (i.e., its own attributes and immediate neighbors) in transaction graphs. Instead, reliable evidence emerges only after aggregating weak signals across multiple distant entities. We formalize this notion by defining the *order* o of a fraud pattern (Figure 1) as the minimum hop distance required to identify a target fraudster. For instance, a 0-order fraud can be detected using the fraudster's own attributes (e.g., a fake identity), whereas detecting a 4-order fraud requires information from four hops away in a transaction graph. In principle, an L-layer Graph Neural Network (GNN) aggregates information within L hops (Cheng et al., 2025), and thus can represent patterns up to order L.

Although existing methods can deepen GNNs to model high-order fraud, they suffer from over-smoothing (Shen et al., 2024) and high computational complexity (Feng et al., 2025b). Furthermore, our analysis reveals that deep GNNs can be more susceptible to embedding drift in Streaming Fraud Detection (SFD). As shown in Figure 2, in SFD, streaming transactions form sequential tasks, continuously introducing concept drift at both the structural and feature levels. Specifically, when task $(i+1)$ arrives, both structural drift $\Delta\mathbf{A}_{i+1}$ and feature drift $\Delta\mathbf{X}_{i+1}$ propagate through L-hop aggregation, eventually inducing embedding drift $\Delta\mathbf{Z}^{L}_{i+1}$. In this work, we derive a theoretical upper bound on embedding drift under different fraud orders and estimate it using the Wasserstein distance (Panaretos & Zemel, 2019). Both our analysis and empirical results show that the Embedding Drift Bound (EDB) for standard GNN backbones can grow exponentially with fraud order due to the amplification effect of multi-layer aggregation. Consequently, existing methods can be less effective for high-order fraud, as they struggle to maintain stable representations under severe embedding drift.

This finding motivates us to exploit Hypergraph Neural Networks (HGNNs) (Antelmi et al., 2023), which model high-order relations via hyperedges and capture distant fraud signals with fewer

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

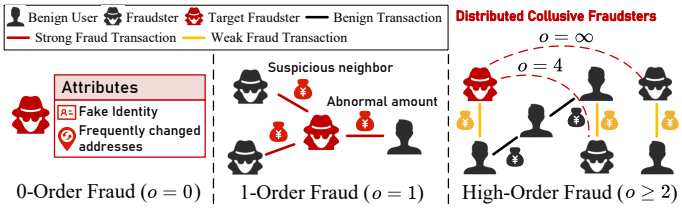


Figure 1: Illustration of fraud patterns at different orders.

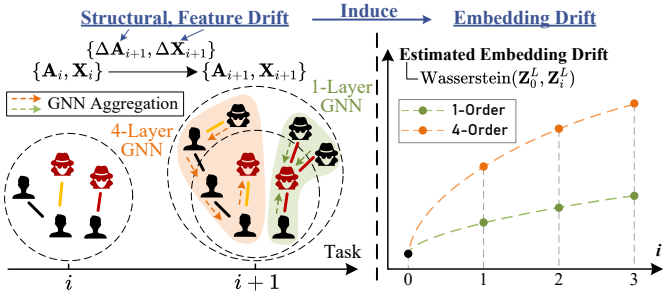


Figure 2: Illustration of two GNNs that capture 1-order and 4-order fraud in SFD, respectively, and their estimated embedding drift using the Wasserstein distance. $\mathbf{A}_i$, $\mathbf{X}_i$, and $\mathbf{Z}_i^L$ denote the adjacency matrix, node features, and node embeddings at task i, respectively.

aggregation layers, thereby reducing embedding drift in SFD. Despite promising results, existing HGNN methods face two key challenges in SFD due to continuously evolving fraud patterns (Lee et al., 2025). **(1) Structural Inflexibility.** Most HGNNs rely on static, hand-crafted hypergraph construction rules (Feng et al., 2019; Gao et al., 2022), making them unable to adapt to evolving high-order fraud patterns in SFD. **(2) Exacerbated Stability-Plasticity Trade-off.** Compared with standard GNNs, HGNNs typically introduce more parameters and incur higher update costs in continual learning (Lee et al., 2025), making it harder to preserve previously learned patterns while rapidly adapting to emerging fraud.

To address these challenges, we propose **MC-SFD**, a **M**ulti-view hypergraph **C**ontinual learning model for **S**treaming **F**raud **D**etection that captures high-order fraud via adaptive hyper-views while reducing embedding drift. Our contributions are summarized as follows:

- We provide the first theoretical analysis of **Embedding Drift Bounds (EDB)** across different fraud orders, revealing the amplification effect under multi-layer aggregation. This insight motivates us to integrate hyper-views to capture high-order fraud with reduced EDB.
- To overcome structural inflexibility, we propose an **RL-based Adaptive Hypergraph Constructor (RL-AHC)**, which dynamically constructs hyper-views through reinforcement learning with LLM-assisted initialization, tracking evolving fraud patterns in SFD.
- To mitigate the stability-plasticity trade-off, MC-SFD freezes the main encoder and updates a lightweight **Multi-view Personalized Prompt Generator (MPPG)** during continual learning, enabling fast adaptation to new fraud patterns with LLM-assisted semantic embeddings.
- Extensive experiments on four SFD datasets demonstrate that MC-SFD consistently outperforms 23 state-of-the-art baselines, with AUC gains of +3.71%, +1.94%, +3.21%, and +1.63% over the strongest baseline in four datasets, respectively.

2 Related Work

Fraud Detection on Graphs. Graph neural networks (GNNs) are widely used for graph-based fraud detection in financial transaction networks (Cheng et al., 2025; Ou et al., 2026b) and social networks (Qiao et al., 2025; Xu et al., 2024), with representative methods such as PC-GNN (Liu et al., 2021). Recent methods, such as DGNN-SR (Yuan et al., 2025), RGTAN (Xiang et al., 2025), and CamFD (Ou et al., 2026b), have incorporated temporal information to capture dynamic fraud patterns. However, most existing methods are designed for non-streaming settings and mainly rely

on local neighborhood aggregation, which limits their ability to capture high-order collusive fraud patterns in SFD.

Streaming Fraud Detection. Real-world anti-fraud systems operate in streaming environments, where transactions arrive continuously and fraud patterns evolve over time (Motie & Raahemi, 2024; Khodabandehlou & Golpayegani, 2024). Existing graph continual learning methods mainly address catastrophic forgetting (Tian et al., 2024; Zhang et al., 2024b), including replay-based approaches (e.g., SGNN-GR(Wang et al., 2022), DSLR (Choi et al., 2024), RL-GNN (Song et al., 2025)) and parameter-isolation approaches (e.g., MSCGL (Cai et al., 2022) and PI-GNN (Zhang et al., 2023)). Nevertheless, these methods mostly focus on low-order structures, and the theoretical understanding of embedding drift under high-order streaming fraud remains limited.

High-order Graph Learning. High-order graph learning captures relations beyond pairwise interactions (Antelmi et al., 2023; Lee et al., 2025), and HGNN-based models (e.g., HGNN (Feng et al., 2019), HGNN+ (Gao et al., 2022), T-HyperGNN (Wang et al., 2024b), MHGNN (Ji et al., 2025), MH-LGC (Ou et al., 2026a), and TROPICAL (Haghighi et al., 2024)) have shown strong capability in hypergraph representation learning. However, most HGNN methods rely on static or heuristic hypergraph construction and are not designed for fast adaptation to evolving streaming fraud patterns.

Extended discussions are provided in Appendix N.

3 Theoretical Analysis and Motivation

Definition 3.1 (Transaction Graph, TG)**.** A TG is defined as $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathbf{X}_{\mathcal{N}}, \mathbf{X}_{\mathcal{E}})$, where each node $u \in \mathcal{N}$ represents a user, and each edge $e = (u, v, t) \in \mathcal{E}$ denotes a transaction occurring between users u and v at time t. $\mathbf{X}_{\mathcal{N}} \in \mathbb{R}^{|\mathcal{N}| \times d_{\mathcal{N}}}$ and $\mathbf{X}_{\mathcal{E}} \in \mathbb{R}^{|\mathcal{E}| \times d_{\mathcal{E}}}$ denote the user and transaction feature matrices, respectively, where $d_{\mathcal{N}}$ and $d_{\mathcal{E}}$ are the corresponding feature dimensions.

Before analyzing high-order fraud patterns, we first formalize the computation of message-passing GNNs. Note that we make two simplifications for tractable analysis, which will be justified in Appendix A: (1) We only consider the feature drift of nodes, denoting $\mathbf{X} = \mathbf{X}_{\mathcal{N}}$. (2) We adopt GCN (Kipf & Welling, 2017) as a representative message-passing GNN.

Definition 3.2 (Message Passing GNNs)**.** Consider an L-layer GNN on $\mathcal{G}$ with adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$. The forward propagation at the l-th layer is defined as:

$$\mathbf{Z}^{l+1} = \sigma\left(\mathbf{A}\mathbf{Z}^{l}\mathbf{W}\right), \quad \mathbf{Z}^{0} = \mathbf{X}, \tag{1}$$

where $\sigma(\cdot)$ is the activation function, $\mathbf{Z}^l$ denotes the node representation at l-th layer, and $\mathbf{W}$ denotes a shared learnable weight matrix across layers. Accordingly, the final-layer representation $\mathbf{Z}^L$, which is also called the node embedding, can be derived as:

$$\mathbf{Z}^{L} = \sigma\left(\mathbf{A}\left[\ldots\sigma\left(\mathbf{A}\mathbf{X}\mathbf{W}\right)\ldots\right]\mathbf{W}\right). \tag{2}$$

Based on the above definitions, we define and analyze the order of fraud as follows:

Definition 3.3 (Order of Fraud Patterns)**.** A fraud pattern has order o if the identification of a target fraudster depends on information from its o-hop neighborhood in $\mathcal{G}$.

Lemma 3.4. *For any L-layer GNN, the embedding of a node depends on information within its L-hop neighborhood. Consequently, detecting an o-order fraud pattern requires $L \geq o$.*

In other words, to capture higher-order fraud patterns, a deeper GNN is required. However, in SFD, the $(i+1)$-th new task introduces both structural drift $\Delta\mathbf{A}_{i+1}$ and feature drift $\Delta\mathbf{X}_{i+1}$, which induce embedding drift:

$$\Delta\mathbf{Z}^{L}_{i+1} = \mathbf{Z}^{L}_{i+1} - \mathbf{Z}^{L}_{i}. \tag{3}$$

Proposition 3.5 (Embedding Drift Bound, EDB)**.** *Consider an L-layer GNN defined by Eq. (1), with an L_σ-Lipschitz continuous activation function $\sigma(\cdot)$ with respect to the Frobenius norm. Upon structural drift $\Delta\mathbf{A}_{i+1}$ and feature drift $\Delta\mathbf{X}_{i+1}$, the embedding drift is bounded as follows:*

$$\|\Delta\mathbf{Z}^{L}_{i+1}\|_F \leq \underbrace{(\gamma)^{L}\,\|\Delta\mathbf{X}_{i+1}\|_F}_{\textit{Feature Drift}} + \underbrace{\sum_{l=1}^{L}(\gamma)^{L-l}\,L_\sigma\alpha\|\mathbf{Z}^{l-1}_{i}\|_F\|\Delta\mathbf{A}_{i+1}\|_{\mathrm{sp}}}_{\textit{Structural Drift}}. \tag{4}$$

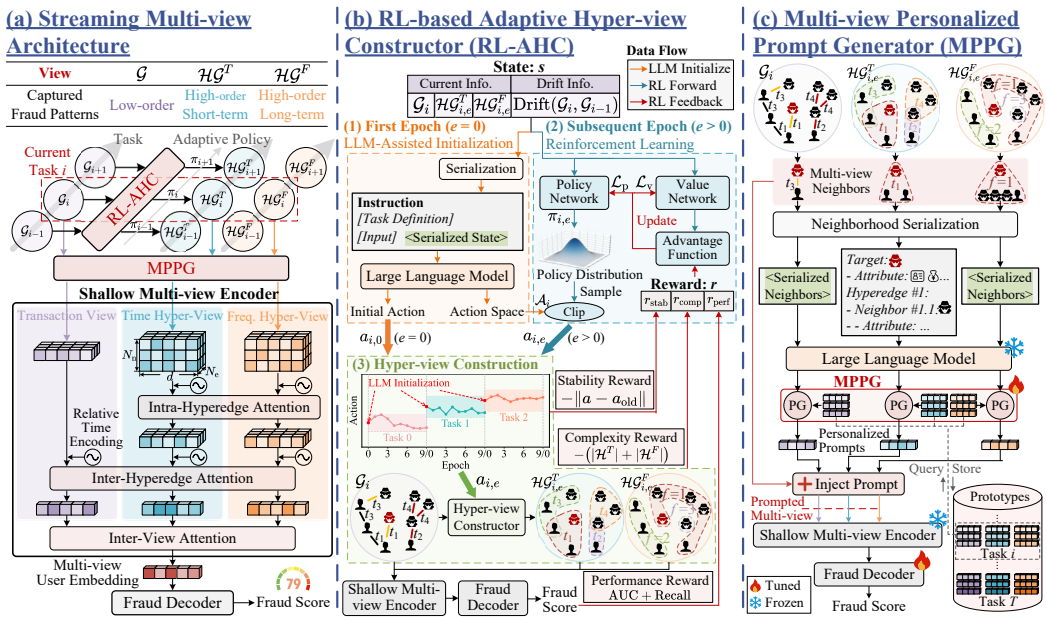


Figure 3: Overall architecture of MC-SFD.

Here, $\|\cdot\|_F$ and $\|\cdot\|_{\text{sp}}$ denote the Frobenius and spectral norms, respectively. We assume $\|\mathbf{W}\|_{\text{sp}} \leq \alpha$ and $\|\mathbf{A}\|_{\text{sp}} \leq \beta$, and define $\gamma = L_\sigma \beta \alpha$.

The proof of Proposition 3.5 is provided in Appendix B. Our experiments in Appendix C provide empirical evidence that $\gamma > 1$ commonly holds for standard GNN backbones across four datasets. When $\gamma > 1$, Eq. (4) suggests that both feature-drift term and structural-drift terms grow exponentially with GNN depth L.

In SFD, high EDB makes it more challenging for fraud detection models to identify evolving fraud patterns, as it may require larger parameter updates and make continual adaption more difficult. This finding motivates us to introduce HGNNs for multi-view fraud detection. Since hyperedges can effectively connect multiple distant users, HGNNs can capture high-order fraud patterns with fewer aggregation layers, thereby reducing the EDB.

Definition 3.6 (Transaction Hypergraph, TH)**.** A TH $\mathcal{HG} = (\mathcal{N}, \mathcal{H}, \mathbf{X}_\mathcal{N}, \mathbf{X}_\mathcal{H})$ is constructed based on $\mathcal{G}$, which shares the same user set $\mathcal{N}$ and user feature matrix $\mathbf{X}_\mathcal{N}$ with $\mathcal{G}$. Each hyperedge $h = (\mathcal{N}_h, t) \in \mathcal{H}$ connects a set of users $\mathcal{N}_h$ with a timestamp t, according to a predefined construction rule (e.g., similar transaction times or frequencies). $\mathbf{X}_\mathcal{H} \in \mathbb{R}^{|\mathcal{H}| \times d_\mathcal{H}}$ denotes the hyperedge feature matrix.

Problem Formulation: Multi-view Streaming Fraud Detection. A streaming TG is defined as $\mathcal{SG} = (\mathcal{G}_1, \ldots, \mathcal{G}_T)$, where $\mathcal{G}_i$ denotes the TG of the i-th chronologically divided task and T is the number of tasks. Each transaction (edge) is associated with a label $y \in \{0, 1\}$, indicating whether the transaction is benign or fraudulent. To capture high-order fraud patterns, we first construct one or more transaction hypergraphs as hyper-views for each $\mathcal{G}_i$. We then incrementally train a model that classifies transactions into two classes using multiple views as follows: $f_{\theta_i} : (\mathcal{G}_i, \mathcal{HG}_i^1, \ldots) \to \hat{\mathcal{Y}}_i$. Here, f_{θ_i} denotes the model parameterized by θ_i, and $\mathcal{HG}_i^1$ denotes one constructed hyper-view. $\hat{\mathcal{Y}}_i \in \{0, 1\}^{|\mathcal{E}_i|}$ denotes the predicted labels.

4 Methodology

4.1 Streaming Multi-view Architecture

The overall architecture of MC-SFD is shown in Figure 3(a). For a streaming transaction graph sequence $\mathcal{SG} = (\mathcal{G}_1, \ldots, \mathcal{G}_T)$, MC-SFD constructs three views for each task: the original transaction

graph view $\mathcal{G}_i$, a time hyper-view $\mathcal{HG}_i^T$, and a frequency hyper-view $\mathcal{HG}_i^F$. These two hyper-views explicitly capture short- and long-term high-order fraud patterns and enable shallow aggregation with reduced EDB (Proposition 3.5).

To effectively integrate the three views, we design a **Shallow Multi-view Encoder (SME)** to compute the multi-view user embedding $\mathbf{Z}_i$:

$$\mathbf{Z}_i = \text{SME}\left(\mathcal{G}_i, \mathcal{HG}_i^T, \mathcal{HG}_i^F\right). \tag{5}$$

SME unifies the message passing of graphs and hypergraphs via three hierarchical attention modules, which effectively capture the complex interactions among different views. See Appendix E for details of the SME design.

Based on $\mathbf{Z}_i$, a downstream fraud decoder predicts transaction-level fraud scores $\hat{y}_{i,e}$.

$$\hat{y}_{i,e} = \text{FraudDecoder}\left(\mathbf{z}_{i,u}, \mathbf{z}_{i,v}\right), \quad e \in \mathcal{E}_i, \tag{6}$$

where $\mathbf{z}_{i,u}$ denotes the embedding of node u in task i, and $\hat{\mathcal{Y}}_i = \{\hat{y}_{i,e} \mid e \in \mathcal{E}_i\}$.

MC-SFD is incrementally optimized using classification loss $\mathcal{L}^{\text{cla}}$ with a learning rate η:

$$\mathcal{L}_i^{\text{cla}} = -\frac{1}{|\mathcal{E}_i|} \sum_{e \in \mathcal{E}_i} \left[y_{i,e} \log \hat{y}_{i,e} + (1 - y_{i,e}) \log(1 - \hat{y}_{i,e})\right], \tag{7}$$

$$\theta_i \leftarrow \text{Optimizer}\left(\theta_{i-1}, \nabla \mathcal{L}_i^{\text{cla}}\right), \tag{8}$$

In addition to single task learning with SME, MC-SFD further introduces RL-AHC to adapt hyper-view construction and MPPG to enable lightweight continual adaptation.

4.2 RL-BASED ADAPTIVE HYPERGRAPH CONSTRUCTOR (RL-AHC)

Most existing methods operate on static hypergraphs, which induces structural inflexibility and hinders their ability to model continuously evolving high-order fraud patterns in SFD (Lee et al., 2025). RL-AHC formulates hyper-view construction as policy learning, dynamically adapting to the evolving patterns with LLM-assisted initialization and reinforcement learning.

LLM-assisted Initialization. As shown in Figure 3(b), when a new task $\mathcal{G}_i$ arrives, RL-AHC employs an LLM agent in the first epoch to initialize the action $a_{i,0}$ and action space $\mathcal{A}_i$ from the initial state $s_{i,0}$, alleviating the cold-start issue and accelerating convergence.

$$a_{i,0},\ \mathcal{A}_i = \text{LLM}(s_{i,0}). \tag{9}$$

The input template and its output format are provided in Appendix M.1.

Reinforcement Learning. In subsequent epochs, RL-AHC optimizes the hyper-view construction policy via Proximal Policy Optimization (PPO) (Schulman et al., 2017; Zheng et al., 2025). For epoch e, a policy network $\mathcal{P}(\cdot)$ outputs the parameters of a Gaussian distribution, which defines a stochastic hyper-view construction policy $\pi(a \mid s_{i,e})$.

$$\mu, \sigma = \mathcal{P}(s_{i,e}), \tag{10}$$

$$\pi(a \mid s_{i,e}) = \mathcal{N}\left(\mu, \Sigma\right), \quad \Sigma = \text{diag}(\sigma^2), \tag{11}$$

$$a_{i,e} = \text{clip}\left(\tilde{a}_{i,e}, \mathcal{A}_i\right), \quad \tilde{a}_{i,e} \sim \pi(a \mid s_{i,e}). \tag{12}$$

The action controls the temporal and frequency tolerances used by the constructor (see Appendix F):

$$\mathcal{HG}_{i,e}^T, \mathcal{HG}_{i,e}^F = \mathcal{C}(\mathcal{G}_i, a_{i,e}), \tag{13}$$

where $\mathcal{C}(\cdot)$ is the hyper-view construction operator.

RL-AHC adopts a multi-objective reward that jointly balances detection performance, hyper-view complexity, and policy stability: The overall reward is defined as follows:

$$r = \alpha_1 r_{\text{perf}} + \alpha_2 r_{\text{comp}} + \alpha_3 r_{\text{stab}},\ r_{\text{perf}} = \text{AUC}_{i,e} + \text{Recall}_{i,e}, \tag{14}$$

$$r_{\text{comp}} = -\left(\left|\mathcal{H}_{i,e}^T\right| + \left|\mathcal{H}_{i,e}^F\right|\right),\ r_{\text{stab}} = -\left\|a_{i,e} - a_{i,e-1}\right\|_2, \tag{15}$$

Table 1: Dataset statistics. Fraud rate denotes the percentage of fraud edges.

	S-FFSD	Credit	Wikipedia	Reddit
#Nodes	31,232	107,539	9,227	10,984
#Edges	77,881	5,125,108	157,474	672,447
#Edge Feat.	123	7	172	172
Fraud Rate	6.75%	2.88%	0.14%	0.54%
Time Span	90 days	180 days	30 days	30 days
#Tasks	10	10	10	10

where $\alpha_1, \alpha_2, \alpha_3$ are coefficients that balance the contributions of the reward terms, and $|\mathcal{H}_{i,e}^T|$ and $|\mathcal{H}_{i,e}^F|$ denote the numbers of hyperedges in the two hyper-views, respectively.

Based on the rewards, the K-step return and advantage are computed using a value network $\mathcal{V}(\cdot)$ as:

$$G_\tau^{(K)} = \sum_{k=0}^{K} \gamma^k r_{\tau+k} + \gamma^{K+1} \mathcal{V}(s_{\tau+K+1}), \quad \delta_\tau = G_\tau^{(K)} - \mathcal{V}(s_\tau), \tag{16}$$

where γ is a discount factor, K denotes the rollout horizon, and τ denotes the rollout step index.

The objectives of the policy and value networks are defined as:

$$\mathcal{L}_{\mathrm{p}}(\theta_{\mathrm{p}}) = -\mathbb{E}_\tau \left[\min \left(\rho_\tau \cdot \delta_\tau, \mathrm{clip} \left(\rho_\tau, 1-\epsilon, 1+\epsilon \right) \cdot \delta_\tau \right) \right], \tag{17}$$

$$\mathcal{L}_{\mathrm{v}}(\theta_{\mathrm{v}}) = \mathbb{E}_\tau \left[\left(G_\tau^{(K)} - \mathcal{V}(s_\tau) \right)^2 \right], \tag{18}$$

where $\rho_\tau = \frac{\pi(a_\tau|s_\tau)}{\pi_{\mathrm{old}}(a_\tau|s_\tau)}$ is the policy probability ratio.

4.3 Multi-view Personalized Prompt Generator (MPPG)

To improve adaptation efficiency in continual learning, we introduce MPPG as shown in Figure 3(c), which updates lightweight learnable prompts instead of the full SME.

Prompt Prototypes. For each task i and view v, MPPG learns a set of k prompt prototypes:

$$\mathbf{P}_i^v = \left\{ \mathbf{p}_{i,1}^v, \ldots, \mathbf{p}_{i,k}^v \right\}, \quad \mathbf{p}_{i,j}^v \in \mathbb{R}^{d_p}, \tag{19}$$

where $d_p = d_{\mathcal{N}}$ matches the node feature dimension. All learned prompt prototypes are stored in a prototype bank and reused during inference.

Prompt Generation (PG). To generate node-specific prompts from a limited set of prototypes, MPPG composes personalized prompts by conditioning on both the target node feature and its *LLM-assisted semantic embedding*. Specifically, for a target node u under view v, MPPG generates a semantic embedding $\mathbf{s}_u^v$ from serialized neighborhood context:

$$\mathbf{s}_u^v = \mathrm{LLM}(\mathcal{S}_u^v), \quad \mathcal{S}_u^v = \mathrm{Serialize}(\mathbf{x}_u, \mathcal{N}^v(u)). \tag{20}$$

To improve training efficiency, these semantic embeddings are pre-computed and cached for each task based on the initial three views. The shared text template for the three views is provided in Appendix M.2.

The semantic-enhanced node representation is obtained via additive modulation and is then used to compute the prototype composition weights $\boldsymbol{\alpha}_u^v$:

$$\boldsymbol{\alpha}_u^v = \mathrm{Softmax} \left(\mathbf{W}_p^v \tilde{\mathbf{x}}_u \right), \quad \tilde{\mathbf{x}}_u = \mathbf{x}_u + \mathbf{W}_s^v \bar{\mathbf{s}}_u^v, \tag{21}$$

where $\boldsymbol{\alpha}_u^v \in \mathbb{R}^k$ indicates the contribution of each prototype; $\mathbf{x}_u$ and $\tilde{\mathbf{x}}_u$ denote the raw and enhanced node features, respectively; $\mathbf{W}_s^v$ is a projection matrix. Personalized prompt is then composed and injected into the node feature.

$$\tilde{\mathbf{p}}_{u,i}^v = \sum_{j=1}^{k} \alpha_{u,j}^v \mathbf{p}_{i,j}^v, \quad \hat{\mathbf{x}}_u^v = \mathbf{x}_u + \tilde{\mathbf{p}}_{u,i}^v. \tag{22}$$

Table 2: Overall streaming fraud detection performance (mean±std).

Datasets	S-FFSD		Credit		Wikipedia		Reddit	
Metrics	AUC↑	AFR↓	AUC↑	AFR↓	AUC↑	AFR↓	AUC↑	AFR↓
TGAT (Retrained)	74.24±2.41	/	90.35±1.24	/	83.69±0.70	/	65.56±0.70	/
RF	56.23±4.33	10.38±1.96	69.62±5.34	9.67±2.62	65.83±4.02	5.69±1.61	54.26±3.64	3.12±0.51
SVM	56.70±4.76	9.79±2.08	69.47±5.07	9.92±2.49	62.41±4.26	5.73±1.74	54.35±2.92	2.94±0.36
TGAT	67.23±2.76	6.84±1.87	83.65±1.45	8.74±0.72	76.31±3.48	6.57±1.69	60.30±3.27	2.78±0.42
HGNN	65.97±3.27	5.88±1.68	83.17±2.03	8.59±1.87	75.73±2.81	5.83±1.73	58.84±2.61	3.18±0.71
SAD	68.10±2.21	6.21±1.60	83.95±1.85	7.64±1.68	78.76±2.75	4.20±2.09	61.97±2.85	2.62±0.36
AdaDyGNN	67.20±2.46	6.46±1.73	83.87±1.97	8.01±1.76	78.01±2.89	4.72±1.66	61.64±3.06	2.83±0.41
T-HyperGNN	68.32±2.76	6.03±1.48	84.28±1.82	7.60±1.58	78.81±3.27	4.85±1.61	60.76±2.95	2.71±0.53
OFA	68.86±2.73	6.13±1.90	83.42±1.89	7.92±1.84	78.64±3.18	4.87±1.86	60.86±2.87	2.78±0.56
TEA-GLM	69.37±2.87	6.21±1.69	84.03±1.75	8.03±1.79	78.82±2.71	4.93±1.79	61.42±2.94	2.54±0.45
GAugLLM	69.67±2.45	5.78±1.79	85.38±1.68	7.63±1.48	78.53±2.48	4.42±1.64	61.56±2.87	2.47±0.39
RGTAN	69.40±2.38	5.47±1.68	84.24±1.60	7.80±1.62	78.23±3.09	4.57±1.87	61.87±2.70	2.87±0.30
KHGNN	69.26±2.86	5.34±1.72	85.71±1.79	7.46±1.38	78.65±2.68	4.93±1.72	60.92±2.86	2.67±0.57
CamFD	70.52±1.98	5.93±1.88	86.04±1.63	7.06±1.50	79.19±2.36	4.80±1.69	61.75±3.08	2.61±0.42
ER-GNN	69.11±2.51	4.01±1.59	87.62±1.56	1.98±1.08	81.35±1.09	-0.58±0.86	62.24±1.57	-1.01±0.49
SSRM	69.03±2.36	4.26±1.75	87.53±1.28	1.56±0.78	81.83±0.75	-1.06±0.80	62.75±1.60	-1.32±0.41
SGNN-GR	69.31±2.28	4.21±1.39	87.86±1.34	1.49±0.97	81.47±1.21	-0.86±0.91	62.81±1.79	-1.39±0.52
DSLR	69.38±2.25	4.34±1.27	87.70±1.46	1.52±0.80	81.62±1.03	-0.92±0.79	62.84±1.74	-1.37±0.49
RL-GNN	69.19±2.79	3.91±1.65	87.91±1.39	1.69±0.98	81.10±1.27	-0.85±0.89	61.97±1.56	-1.33±0.54
ER-GNN+TGAT	69.97±2.26	3.65±1.28	87.98±1.49	1.76±0.69	82.14±1.32	-0.76±0.84	62.93±1.73	-1.26±0.54
SSRM+TGAT	69.85±1.92	3.73±1.32	88.06±1.50	1.48±0.74	82.31±1.07	-0.98±0.89	63.08±1.67	-1.30±0.59
SGNN-GR+TGAT	70.02±1.87	3.57±1.45	88.03±1.32	1.56±0.81	82.38±1.29	-0.80±0.76	62.98±1.81	-1.38±0.51
DSLR+TGAT	70.15±1.76	3.61±1.20	88.08±1.46	1.51±0.77	82.40±1.22	-0.87±0.81	63.02±1.80	-1.36±0.61
RL-GNN+TGAT	70.04±2.13	3.48±1.42	88.14±1.39	1.54±0.86	82.09±1.34	-0.84±0.69	62.81±1.65	-1.29±0.72
MC-SFD	**74.23±0.95**	**2.73±0.73**	**90.08±1.07**	**1.27±0.71**	**85.61±0.92**	**-1.53±0.78**	**64.71±1.76**	**-1.50±0.57**

5 EXPERIMENTS

We evaluate MC-SFD with four research questions: **RQ1.** Does MC-SFD improve streaming fraud detection performance under continual learning constraints? **RQ2.** Do the empirical embedding-drift trends match our theoretical analysis? **RQ3.** How do the proposed modules contribute to the overall gains? **RQ4.** What computational overhead does MC-SFD introduce in continual learning?

5.1 EXPERIMENTAL SETUP

Datasets. We use four streaming fraud detection datasets: two financial transaction datasets (S-FFSD (Xiang et al., 2025) and Credit), one web-editing dataset (Wikipedia (Tian et al., 2023)), and one posting dataset (Reddit (Tian et al., 2023)). Each dataset is chronologically split into ten sequential tasks. Following prior work, S-FFSD uses a 60%-20%-20% train/validation/test split (Xiang et al., 2025), while other datasets use 70%-15%-15% (Tian et al., 2023).

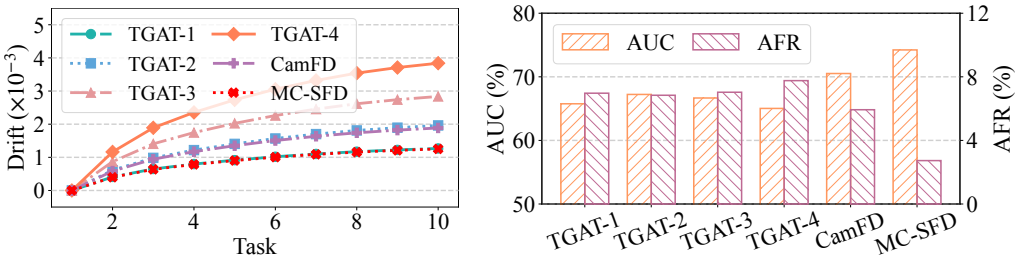
Baselines. We compare against 23 baselines: two traditional machine learning methods (RF and SVM), eleven non-continual graph methods, five continual graph methods, and five TGAT-based continual variants. We also report *TGAT (Retrained)* as a full-data retraining reference, which is impractical in real-world systems.

Metrics. We report AUC and AFR (average forgetting rate). AUC is a threshold-independent ranking metric for imbalanced fraud detection (Ou et al., 2026b). AFR quantifies forgetting as $\text{AFR} = \frac{1}{T}\sum_{i=1}^{T}(p_{i,i} - p_{T,i})$, where $p_{i,j}$ is the AUC on task j after training tasks 1 through i.

Implementation. All tables report mean ± standard deviation over ten repeated runs with different random initializations. MC-SFD uses an 8-head, 128-dimensional TGAT backbone in SME (Xu et al., 2020), and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) for both RL-AHC initialization and semantic embedding in MPPG. RL-AHC hyperparameters are $\alpha_1 = 1.0$, $\alpha_2 = 10^{-5}$, $\alpha_3 = 10^{-2}$, $\gamma = 0.9$, and $K = 1$. MPPG uses $k = 3$ prompt prototypes per view. Experiments are conducted on one NVIDIA V100 (32GB), Intel Xeon Gold 6230 CPUs, 64GB RAM, and PyTorch 1.10.1.

5.2 RQ1: OVERALL PERFORMANCE

Table 2 demonstrates the overall streaming fraud detection (SFD) performance of MC-SFD and the baselines in terms of AUC and AFR. The incrementally trained baselines can be categorized into four groups. *(1) Traditional machine learning methods* ignore graph dependencies and therefore



(a) Embedding drift (b) Performance

Figure 4: Embedding drift evaluation on S-FFSD.

Table 3: Ablation study.

Datasets	S-FFSD		Credit		Wikipedia		Reddit	
Metrics	AUC↑	AFR↓	AUC↑	AFR↓	AUC↑	AFR↓	AUC↑	AFR↓
w/o $\mathcal{HG}^T$	70.72±1.36	3.39±1.38	89.11±1.27	1.46±0.98	83.29±1.22	-1.08±0.92	64.02±1.74	-1.30±0.66
w/o $\mathcal{HG}^F$	73.89±0.98	2.81±1.02	88.94±1.16	1.58±0.86	83.56±1.14	-1.14±0.87	63.48±1.85	-1.36±0.59
w/o LLM-Init	73.91±1.26	2.91±0.84	89.68±1.36	1.42±0.63	85.03±0.74	-1.38±0.89	64.26±1.99	-1.36±0.62
w/o RL	72.27±1.35	3.30±0.71	88.93±1.52	1.60±0.91	83.90±0.85	-1.10±0.81	63.62±1.70	-1.30±0.67
w/o RL-AHC	71.26±1.65	3.35±0.71	88.76±1.42	1.66±0.84	83.72±0.95	-1.05±0.75	63.59±1.65	-1.30±0.72
w/o LLM-Emb	73.98±1.58	2.81±0.82	89.81±1.38	1.39±0.89	85.42±1.20	-1.42±0.91	64.53±1.84	-1.46±0.47
w/o MPPG	73.86±0.82	2.96±0.85	89.64±1.47	1.52±0.86	85.08±1.13	-1.28±0.96	64.37±1.88	-1.44±0.59
MC-SFD	**74.23±0.95**	**2.73±0.73**	**90.08±1.07**	**1.27±0.71**	**85.61±0.92**	**-1.53±0.78**	**64.71±1.76**	**-1.50±0.57**

perform worst in streaming fraud detection. *(2) Non-continual graph learning methods* can capture structural dependencies but suffer from severe forgetting in incremental training, leading to high AFR. *(3) Continual graph learning methods* reduce forgetting via replay or parameter isolation, but their static backbones limit temporal fraud modeling. *(4) TGAT-based continual methods* share the same temporal encoder as MC-SFD, providing a fair comparison. The improvements over their static-backbone counterparts highlight the importance of modeling temporal information for SFD.

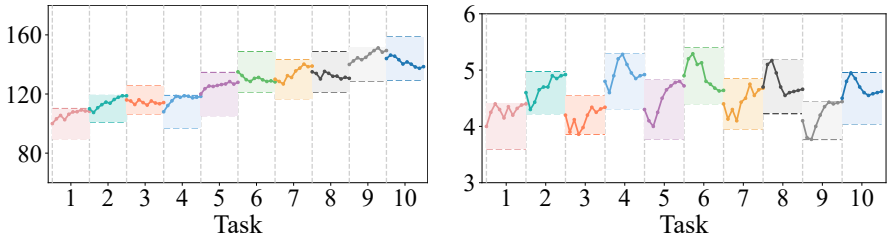
MC-SFD consistently outperforms 23 baselines across all test cases. Compared with the strongest baseline for each dataset, MC-SFD improves the AUC by +3.71%, +1.94%, +3.21%, and +1.63%, respectively. Notably, MC-SFD is also close to the full-data retrained TGAT on S-FFSD (74.23% vs. 74.24%) and Credit (90.08% vs. 90.35%), and even surpasses it on Wikipedia (85.61% vs. 83.69%), further demonstrating the effectiveness of our method.

5.3 RQ2: Embedding Drift Evaluation

To verify the theoretical analysis in Section 3, we estimate the embedding drift using Wasserstein distance (Appendix F.1) for different GNN encoders on S-FFSD. Figure 4(a) compares TGAT with 1–4 aggregation layers, CamFD (2-layer TGAT backbone), and MC-SFD across ten continual tasks. As TGAT depth increases from 1 to 4 layers, the final-task drift grows from approximately 1.3×10^{-3} to 3.9×10^{-3}, which matches the drift amplification trend in our analysis. CamFD exhibits a drift curve comparable to TGAT-2. According to Lemma 3.4, an L-layer GNN captures patterns within at most L hops. Thus, CamFD and TGAT-2 are limited to 2-hop patterns. Although deeper models (e.g., TGAT-4) can capture higher-order patterns, they suffer from substantially larger embedding drift, leading to degraded performance (Figure 4(b)). In contrast, MC-SFD captures high-order patterns with a shallow multi-view encoder, achieving lower embedding drift while maintaining superior detection performance.

5.4 RQ3: Component Analysis

Ablation study. Table 3 reports the ablation results. *w/o* $\mathcal{HG}^T$ and *w/o* $\mathcal{HG}^F$ remove the temporal and frequency hyper-views, respectively, both of which lead to consistent performance drops, indicating that the two views provide complementary information. *w/o LLM-Init* and *w/o RL* disable the LLM-assisted initialization and the reinforcement learning component in RL-AHC, respectively, while *w/o RL-AHC* removes the entire module and freezes the hyper-views. All three variants show clear degradation, highlighting the benefit of adaptive hyper-view construction. *w/o LLM-Emb* removes



(a) Time tolerance Δt (b) Frequency tolerance Δf

Figure 5: Evolving RL-AHC actions during continual learning on S-FFSD.

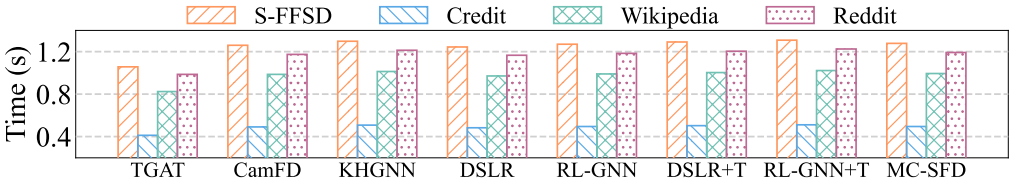


Figure 6: Per-batch training time comparison.

semantic embeddings in MPPG, and *w/o MPPG* disables the entire module. The results suggest that semantic embeddings provide a modest but generally consistent contribution within MPPG.

Adaptive construction in RL-AHC. Figure 5 shows the evolving actions in RL-AHC, which determine the time and frequency tolerances (Δt and Δf) for node grouping in the two hyper-views. Both parameters vary noticeably during continual learning, reflecting the adaptive behavior of RL-AHC. In contrast, *w/o RL-AHC* in Table 3 fixes Δt and Δf, leading to clear performance degradation. These results highlight the importance of adaptive hyper-view construction for capturing evolving high-order fraud patterns.

5.5 RQ4: Efficiency

To evaluate the efficiency of MC-SFD, we measure the per-batch training time of MC-SFD and seven representative baselines in Figure 6 (DSLR+T and RL-GNN+T refer to DSLR+TGAT and RL-GNN+TGAT). The results show that MC-SFD achieves a per-batch training time comparable to that of SOTA baselines and is only slightly slower than TGAT. The efficiency stems from three factors. (1) SME employs single-layer aggregation for each view ($L = 1$), which reduces the EDB in Proposition 3.5 and lowers the computational cost. (2) RL-AHC constructs sparse hyper-views under the complexity reward ($|\mathcal{HG}^T|, |\mathcal{HG}^F| \ll |\mathcal{E}|$). (3) MPPG freezes SME during continual learning, updating lightweight prompt parameters. See Appendix G for a detailed complexity analysis.

6 Conclusion

We formalize high-order fraud patterns in streaming transaction networks and theoretically reveal that their detection is constrained by the exponential amplification of embedding drift bounds (EDB) in existing methods. Motivated by this insight, we propose MC-SFD, a multi-view hypergraph continual learning model that captures high-order fraud via hypergraph modeling and effectively reduces EDB. Specifically, MC-SFD incorporates an RL-based Adaptive Hypergraph Constructor (RL-AHC) to dynamically evolve hypergraph structures and a Multi-view Personalized Prompt Generator (MPPG) that enables efficient adaptation with minimal parameter updates. We further use LLMs in RL-AHC for policy initialization and in MPPG for semantic enhancement. Across four streaming fraud detection datasets, MC-SFD consistently outperforms 23 state-of-the-art baselines, with AUC gains ranging from 1.63% to 3.71%.

Broader Impact. MC-SFD improves the detection of evolving high-order collusive fraud in streaming transaction networks, which can enhance financial system robustness and reduce economic losses. Beyond fraud detection, Proposition 3.5 provides the first theoretical analysis of high-order graph patterns under continual learning, offering a useful foundation for broader graph continual learning research and method design.

AI USE STATEMENT

Generative AI was used to adapt the manuscript's LaTeX source to the official ICLR 2027 template. This use was limited to LaTeX formatting and template compatibility, and it did not generate or revise the paper's scientific claims, experimental results, or conclusions. The authors remain responsible for the final manuscript.

ETHICS STATEMENT

This work uses three public datasets and one private banking dataset (Credit), as described in Section 5.1 and Appendix H. All public datasets used in our experiments have been anonymized. We do not redistribute Credit, which remains subject to the data provider's access and use restrictions. The LLM components are deployed locally or with equivalent privacy safeguards to avoid disclosing identifying records to third-party services. Overall, beyond the data-use considerations discussed above, this study does not involve human-subject experiments, the collection of personally identifiable information, or other direct ethical concerns.

REPRODUCIBILITY STATEMENT

The method and training objectives are specified in Section 4, and the experimental protocol, chronological splits, metrics, model choices, hyperparameters, computing environment, and repeated-run reporting are given in Section 5.1. The appendix provides the complete proofs (Appendix B), training procedure (Appendix D), design details of SME and RL-AHC modules (Appendices E and F), complexity analysis (Appendix G), dataset and baseline descriptions (Appendices H and I), sensitivity analyses, and LLM prompt templates (Appendix M). An anonymized implementation, including configuration files and data-processing and evaluation scripts, will be available at `https://anonymous.4open.science/r/MC-SFD-4443/README.md`. Although the private Credit dataset cannot be redistributed, we will provide its expected schema and preprocessing instructions. The remaining datasets can be obtained from the sources cited in Appendix H.

REFERENCES

Alessia Antelmi, Gennaro Cordasco, Mirko Polato, Vittorio Scarano, Carmine Spagnuolo, and Dingqi Yang. A survey on hypergraph representation learning. *ACM Computing Surveys*, 56(1):1–38, 2023.

Jie Cai, Xin Wang, Chaoyu Guan, Yateng Tang, Jin Xu, Bin Zhong, and Wenwu Zhu. Multimodal continual graph learning with neural architecture search. In *Proceedings of the ACM Web Conference 2022*, pp. 1292–1300, 2022.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*, 12(2):475–483, 2026. doi: 10.1109/TBDATA.2025.3627488.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 353–364, 2024.

Dawei Cheng, Yao Zou, Sheng Xiang, and Changjun Jiang. Graph neural networks for financial fraud detection: a review. *Frontiers of Computer Science*, 19(9):199609, 2025.

Seungyoon Choi, Wonjoong Kim, Sungwon Kim, Yeonjun In, Sein Kim, and Chanyoung Park. Dslr: Diversity enhancement and structure learning for rehearsal-based graph continual learning. In *Proceedings of the ACM Web Conference 2024*, pp. 733–744, 2024.

Yi Fang, Dongzhe Fan, Daochen Zha, and Qiaoyu Tan. Gaugllm: Improving graph contrastive learning for text-attributed graphs with large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 747–758, 2024.

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 3558–3565, 2019.

Yifan Feng, Yifan Zhang, Shihui Ying, Shaoyi Du, and Yue Gao. Kernelized hypergraph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(10):8938–8954, 2025a. doi: 10.1109/TPAMI.2025.3585179.

ZhengZhao Feng, Rui Wang, TianXing Wang, Mingli Song, Sai Wu, and Shuibing He. A comprehensive survey of dynamic graph neural networks: Models, frameworks, benchmarks, experiments and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 2025b.

Yue Gao, Yifan Feng, Shuyi Ji, and Rongrong Ji. Hgnn+: General hypergraph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3181–3199, 2022.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ibrahim Y Hafez, Ahmed Y Hafez, Ahmed Saleh, Amr A Abd El-Mageed, and Amr A Abohany. A systematic review of ai-enhanced techniques in credit card fraud detection. *Journal of Big Data*, 12(1):6, 2025.

Venus Haghighi, Behnaz Soltani, Nasrin Shabani, Jia Wu, Yang Zhang, Lina Yao, Quan Z Sheng, and Jian Yang. Tropical: Transformer-based hypergraph learning for camouflaged fraudster detection. In *2024 IEEE International Conference on Data Mining (ICDM)*, pp. 121–130. IEEE, 2024.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Thi Kieu Khanh Ho, Ali Karami, and Narges Armanfard. Graph anomaly detection in time series: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

Shuyi Ji, Yifan Feng, Donglin Di, Shihui Ying, and Yue Gao. Mode hypergraph neural network. *IEEE Transactions on Neural Networks and Learning Systems*, 36(8):13900–13912, 2025. doi: 10.1109/TNNLS.2025.3542176.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

Samira Khodabandehlou and Alireza Hashemi Golpayegani. Fifraud: unsupervised financial fraud detection in dynamic graph streams. *ACM Transactions on Knowledge Discovery from Data*, 18 (5):1–29, 2024.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Geon Lee, Fanchen Bu, Tina Eliassi-Rad, and Kijung Shin. A survey on hypergraph mining: Patterns, tools, and generators. *ACM Computing Surveys*, 57(8):1–36, 2025.

Hanjie Li, Changsheng Li, Kaituo Feng, Ye Yuan, Guoren Wang, and Hongyuan Zha. Robust knowledge adaptation for dynamic graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):6920–6933, 2024a.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. A survey of graph meets large language model: progress and future directions. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 8123–8131, 2024b.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One for all: Towards training one graph model for all classification tasks. In *The Twelfth International Conference on Learning Representations*, 2024.

Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Pick and choose: a gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the web conference 2021*, pp. 3168–3177, 2021.

Soroor Motie and Bijan Raahemi. Financial fraud detection using graph neural networks: A systematic review. *Expert Systems with Applications*, 240:122156, 2024.

Rui Ou, Kun Zhu, Nana Zhang, Jiangtong Li, Chaochao Chen, Yuhua Xu, and Changjun Jiang. Targeting borderline fraudsters: Multi-view hypergraph fraud detection with llm-guided contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 15591–15599, 2026a.

Rui Ou, Kun Zhu, Nana Zhang, MengChu Zhou, and Changjun Jiang. Camfd: Semi-supervised camouflage-aware fraud detection based on dynamic graphs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 56(2):1408–1422, 2026b. doi: 10.1109/TSMC.2025.3648571.

Junjun Pan, Yixin Liu, Xin Zheng, Yizhen Zheng, Alan Wee-Chung Liew, Fuyi Li, and Shirui Pan. A label-free heterophily-guided approach for unsupervised graph fraud detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 12443–12451, 2025.

Victor M Panaretos and Yoav Zemel. Statistical aspects of wasserstein distances. *Annual review of statistics and its application*, 6(1):405–431, 2019.

Hezhe Qiao, Hanghang Tong, Bo An, Irwin King, Charu Aggarwal, and Guansong Pang. Deep graph anomaly detection: A survey and new perspectives. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Dazhong Shen, Chuan Qin, Qi Zhang, Hengshu Zhu, and Hui Xiong. Handling over-smoothing and over-squashing in graph convolution with maximization operation. *IEEE Transactions on Neural Networks and Learning Systems*, 36(5):8743–8756, 2024.

Lei Song, Jiaxing Li, Qinghua Si, Shihan Guan, and Youyong Kong. Exploring rationale learning for continual graph learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 20540–20548, 2025.

Junwei Su, Difan Zou, Zijun Zhang, and Chuan Wu. Towards robust graph incremental learning on evolving graphs. In *International Conference on Machine Learning*, pp. 32728–32748. PMLR, 2023.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 491–500, 2024.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Sheng Tian, Jihai Dong, Jintang Li, Wenlong Zhao, Xiaolong Xu, Baokun Wang, Bowen Song, Changhua Meng, Tianyi Zhang, and Liang Chen. Sad: semi-supervised anomaly detection on dynamic graphs. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 2306–2314, 2023.

Zonggui Tian, Du Zhang, and Hong-Ning Dai. Continual learning on graphs: A survey. *arXiv preprint arXiv:2402.06330*, 2024.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Duo Wang, Yuan Zuo, Fengzhi Li, and Junjie Wu. Llms as zero-shot graph learners: Alignment of gnn representations with llm token embeddings. *Advances in neural information processing systems*, 37:5950–5973, 2024a.

Fuli Wang, Karelia Pena-Pena, Wei Qian, and Gonzalo R Arce. T-hypergnns: Hypergraph neural networks via tensor representations. *IEEE Transactions on Neural Networks and Learning Systems*, 36(3):5044–5058, 2024b.

Junshan Wang, Wenhao Zhu, Guojie Song, and Liang Wang. Streaming graph neural networks with generative replay. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 1878–1888, 2022.

Sheng Xiang, Guibin Zhang, Dawei Cheng, and Ying Zhang. Enhancing attribute-driven fraud detection with risk-aware graph representation. *IEEE Transactions on Knowledge and Data Engineering*, 37(5):2501–2512, 2025. doi: 10.1109/TKDE.2025.3543887.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

Fan Xu, Nan Wang, Hao Wu, Xuezhi Wen, Xibin Zhao, and Hai Wan. Revisiting graph-based fraud detection in sight of heterophily and spectrum. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 9214–9222, 2024.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Shuo Yu, Yingbo Wang, Ruolin Li, Guchun Liu, Yanming Shen, Shaoxiong Ji, Bowen Li, Fengling Han, Xiuzhen Zhang, and Feng Xia. Graph2text or graph2token: A perspective of large language models for graph learning. *ACM Transactions on Information Systems*, 44(3):1–49, 2026.

Qi Yuan, Yang Liu, Yateng Tang, Xinhuan Chen, Xuehao Zheng, Qing He, and Xiang Ao. Dynamic graph learning with static relations for credit risk assessment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pp. 13133–13141, 2025.

Mengmei Zhang, Mingwei Sun, Peng Wang, Shen Fan, Yanhu Mo, Xiaoxiao Xu, Hong Liu, Cheng Yang, and Chuan Shi. Graphtranslator: Aligning graph model to large language model for open-ended tasks. In *Proceedings of the ACM Web Conference 2024*, pp. 1003–1014, 2024a.

Peiyan Zhang, Yuchen Yan, Chaozhuo Li, Senzhang Wang, Xing Xie, Guojie Song, and Sunghun Kim. Continual learning on dynamic graphs via parameter isolation. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pp. 601–611, 2023.

Xikun Zhang, Dongjin Song, and Dacheng Tao. Continual learning on graphs: Challenges, solutions, and opportunities. *arXiv preprint arXiv:2402.11565*, 2024b.

Mingsheng Zheng, Junwei Zhang, Changshuai Zhan, Xinyu Ren, and Shuai Lü. Proximal policy optimization with reward-based prioritization. *Expert Systems with Applications*, 283:127659, 2025.

Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 4714–4722, 2021.

Yun Zhu, Yaoke Wang, Haizhou Shi, and Siliang Tang. Efficient tuning and inference for large language models on textual graphs. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 5734–5742, 2024.

Yao Zou and Dawei Cheng. Effective high-order graph representation learning for credit card fraud detection. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 7581–7589, 2024.

A JUSTIFICATION OF TWO SIMPLIFICATIONS

In Section 3, we introduce two simplifications for tractable drift-propagation analysis: (1) we focus on node (user) feature drift while omitting explicit edge-feature drift, and (2) we adopt GCN as a representative message-passing GNN. Here, we justify these assumptions by showing that (i) edge-feature drift can be equivalently absorbed into the structural drift term $\Delta\mathbf{A}$ through re-parameterization, and (ii) many GNN variants share the same algebraic formulation as GCN with a re-parameterized aggregation matrix $\mathbf{A}$.

A.1 FOCUSING ON NODE FEATURE DRIFT

Recall that a transaction graph at task i is $\mathcal{G}_i = (\mathcal{N}_i, \mathcal{E}_i, \mathbf{X}_{\mathcal{N},i}, \mathbf{X}_{\mathcal{E},i})$, where $\mathbf{X}_{\mathcal{N},i}$ and $\mathbf{X}_{\mathcal{E},i}$ denote node (user) and edge (transaction) features, respectively. In Section 3, we focus on node feature drift $\Delta\mathbf{X}_{i+1} = \Delta\mathbf{X}_{\mathcal{N},i+1} = \mathbf{X}_{\mathcal{N},i+1} - \mathbf{X}_{\mathcal{N},i}$. Below we show that, for common edge-aware message-passing schemes, edge-feature drift $\Delta\mathbf{X}_{\mathcal{E},i+1}$ can be equivalently re-parameterized as the drift of the re-parameterized aggregation matrix $\Delta\mathbf{A}_{i+1}$. Many edge-aware GNNs incorporate edge features by assigning edge-level nonnegative scalar gates $g(\mathbf{e}_{i,uv})$ and reweighting neighbor messages accordingly. The attention coefficients $\alpha_{u,v}$ in TGAT (Xu et al., 2020) are typical examples of such scalar gating. Let $\mathbf{A}_i$ denote the adjacency matrix induced purely by topology. The re-parameterized matrix is:

$$\tilde{\mathbf{A}}_i(u,v) = \mathbf{A}_i(u,v) \cdot g(\mathbf{e}_{i,uv}) \,. \tag{23}$$

Using Eq. (23), the layer update can be written in the same algebraic form as Eq. (1):

$$\mathbf{Z}_i^l = \sigma\Big(\tilde{\mathbf{A}}_i \, \mathbf{Z}_i^{l-1}\mathbf{W}\Big) \,. \tag{24}$$

Therefore, edge-feature drift $\Delta\mathbf{X}_{\mathcal{E},i+1}$ can be treated as part of the structural drift term $\Delta\mathbf{A}_{i+1}$ in Proposition 3.5.

A.2 USING GCN AS A REPRESENTATIVE GNN

We use GCN as a representative GNN because it has the simplest matrix form, as shown in Eq. (1). Many message-passing GNN variants can be written in the same form by replacing $\mathbf{A}_i$ with a dynamic data-dependent adjacency/aggregation matrix. For example, TGAT (Xu et al., 2020) and other attention-based GNNs compute attention coefficients $\alpha_{u,v}$ to assign a learnable weight to each edge, which is equivalent to constructing a dynamic adjacency matrix, and the message passing can still be written in the same form as Eq. (1). Consequently, Proposition 3.5, which is derived under the GCN formulation, is not restricted to vanilla GCN but can be extended to many message-passing GNNs.

B PROOF OF PROPOSITION 3.5

Proof. At task i, let $\mathbf{A}_i$ denote the normalized adjacency matrix and let $\mathbf{X}_i$ denote the node feature matrix. For $l = 0, \ldots, L$, $\mathbf{Z}_i^l$ denotes the node representation at layer l. And the node embedding is the representation at the final layer $\mathbf{Z}_i^L$.

Given a message-passing GNN defined by Eq. (1), the forward computation at task i can be formalized as:

$$\mathbf{Z}_i^0 = \mathbf{X}_i, \qquad \mathbf{Z}_i^l = \sigma\big(\mathbf{A}_i\mathbf{Z}_i^{l-1}\mathbf{W}\big) \,, \;\; l = 1, \ldots, L. \tag{25}$$

When the transaction graph evolves to task $i+1$, the structural and feature drift are defined as:

$$\Delta\mathbf{A}_{i+1} = \mathbf{A}_{i+1} - \mathbf{A}_i, \qquad \Delta\mathbf{X}_{i+1} = \mathbf{X}_{i+1} - \mathbf{X}_i, \tag{26}$$

which induce layer-wise representation drift:

$$\Delta\mathbf{Z}_{i+1}^l = \mathbf{Z}_{i+1}^l - \mathbf{Z}_i^l, \;\; l = 0, \ldots, L. \tag{27}$$

In particular, the embedding drift is denoted as $\Delta\mathbf{Z}_{i+1}^L$.

For any layer $l \in \{1, \ldots, L\}$, by the L_σ-Lipschitz continuity of $\sigma(\cdot)$:

$$\|\Delta \mathbf{Z}_{i+1}^{l}\|_F \leq L_\sigma \left\|\mathbf{A}_{i+1}\mathbf{Z}_{i+1}^{l-1}\mathbf{W} - \mathbf{A}_i\mathbf{Z}_i^{l-1}\mathbf{W}\right\|_F. \tag{28}$$

Adding and subtracting $\mathbf{A}_{i+1}\mathbf{Z}_i^{l-1}\mathbf{W}$ and applying the triangle inequality yield:

$$\begin{aligned}\|\Delta \mathbf{Z}_{i+1}^{l}\|_F &\leq L_\sigma\Big(\|\mathbf{A}_{i+1}\left(\mathbf{Z}_{i+1}^{l-1} - \mathbf{Z}_i^{l-1}\right)\mathbf{W}\|_F + \|\left(\mathbf{A}_{i+1} - \mathbf{A}_i\right)\mathbf{Z}_i^{l-1}\mathbf{W}\|_F\Big) \\ &= L_\sigma\Big(\|\mathbf{A}_{i+1}\Delta \mathbf{Z}_{i+1}^{l-1}\mathbf{W}\|_F + \|\Delta\mathbf{A}_{i+1}\mathbf{Z}_i^{l-1}\mathbf{W}\|_F\Big).\end{aligned} \tag{29}$$

Using $\|\mathbf{AZW}\|_F \leq \|\mathbf{A}\|_{\mathrm{sp}}\|\mathbf{Z}\|_F\|\mathbf{W}\|_{\mathrm{sp}}$ and the bounds $\|\mathbf{A}_{i+1}\|_{\mathrm{sp}} \leq \beta$ and $\|\mathbf{W}\|_{\mathrm{sp}} \leq \alpha$, we obtain:

$$\|\Delta \mathbf{Z}_{i+1}^{l}\|_F \leq \gamma\|\Delta \mathbf{Z}_{i+1}^{l-1}\|_F + L_\sigma\alpha\|\mathbf{Z}_i^{l-1}\|_F\,\|\Delta\mathbf{A}_{i+1}\|_{\mathrm{sp}}, \tag{30}$$

where $\gamma = L_\sigma\beta\alpha$. Moreover, $\Delta\mathbf{Z}_{i+1}^{0} = \mathbf{Z}_{i+1}^{0} - \mathbf{Z}_i^{0} = \mathbf{X}_{i+1} - \mathbf{X}_i = \Delta\mathbf{X}_{i+1}$.

Unrolling (30) from $l = 1$ to L yields:

$$\|\Delta \mathbf{Z}_{i+1}^{L}\|_F \leq (\gamma)^L\,\|\Delta\mathbf{X}_{i+1}\|_F + \sum_{l=1}^{L}(\gamma)^{L-l}\,L_\sigma\alpha\|\mathbf{Z}_i^{l-1}\|_F\,\|\Delta\mathbf{A}_{i+1}\|_{\mathrm{sp}}. \tag{31}$$

This matches Eq. (4) and completes the proof. □

Remark B.1. Eq. (30) highlights how drift propagates through message passing: each layer amplifies the previous-layer representation drift by $\gamma = L_\sigma\beta\alpha$ and adds an additional perturbation term proportional to $\|\Delta\mathbf{A}_{i+1}\|_{\mathrm{sp}}$. Unrolling the recursion yields Eq. (4): feature drift at the input is amplified as $(\gamma)^l\,\|\Delta\mathbf{X}_{i+1}\|_F$ at layer l (and $(\gamma)^L\,\|\Delta\mathbf{X}_{i+1}\|_F$ for the final embedding), while structural drift contributes at every layer and is likewise weighted by powers of γ. Therefore, when $\gamma > 1$, deeper aggregation (larger L), which is typically required to capture higher-order fraud patterns in existing methods, leads to rapidly growing embedding drift across tasks. This motivates our shallow aggregation designs: using a small L (e.g., $L = 1$ for SME in Section 4.1) suppresses the $(\gamma)^L$ amplification of feature drift and reduces the number of accumulated structural-drift terms, while high-order dependencies can still be effectively captured via multi-view hyperedges.

C γ IN STANDARD GNNS

Proposition 3.5 shows that the embedding drift bound (EDB) grows exponentially with GNN depth when $\gamma = L_\sigma\beta\alpha > 1$. To assess whether this expansive regime is relevant to standard GNNs, we measure an empirical amplification proxy for γ using the GNN backbones of all GNN-based baselines on four datasets. We also discuss the case where $\gamma \leq 1$.

Evaluated GNN Backbones. Table 4 reports the backbone GNNs of the compared baselines in our experiments. Note that five baselines do not employ a GNN backbone and therefore fall outside the scope of our discussion: RF and SVM are non-graph models, whereas HGNN, T-HyperGNN, and KHGNN are hypergraph-based models. To cover all compared baselines, we evaluate seven GNN backbones, including four widely used static GNN backbones (GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), and GIN (Xu et al., 2019)) and three temporal GNN backbones (TGAT (Xu et al., 2020), AdaDyGNN (Li et al., 2024a), and RGTAN (Xiang et al., 2025)).

Table 4: GNN backbones of the compared baselines. The slash denotes models without a GNN backbone.

Baseline	GNN backbone	Layers
RF	/	/
SVM	/	/
TGAT	TGAT	2
HGNN	/	/
SAD	TGAT	2
AdaDyGNN	AdaDyGNN	1
T-HyperGNN	/	/
OFA	GIN	2
TEA-GLM	GraphSAGE	2
GAugLLM	GCN, GraphSAGE, GAT, GIN	2
RGTAN	RGTAN	2
KHGNN	/	/
CamFD	TGAT	2
ER-GNN	GAT	2
SSRM	GCN, GraphSAGE, GAT, GIN	2
SGNN-GR	GraphSAGE	2
DSLR	GAT	2
RL-GNN	GCN, GraphSAGE, GAT, GIN	2
ER-GNN+TGAT	TGAT	2
SSRM+TGAT	TGAT	2
SGNN-GR+TGAT	TGAT	2
DSLR+TGAT	TGAT	2
RL-GNN+TGAT	TGAT	2

Empirical Amplification Proxy $\widehat{\gamma}$. The theoretical γ cannot be recovered exactly for modern GNNs with input-dependent attention, multiple message branches, gates, or discrete neighbor selection. We therefore define an empirical proxy that measures the maximum perturbation amplification of each realized layer.

For run r, task t, and layer l, let $\mathbf{Z}_{r,t}^{l-1}$ denote the input node-representation matrix of the layer. We first perform a forward pass and fix the resulting normalized aggregation coefficients, attention weights, gates, and selection masks. Conditional on these realized quantities, the message-propagation part of the layer defines a linear operator $\mathcal{T}_{r,t,l}$. Given an arbitrary perturbation $\Delta\mathbf{Z}^{l-1}$ of the layer input, $\mathcal{T}_{r,t,l}(\Delta\mathbf{Z}^{l-1})$ is the resulting perturbation at the output of message propagation, before the nonlinearities are applied.

Because γ in Proposition 3.5 uniformly bounds the propagation of any input perturbation, its empirical proxy should likewise capture the largest possible amplification over all perturbation directions. We therefore define

$$\|\mathcal{T}_{r,t,l}\|_{\max F} = \max_{\Delta\mathbf{Z}^{l-1}\neq\mathbf{0}} \frac{\left\|\mathcal{T}_{r,t,l}(\Delta\mathbf{Z}^{l-1})\right\|_F}{\|\Delta\mathbf{Z}^{l-1}\|_F}. \tag{32}$$

$\|\mathcal{T}_{r,t,l}\|_{\max F}$ is the smallest constant c such that $\|\mathcal{T}_{r,t,l}(\Delta\mathbf{Z}^{l-1})\|_F \le c\|\Delta\mathbf{Z}^{l-1}\|_F$ for every input perturbation. The ratio removes the effect of perturbation scale, while the maximum selects the most strongly amplified direction. We use the Frobenius norm because both perturbations are node-representation matrices and this norm measures their overall magnitude across all nodes and feature channels.

Directly solving the maximization in Eq. (32) is computationally infeasible for large GNN layers. Since $\mathcal{T}_{r,t,l}$ is linear under the fixed propagation quantities, we use power iteration to progressively refine a test perturbation toward the direction most strongly amplified by $\mathcal{T}_{r,t,l}$. Starting from a

random input perturbation $\mathbf{V}^{(0)}$ with $\|\mathbf{V}^{(0)}\|_F = 1$, we repeat the following two updates:

$$\mathbf{U}^{(k)} \leftarrow \frac{\mathcal{T}_{r,t,l}(\mathbf{V}^{(k)})}{\|\mathcal{T}_{r,t,l}(\mathbf{V}^{(k)})\|_F}, \tag{33}$$

$$\mathbf{V}^{(k+1)} \leftarrow \frac{\mathcal{T}^*_{r,t,l}(\mathbf{U}^{(k)})}{\|\mathcal{T}^*_{r,t,l}(\mathbf{U}^{(k)})\|_F}, \tag{34}$$

The first update propagates the current input perturbation through the layer and normalizes the resulting output direction. The second update maps this output direction back to the input space and again normalizes it, producing the next input direction. The normalizations retain only the directions and prevent their magnitudes from accumulating across iterations. Here, $\mathcal{T}^*_{r,t,l}$ denotes this reverse linear operation, which plays the same role as the transpose of an ordinary matrix and is characterized by $\langle \mathcal{T}_{r,t,l}(\mathbf{V}), \mathbf{U}\rangle_F = \langle \mathbf{V}, \mathcal{T}^*_{r,t,l}(\mathbf{U})\rangle_F$. Together, the two updates apply $\mathcal{T}^*_{r,t,l}\mathcal{T}_{r,t,l}$ to the input direction, so directions with stronger amplification are progressively reinforced. After convergence, $\mathbf{V}^{(K)}$ approximates the unit-norm input direction with the largest amplification, and we can estimate $\|\mathcal{T}_{r,t,l}\|_{\max F}$ as:

$$\|\mathcal{T}_{r,t,l}\|_{\max F} \approx \left\|\mathcal{T}_{r,t,l}(\mathbf{V}^{(K)})\right\|_F. \tag{35}$$

This procedure only requires forward applications of $\mathcal{T}_{r,t,l}$ and reverse applications of $\mathcal{T}^*_{r,t,l}$, without constructing an explicit matrix representation of the operator.

We next account for the nonlinear activation and derive the layer-wise empirical amplification proxy:

$$\widehat{\gamma}_{r,t,l} = L_{\sigma,l}\,\|\mathcal{T}_{r,t,l}\|_{\max F}. \tag{36}$$

Relationship Between $\widehat{\gamma}$ and γ. We first consider the simplified GNN in Eq. (1), for which the message-propagation operator is

$$\mathcal{T}(\Delta\mathbf{Z}) = \mathbf{A}\Delta\mathbf{Z}\mathbf{W}. \tag{37}$$

For every input perturbation,

$$\|\mathcal{T}(\Delta\mathbf{Z})\|_F \le \|\mathbf{A}\|_{\mathrm{sp}}\,\|\mathbf{W}\|_{\mathrm{sp}}\,\|\Delta\mathbf{Z}\|_F, \tag{38}$$

where $\|\cdot\|_{\mathrm{sp}}$ denotes the matrix spectral norm, i.e., its largest singular value. This bound is attained when $\Delta\mathbf{Z}$ is aligned with the leading singular directions of $\mathbf{A}$ and $\mathbf{W}$. Therefore,

$$\|\mathcal{T}\|_{\max F} = \|\mathbf{A}\|_{\mathrm{sp}}\,\|\mathbf{W}\|_{\mathrm{sp}}, \qquad \widehat{\gamma} = L_\sigma\,\|\mathbf{A}\|_{\mathrm{sp}}\,\|\mathbf{W}\|_{\mathrm{sp}}. \tag{39}$$

Therefore, for the simplified GNN, $\widehat{\gamma}$ exactly recovers the tight theoretical amplification factor γ: setting $\beta = \|\mathbf{A}\|_{\mathrm{sp}}$ and $\alpha = \|\mathbf{W}\|_{\mathrm{sp}}$ in Proposition 3.5 gives $\widehat{\gamma} = L_\sigma\beta\alpha = \gamma$.

For modern input-dependent architectures, $\widehat{\gamma}$ instead describes the amplification conditional on the propagation quantities observed during the forward pass. For example, consider an attention layer $\mathcal{F}(\mathbf{Z}) = \sigma(\mathbf{P}(\mathbf{Z})\mathbf{Z}\mathbf{W})$, whose propagation matrix $\mathbf{P}(\mathbf{Z})$ depends on its input. Around the observed input $\mathbf{Z}_*$, its first-order output perturbation contains two components:

$$\Delta\mathcal{F} \approx D_\sigma\left[\underbrace{\mathbf{P}(\mathbf{Z}_*)\Delta\mathbf{Z}\mathbf{W}}_{\text{direct propagation}} + \underbrace{\Delta\mathbf{P}\,\mathbf{Z}_*\mathbf{W}}_{\text{change in attention}}\right], \tag{40}$$

where D_σ is the local action of the activation and $\Delta\mathbf{P}$ is the change in the attention matrix caused by $\Delta\mathbf{Z}$. After fixing $\mathbf{P}(\mathbf{Z}_*)$, $\mathcal{T}$ captures the direct-propagation component. By the definition of $\|\mathcal{T}\|_{\max F}$ and the L_σ-Lipschitz continuity of the activation,

$$\|D_\sigma[\mathcal{T}(\Delta\mathbf{Z})]\|_F \le L_\sigma\,\|\mathcal{T}(\Delta\mathbf{Z})\|_F \le \widehat{\gamma}\,\|\Delta\mathbf{Z}\|_F. \tag{41}$$

The proxy does not capture the second component, nor analogous changes in input-dependent gates or selection masks. Nevertheless, by consistently quantifying direct message-propagation amplification under the coefficients realized during inference, $\widehat{\gamma}$ remains a useful diagnostic of expansive trends across layers, checkpoints, backbones, and datasets.

Experimental Setup. For every backbone, we follow the architecture and hyperparameter settings in its original paper and official implementation, including the number of message-passing layers. We evaluate every backbone on all four datasets using the data splits in Section 5.1, and collect the checkpoint after each of the 10 continual tasks in each of the 10 repeated runs (e.g. 100 checkpoints in total).

For every layer and test mini-batch at a checkpoint, we record the realized propagation coefficients and run $K = 200$ power iterations from each of two independently initialized random perturbations. The forward update $\mathcal{T}_{r,t,l}(\mathbf{V})$ and the reverse update $\mathcal{T}^*_{r,t,l}(\mathbf{U})$ are computed with the corresponding sparse message-passing operations while these coefficients remain fixed. For input-dependent backbones, the resulting $\widehat{\gamma}$ can vary across test mini-batches because each mini-batch realizes a different propagation operator. We define the checkpoint-level $\widehat{\gamma}_{r,t,l}$ as the median of these mini-batch estimates, which is robust to a small number of atypical neighborhood structures or attention patterns.

Table 5: Results of $\widehat{\gamma}$ for 7 evaluated backbones across four datasets. Each cell reports the median checkpoint-level value followed by the percentage of $\widehat{\gamma} > 1$.

Backbone	Layer	S-FFSD	Credit	Wikipedia	Reddit
GCN	1	1.134 / 100%	1.369 / 100%	1.161 / 100%	1.228 / 100%
	2	1.139 / 100%	1.670 / 100%	1.481 / 100%	1.531 / 100%
GraphSAGE	1	1.085 / 93%	1.272 / 100%	1.128 / 100%	1.129 / 100%
	2	1.122 / 100%	1.754 / 100%	1.321 / 100%	2.054 / 100%
GAT	1	1.961 / 100%	2.187 / 100%	1.972 / 100%	1.945 / 100%
	2	1.984 / 100%	2.539 / 100%	2.537 / 100%	2.507 / 100%
GIN	1	2.766 / 100%	3.429 / 100%	2.784 / 100%	2.756 / 100%
	2	2.785 / 100%	4.395 / 100%	3.619 / 100%	3.538 / 100%
TGAT	1	1.677 / 100%	1.858 / 100%	1.643 / 100%	2.763 / 100%
	2	2.210 / 100%	2.318 / 100%	1.917 / 100%	4.392 / 100%
AdaDyGNN	1	2.192 / 100%	3.352 / 100%	3.830 / 100%	2.013 / 100%
RGTAN	1	1.684 / 100%	1.480 / 100%	1.601 / 98%	1.202 / 100%
	2	1.884 / 100%	1.957 / 100%	1.716 / 100%	1.484 / 100%

Experimental Results. Table 5 reports the 10-run results for all 7 evaluated backbones. Overall, all 52 settings have a median $\widehat{\gamma} > 1$, and 5,191 of the 5,200 checkpoint-level measurements (99.8%) satisfy $\widehat{\gamma} > 1$. For GCN, Eq. (39) shows that $\widehat{\gamma}$ recovers the tight theoretical amplification factor γ, and all 800 measurements across the GCN settings satisfy $\widehat{\gamma} = \gamma > 1$. Therefore, according to Proposition 3.5, the embedding drift bound (EDB) of these GCNs grows exponentially with GNN depth. For the other backbones, input-dependent operations make $\widehat{\gamma}$ a conditional approximation of the theoretical γ. Nevertheless, all 44 medians and 4,391 of the 4,400 checkpoint-level measurements (99.8%) satisfy $\widehat{\gamma} > 1$, suggesting that both classical and SOTA GNN backbones also tend to amplify the EDB as depth increases.

Overall, our results suggest that $\gamma > 1$ is common among standard GNN backbones. However, some architectural constraints such as weight-norm regularization may yield $\gamma < 1$, motivating the discussion below.

Discussion on the Case of $\gamma < 1$. Proposition 3.5 shows that the EDB grows exponentially with GNN depth when $\gamma > 1$. However, $\gamma < 1$ does not guarantee that the EDB decreases with L. Although the feature-drift term $\gamma^L \|\Delta \mathbf{X}_{i+1}\|_F$ decays with depth and earlier structural perturbations are attenuated by γ^{L-l}, each additional layer also introduces a new structural-drift term proportional to $L_\sigma \alpha \|\mathbf{Z}_i^{l-1}\|_F \|\Delta \mathbf{A}_{i+1}\|_{\mathrm{sp}}$. Therefore, the EDB can still increase with L when newly introduced structural drift outweighs the attenuation of previous terms. This is especially likely when a new task adds many edges or significantly changes edge features, thereby altering aggregation weights.

D ALGORITHM OVERVIEW

The overall training procedure of MC-SFD is summarized in Algorithm 1. In the first epoch of each streaming task, MC-SFD invokes an LLM agent to initialize the action and action space for hyper-view construction. In the subsequent epochs, RL-AHC optimizes the hyper-view construction policy via PPO, while MPPG generates personalized prompts for the three views and SME performs multi-view message passing for fraud prediction. In the parameter update stage, the first task ($i = 1$) jointly optimizes SME, MPPG, and the fraud decoder, while subsequent tasks freeze SME and update only lightweight components for efficient adaptation.

Algorithm 1 MC-SFD Training Procedure in Streaming Fraud Detection.

Input: streaming tasks $\mathcal{SG} = (\mathcal{G}_1, \ldots, \mathcal{G}_T)$, number of epochs E.
Output: trained MC-SFD parameters.
for $i = 1$ **to** T **do**
  // RL-AHC Init
  Build the initial RL state $s_{i,0}$ using graph statistics and the estimated drift via Eq. (51).
  Query the LLM to obtain the initial action and action space $(a_{i,0}, \mathcal{A}_i)$ via Eq. (9).
  Construct initial hyper-views $\mathcal{HG}^T_{i,0}, \mathcal{HG}^F_{i,0}$ according to $a_{i,0}$ via Eq. (13).
  // MPPG Init
  Pre-compute and cache semantic embedding $\mathbf{s}^v_u$ via Eq. (20).
  for $e = 1$ **to** E **do**
    // RL-AHC
    if $e > 1$ **then**
      Obtain the action $a_{i,e}$ from the RL agent via Eqs. (10)–(12).
      Construct hyper-views $\mathcal{HG}^T_{i,e}, \mathcal{HG}^F_{i,e}$ according to $a_{i,e}$ via Eq. (13).
    end if
    // MPPG
    for each view $v \in \{\mathcal{G}, \mathcal{HG}^T, \mathcal{HG}^F\}$ **do**
      Compute prototype weights $\boldsymbol{\alpha}^v_u$ via Eq. (21). *// personalized weighting*
      Compose and inject the personalized prompt into the node feature via Eq. (22).
    end for
    // SME
    for each view $v \in \{\mathcal{G}, \mathcal{HG}^T, \mathcal{HG}^F\}$ **do**
      Aggregate nodes into hyperedge messages $\mathbf{m}^v_h$ via Eqs. (45)(46).
      Aggregate $\mathbf{m}^v_h$ into view-specific node embeddings $\mathbf{Z}^v_{i,e}$ via Eqs. (47)(48).
    end for
    Fuse $\mathbf{Z}^{\mathcal{G}}_{i,e}, \mathbf{Z}^{\mathcal{HG}^T}_{i,e}, \mathbf{Z}^{\mathcal{HG}^F}_{i,e}$ to obtain the multi-view embedding $\mathbf{Z}_{i,e}$ via Eqs. (49)(50).
    Predict fraud scores $\hat{\mathcal{Y}}_{i,e}$ via Eq. (6). *// fraud decoder*
    // Update Parameters
    if $i = 1$ **then**
      Update SME, MPPG, and the fraud decoder via Eqs. (7)(8).
    else
      Update MPPG and the fraud decoder via Eqs. (7)(8) while freezing SME.
    end if
    Compute the reward $r_{i,e}$ via Eqs. (14)(15).
    Update policy and value networks $\mathcal{P}(\cdot), \mathcal{V}(\cdot)$ via Eqs. (16)–(18).
    Update state for next iteration via Eq. (51).
  end for
end for

E DESIGN DETAILS OF SHALLOW MULTI-VIEW ENCODER (SME)

At each task i, SME takes the transaction graph view $\mathcal{G}_i$ and two hyper-views $\mathcal{HG}^T_i, \mathcal{HG}^F_i$ as input, and outputs the multi-view user embedding $\mathbf{Z}_i$. Notably, the graph view $\mathcal{G}_i$ can be regarded as a special case of a hypergraph in which each hyperedge connects exactly two nodes. Therefore, the three views can be unified under a common hypergraph formulation with a shared node set $\mathcal{N}_i$, hyperedge

set $\mathcal{H}_i^v$, node features $\mathbf{X}_{\mathcal{N},i}$, hyperedge features $\mathbf{X}_{\mathcal{H},i}^v$, and incidence matrix $\mathbf{H}_i^v \in \{0,1\}^{|\mathcal{N}_i| \times |\mathcal{H}_i^v|}$, where:

$$\mathbf{H}_i^v(u,h) = \begin{cases} 1, & \text{if } u \in h, \\ 0, & \text{otherwise.} \end{cases} \tag{42}$$

When the context is clear, we omit the task index i for notational simplicity.

SME performs single-layer aggregation ($L = 1$) within each view and then fuses the three view-specific representations to obtain the multi-view embedding $\mathbf{Z}$. It consists of three hierarchical attention modules. (1) Intra-hyperedge attention aggregates connected nodes into hyperedge messages. (2) Inter-hyperedge attention aggregates incident hyperedges into view-level node representations. (3) Inter-view attention fuses the three view-specific representations into the multi-view embedding.

Basic attention. All attention layers use the following scaled dot-product attention module:

$$\text{Attn}(q,k,v) = \text{softmax}\left(QK^T/\sqrt{d}\right)V, \tag{43}$$

where $Q = \mathbf{W}_q \cdot q, K = \mathbf{W}_k \cdot k, V = \mathbf{W}_v \cdot v$; $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ are projection matrices that map the inputs into a d-dimensional latent space.

Relative time encoding. To effectively capture temporal information in SFD, we incorporate a relative time encoding $\Phi(\Delta t) \in \mathbb{R}^{d_\Phi}$:

$$\Phi(\Delta t) = [\cos(\omega_1 \Delta t), \sin(\omega_1 \Delta t), \ldots, \cos(\omega_{d_\Phi/2}\Delta t), \sin(\omega_{d_\Phi/2}\Delta t)], \tag{44}$$

where $\{\omega_k\}_{k=1}^{d_\Phi/2}$ are learnable frequencies. In practice, Δt is defined as the time gap between the current reference time of the target node and the timestamp associated with the neighbor.

(1) Intra-hyperedge attention (node→hyperedge). For a view v and a hyperedge $h \in \mathcal{H}^v$, let $\mathcal{N}_h$ denote the set of nodes incident to h. Intra-hyperedge attention aggregates connected nodes within each hyperedge into a hyperedge message conditioned on the hyperedge feature. Specifically, let $\mathbf{x}_h^v$ denote the feature of hyperedge h and $\mathbf{x}_u$ denote the feature of node u. We form a hyperedge query token and node key/value tokens as:

$$\mathbf{q}_h^v = [\mathbf{x}_h^v \mathbf{W}_{\mathcal{H}}^v \,||\, \Phi(0)], \qquad \mathbf{k}_{u\to h}^v = \mathbf{v}_{u\to h}^v = [\mathbf{x}_u \mathbf{W}_{\mathcal{N}}^v \,||\, \Phi(\Delta t_{u,h})], \tag{45}$$

where $\mathbf{W}_{\mathcal{H}}^v$ and $\mathbf{W}_{\mathcal{N}}^v$ are view-specific projection matrices and $\Delta t_{u,h}$ is the relative time gap associated with the incidence (u,h). The hyperedge message is then computed as:

$$\mathbf{m}_h^v = \text{Attn}\Big(q = \mathbf{q}_h^v,\ k = \{\mathbf{k}_{u\to h}^v\}_{u\in\mathcal{N}_h},\ v = \{\mathbf{v}_{u\to h}^v\}_{u\in\mathcal{N}_h}\Big). \tag{46}$$

(2) Inter-hyperedge attention (hyperedge→node). For a target node $u \in \mathcal{N}$, let $\mathcal{H}^v(u) = \{h \in \mathcal{H}^v : \mathbf{H}_i^v(u,h) = 1\}$ be the incident hyperedges in view v. Inter-hyperedge attention aggregates the incident hyperedge messages into the view-level node representation $\mathbf{z}_u^v$:

$$\mathbf{q}_u^v = [\mathbf{x}_u \mathbf{W}_{\mathcal{N}}^v \,||\, \Phi(0)], \qquad \mathbf{k}_{h\to u}^v = \mathbf{v}_{h\to u}^v = [\mathbf{m}_h^v \,||\, \Phi(\Delta t_{u,h})], \tag{47}$$

$$\mathbf{z}_u^v = \text{Attn}\Big(q = \mathbf{q}_u^v,\ k = \{\mathbf{k}_{h\to u}^v\}_{h\in\mathcal{H}^v(u)},\ v = \{\mathbf{v}_{h\to u}^v\}_{h\in\mathcal{H}^v(u)}\Big). \tag{48}$$

Stacking $\mathbf{z}_u^v$ over all $u \in \mathcal{N}$ yields the view-level representation matrix $\mathbf{Z}^v \in \mathbb{R}^{|\mathcal{N}|\times d}$.

(3) Inter-view attention (view fusion). Finally, for each node u, inter-view attention fuses the three view-specific representations into the multi-view embedding $\mathbf{z}_u$:

$$\beta_u^v = \mathbf{a}^\top \tanh\left(\mathbf{W}_v \mathbf{z}_u^v\right), \quad \alpha_u^v = \frac{\exp(\beta_u^v)}{\sum_{v'\in\{\mathcal{G},\mathcal{HG}^T,\mathcal{HG}^F\}} \exp(\beta_u^{v'})}, \tag{49}$$

$$\mathbf{z}_u = \sum_{v\in\{\mathcal{G},\mathcal{HG}^T,\mathcal{HG}^F\}} \alpha_u^v \mathbf{z}_u^v. \tag{50}$$

Stacking $\mathbf{z}_u$ over all $u \in \mathcal{N}$ yields the multi-view embedding $\mathbf{Z}$.

F Design Details of RL-based Adaptive Hyper-View Construction

F.1 State

At task i and training epoch e, we summarize the current transaction graph view $\mathcal{G}_i$ and the two constructed hyper-views $\mathcal{HG}_{i,e}^T, \mathcal{HG}_{i,e}^F$ into a compact state vector $s_{i,e}$.

$$s_{i,e} = \Big[\underbrace{|\mathcal{N}_i|, |\mathcal{E}_i|, \rho_i, \text{Drift}\,(\mathcal{G}_i, \mathcal{G}_{i-1})}_{\mathcal{G}_i}, \underbrace{|\mathcal{H}_{i,e}^T|, \text{Avg}(\ell_{i,e}^T), \text{Var}(\ell_{i,e}^T)}_{\mathcal{HG}_{i,e}^T}, \underbrace{|\mathcal{H}_{i,e}^F|, \text{Avg}(\ell_{i,e}^F), \text{Var}(\ell_{i,e}^F)}_{\mathcal{HG}_{i,e}^F}, \Big], \tag{51}$$

where ρ_i denotes the fraud ratio in task i, $\text{Drift}\,(\mathcal{G}_i, \mathcal{G}_{i-1})$ denotes the estimated embedding drift between the current and previous tasks, and $\text{Avg}(\ell_{i,e}^v)$ and $\text{Var}(\ell_{i,e}^v)$ are the average and variance of hyperedge sizes in hyper-view v at epoch e. Notably, the state in the transaction view is task-specific, whereas the hyper-view states are both task-specific and epoch-specific because the hyper-views are dynamically constructed during training.

Estimated embedding drift. To expose the level of drift to the RL agent, we measure the embedding drift for task i using the *Wasserstein distance* (Panaretos & Zemel, 2019) between node embeddings from the current and previous tasks.

$$\text{Drift}\,(\mathcal{G}_i, \mathcal{G}_{i-1}) = \text{Wasserstein}\left(\{\mathbf{z}_u^{(i-1,e_{\text{end}})}\}_{u\in\mathcal{N}_{i-1}}, \{\mathbf{z}_u^{(i,0)}\}_{u\in\mathcal{N}_{i-1}}\right), \tag{52}$$

where $\mathbf{z}_u^{(i-1,e_{\text{end}})}$ denotes the node embedding of user u at the last epoch of the previous task, and $\mathbf{z}_u^{(i,0)}$ denotes the embedding before the first epoch of the current task (i.e., with parameters inherited from task $i-1$). In practice, we implement $\text{Wasserstein}(\cdot)$ using the 100-sliced Wasserstein distance $SW^{(100)}(\cdot)$, which is computed between empirical distributions of node embeddings using 100 random one-dimensional projections:

$$SW^{(100)}(X,Y) = \frac{1}{100}\sum_{m=1}^{100} W\Big(\{\langle\boldsymbol{\theta}_m, x\rangle\}_{x\in X}, \{\langle\boldsymbol{\theta}_m, y\rangle\}_{y\in Y}\Big), \quad \boldsymbol{\theta}_m \sim \text{Unif}(\mathbb{S}^{d-1}), \tag{53}$$

where $\langle\cdot,\cdot\rangle$ denotes the inner product, d is the embedding dimension, $\boldsymbol{\theta}_m \in \mathbb{R}^d$ is a random projection direction, and $\text{Unif}(\mathbb{S}^{d-1})$ denotes the uniform distribution over the unit sphere $\mathbb{S}^{d-1} = \{\boldsymbol{\theta} \in \mathbb{R}^d : \|\boldsymbol{\theta}\|_2 = 1\}$. $W(X,Y)$ denotes the one-dimensional Wasserstein distance between the projected samples, which can be computed as:

$$W(X,Y) = \frac{1}{n}\sum_{k=1}^{n} \left|x_{(k)} - y_{(k)}\right|, \tag{54}$$

where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$, and $x_{(k)}$ and $y_{(k)}$ denote the k-th smallest elements of the sorted samples.

F.2 Action

RL-AHC uses a continuous action vector to control the granularity of hyper-view construction:

$$a = [\Delta t, \Delta f], \tag{55}$$

where Δt and Δf are the temporal and frequency tolerances used by the construction operator $\mathcal{C}(\cdot)$ in Eq. (13). Intuitively, larger tolerances lead to coarser grouping (larger hyperedges and denser hyper-views), while smaller tolerances yield finer grouping.

For each task i, the LLM-assisted initialization in Eq. (9) provides an initial action $a_{i,0}$ and a task-specific bounded action space:

$$\mathcal{A}_i = [\Delta t_{\min}, \Delta t_{\max}] \times [\Delta f_{\min}, \Delta f_{\max}]. \tag{56}$$

The sampled action is then clipped to this box-constrained action space, as in Eq. (12).

F.3 Policy Network (Actor)

The policy network $\mathcal{P}(\cdot)$ is implemented as a lightweight multi-layer perceptron (MLP) that maps the state $s_{i,e}$ to the parameters of a diagonal Gaussian distribution:

$$[\mu, \log \sigma] = \mathrm{MLP}_{\text{actor}}(s_{i,e}), \qquad \sigma = \mathrm{Softplus}(\log \sigma) + \varepsilon, \tag{57}$$

where $\mu \in \mathbb{R}^2$ and $\sigma \in \mathbb{R}^2_+$ correspond to the two action dimensions $[\Delta t, \Delta f]$. We then sample $\tilde{a} \sim \mathcal{N}(\mu, \mathrm{diag}(\sigma^2))$ and apply box clipping to obtain the final action a. In implementation, $\mathrm{MLP}_{\text{actor}}(\cdot)$ consists of a small feed-forward network with ReLU activations and two output heads (one for μ and one for $\log \sigma$).

F.4 Value Network (Critic)

The value network $\mathcal{V}(\cdot)$ is implemented as an MLP with the same input $s_{i,e}$ and a scalar output:

$$\mathcal{V}(s_{i,e}) = \mathrm{MLP}_{\text{critic}}(s_{i,e}) \in \mathbb{R}. \tag{58}$$

The critic MLP uses the same feed-forward design as the actor MLP but ends with a single linear output unit. It is trained using the value regression loss in Eq. (18), while the actor is trained using the clipped PPO objective in Eq. (17).

G Complexity Analysis

For the TGAT backbone, the per-epoch time complexity is $\mathcal{O}(|\mathcal{E}|c^L d^2)$, where $|\mathcal{E}|$ is the number of transaction edges, c is the sampled neighborhood size, L is the number of message-passing layers, and d is the hidden dimension (Xu et al., 2020).

MC-SFD introduces two additional hyper-views and applies SME over three views. Let

$$|\mathbf{E}| = |\mathcal{E}| + |\mathcal{H}^T| + |\mathcal{H}^F|, \tag{59}$$

where $|\mathcal{H}^T|$ and $|\mathcal{H}^F|$ denote the numbers of hyperedges in the time and frequency hyper-views. Since SME uses shallow encoding with $L = 1$, its per-epoch complexity is:

$$\mathcal{O}(|\mathbf{E}|c^L d^2) = \mathcal{O}(|\mathbf{E}|cd^2). \tag{60}$$

Although RL-AHC and MPPG both involve LLM-assisted components (i.e., hyper-view initialization in RL-AHC and semantic embedding generation in MPPG), these components can be pre-computed and cached, allowing both modules to remain computationally efficient during training. For RL-AHC, the dominant overhead is hyper-view construction and statistics update, which is linear in the multi-view size, i.e., $\mathcal{O}(|\mathbf{E}|)$. For MPPG, prompt composition with k prototypes introduces $\mathcal{O}(|\mathbf{E}|kd)$. Therefore, the overall per-epoch complexity is:

$$\mathcal{O}(|\mathbf{E}|cd^2 + |\mathbf{E}|kd + |\mathbf{E}|). \tag{61}$$

In practice, k is small and SME remains the dominant term, yielding $\mathcal{O}(|\mathbf{E}|cd^2)$.

The space complexity is dominated by node features and sparse multi-view structures, i.e., $\mathcal{O}(|\mathcal{N}|d + |\mathbf{E}|)$.

H Dataset Descriptions

This section briefly introduces the four streaming fraud detection datasets used in our experiments. Summary statistics are reported in Table 1.

S-FFSD (Xiang et al., 2025) is a synthetic financial transaction dataset based on real financial data for streaming fraud detection. In our formulation, nodes represent users, while each edge denotes a timestamped transaction between a pair of users. A positive edge corresponds to a fraudulent transaction.

Credit is a private real-world credit card transaction dataset collected from a major commercial bank in China for streaming fraud detection. Similar to S-FFSD, we model users as nodes and individual

transactions between user pairs as edges. A positive edge also corresponds to a fraudulent transaction. The ground-truth labels in Credit are verified by domain experts.

Wikipedia (Tian et al., 2023) is a dynamic web-editing dataset derived from the Wikipedia encyclopedia. Similar to the transaction datasets above, we model users as nodes and interactions as edges in a temporal graph. A positive edge indicates that a user is banned from editing because of malicious page edits.

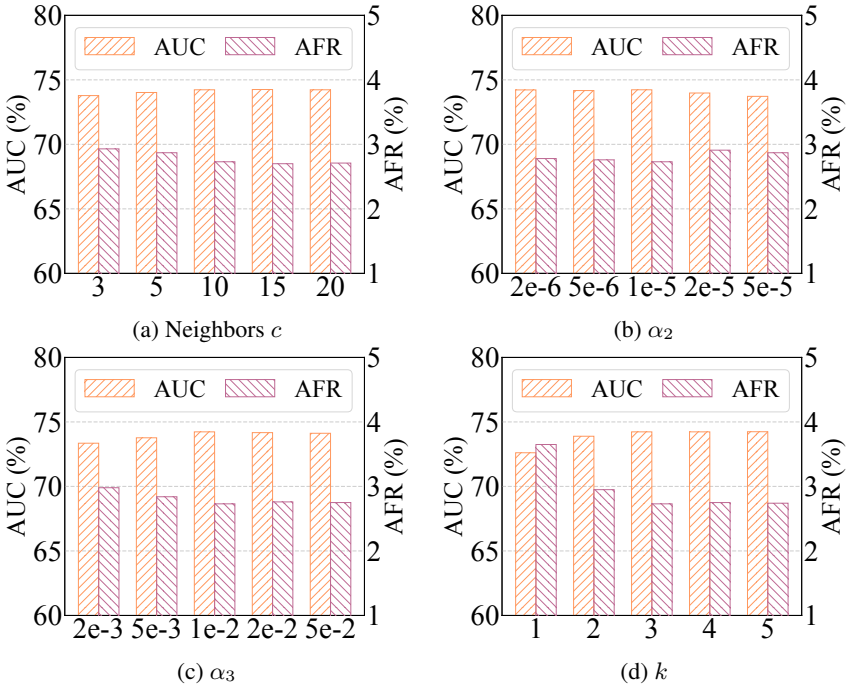
Reddit (Tian et al., 2023) is a dynamic posting dataset derived from the Reddit platform. Similar to the other datasets, we model users as nodes and interactions as edges in a temporal graph. A positive edge indicates that a user is banned from posting in a subreddit due to malicious posts.

For continual evaluation, all four datasets are chronologically divided into ten sequential tasks. Following prior work, S-FFSD uses a 60%-20%-20% train/validation/test split (Xiang et al., 2025), while Credit, Wikipedia, and Reddit use a 70%-15%-15% split (Tian et al., 2023).

I Baseline Descriptions

This section briefly introduces the compared baselines in Table 2, excluding the two traditional methods (RF and SVM).

- TGAT (Xu et al., 2020) is the GNN backbone of the proposed MC-SFD, which models temporal dependencies on dynamic graphs through time-aware attention. We report the results of both fully retrained and incrementally trained versions of TGAT in Table 2.
- HGNN (Feng et al., 2019) is a foundational work that incorporates hypergraphs to capture high-order relations in graph data.
- SAD (Tian et al., 2023) designs a semi-supervised anomaly detector via Gaussian distribution modeling.
- AdaDyGNN (Li et al., 2024a) improves the robustness of temporal GNNs through adaptive knowledge transfer under evolution.
- T-HyperGNN (Wang et al., 2024b) incorporates tensorized hypergraph representations for high-order structure learning.
- OFA (Liu et al., 2024) leverages large language models to train one graph model that can generalize across multiple classification tasks.
- TEA-GLM (Wang et al., 2024a) aligns GNN representations with LLM token embeddings for zero-shot graph learning.
- GAugLLM (Fang et al., 2024) enhances graph contrastive learning on text-attributed graphs with LLM guidance.
- RGTAN (Xiang et al., 2025) integrates risk-aware temporal graph modeling to capture fraud patterns in dynamic graphs.
- KHGNN (Feng et al., 2025a) introduces kernelized operators to improve hypergraph neural networks for high-order learning tasks.
- CamFD (Ou et al., 2026b) proposes a camouflage-aware semi-supervised fraud detector on dynamic graphs via consistency-sensitive contrastive learning.
- ER-GNN (Zhou & Cao, 2021) mitigates catastrophic forgetting in graph continual learning with sampling-based experience replay.
- SSRM (Su et al., 2023) improves the robustness of replay-based graph continual learning in streaming scenarios.
- SGNN-GR (Wang et al., 2022) generates random walk sequences to approximate the subgraph structure and develops the first generative replay method in graph continual learning.
- DSLR (Choi et al., 2024) combines diversity enhancement and structure learning for rehearsal-based graph continual learning.
- RL-GNN (Song et al., 2025) uses rationale learning to improve continual graph learning stability and transfer.



(a) Neighbors c (b) α_2

(c) α_3 (d) k

Figure 7: Parameter sensitivity on S-FFSD.

- ER-GNN+TGAT, SSRM+TGAT, SGNN-GR+TGAT, DSLR+TGAT, and RL-GNN+TGAT are the temporal counterparts of the five graph continual learning baselines, which employ TGAT as their GNN backbones.

J Parameter Sensitivity

We evaluate the parameter sensitivity of MC-SFD on S-FFSD in Figure 7. For a fair comparison, we vary one hyperparameter at a time while keeping all others at their default settings in Section 5.

For SME, Figure 7(a) varies the number of sampled neighbors c per view. A very small c may miss useful local context, while a large c introduces extra noise and computation. The model remains stable over a moderate range, and $c = 5$ achieves a good performance-efficiency trade-off.

For RL-AHC, Figures 7(b) and 7(c) show the effects of the complexity and stability coefficients (α_2 and α_3 in Eqs. (14) and (15)). When these coefficients are too small, the policy regularization is weak. And when they are too large, exploration is over-constrained. A moderate setting, $\alpha_2 = 10^{-5}$ and $\alpha_3 = 10^{-2}$, provides more reliable adaptation.

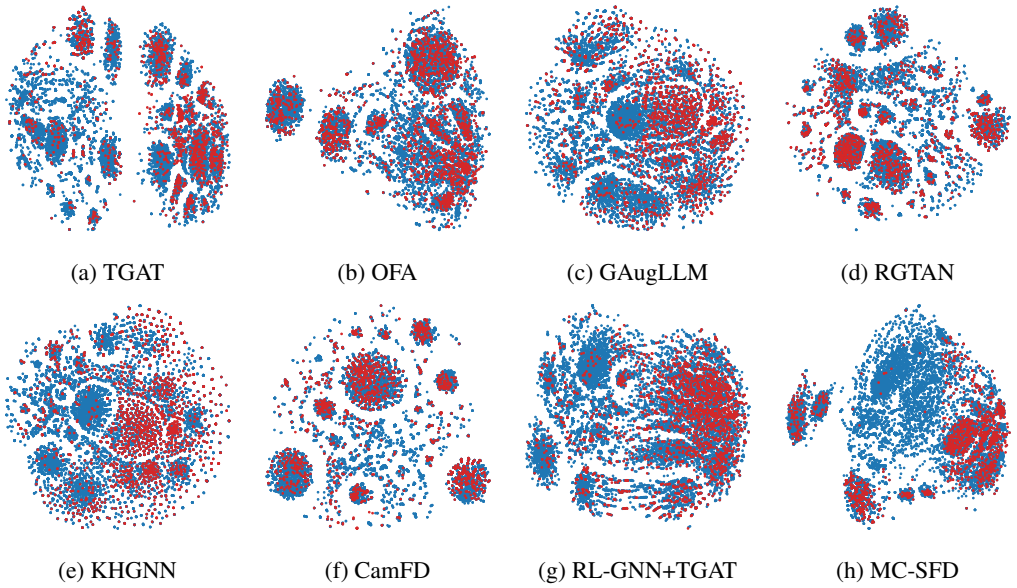
For MPPG, Figure 7(d) varies the number of prompt prototypes k in Eq. (19). Increasing k improves representation capacity at first, but gains quickly saturate and overhead grows. A small prototype bank ($k = 3$) is sufficient to maintain strong plasticity with lightweight updates.

K LLM Backbone Sensitivity

To evaluate the robustness of MC-SFD under different LLM backbones, we replace the default Llama-3.1-8B-Instruct (Grattafiori et al., 2024) with three alternatives: Mistral-7B-Instruct-v0.3 (Jiang et al., 2023), Qwen2.5-7B-Instruct (Team, 2024), and Qwen3-8B (Team, 2025). As shown in Table 6, MC-SFD achieves consistently strong performance across all four LLM backbones. The overall gaps are small across datasets, indicating that MC-SFD is not sensitive to a specific LLM choice.

Table 6: Backbone sensitivity of MC-SFD under different LLMs (mean±std).

Datasets	S-FFSD		Credit		Wikipedia		Reddit	
Metrics	AUC↑	AFR↓	AUC↑	AFR↓	AUC↑	AFR↓	AUC↑	AFR↓
Mistral-7B	73.71±1.14	2.84±0.76	89.93±1.26	1.30±0.78	85.34±0.96	-1.50±0.77	64.45±1.80	-1.42±0.61
Qwen2.5-7B	73.87±1.23	2.82±0.98	89.81±1.21	1.26±0.83	85.39±1.03	-1.47±0.82	64.43±1.83	-1.45±0.63
Qwen3-8B	74.18±0.89	2.76±0.81	90.11±1.16	1.29±0.80	85.48±0.87	-1.52±0.72	64.68±1.87	-1.46±0.51
Llama3.1-8B	74.23±0.95	2.73±0.73	90.08±1.07	1.27±0.71	85.61±0.92	-1.53±0.78	64.71±1.76	-1.50±0.57



(a) TGAT (b) OFA (c) GAugLLM (d) RGTAN

(e) KHGNN (f) CamFD (g) RL-GNN+TGAT (h) MC-SFD

Figure 8: Visualization results on S-FFSD dataset, where blue and red nodes denote benign and fraudulent transactions, respectively.

L VISUALIZATION

To visualize the fraud detection capability of different models, we present a 2D projection of the learned node embeddings of MC-SFD and seven representative baselines on the S-FFSD dataset using t-SNE (Van der Maaten & Hinton, 2008). As shown in Figure 8, MC-SFD exhibits a clearer separation between benign users and fraudsters, indicating stronger discriminative capability in the node embedding space. In addition, the embeddings produced by MC-SFD appear more compact within each class, while the overlap between the two classes is reduced compared with the baselines. This suggests that MC-SFD captures more informative structural and temporal patterns for distinguishing fraudulent behavior during the streaming learning process. The visualization results are also consistent with the quantitative performance reported in Table 2.

M PROMPT TEMPLATES

This section presents the text templates used in our experiments for the two LLM-assisted components of MC-SFD: initialization of hyper-view construction in RL-AHC and neighborhood semantic encoding in MPPG.

M.1 PROMPT FOR RL-AHC INITIALIZATION

At the first epoch of task i, RL-AHC queries the LLM to initialize the action $a_{i,0}$ and bounded action space $\mathcal{A}_i$ (Eq. (9)) using the current graph statistics, estimated embedding drift, and previous task's final construction parameters. The prompt template is provided below.

```
[Task Definition]
You are an expert in fraud detection and streaming graph modeling.

Dataset: <dataset_name>
Description:
<dataset_description>
The dataset is chronologically divided into 10 sequential graph tasks.

For the current task (task <task_index>), initialize the construction
    parameters for two complementary hyper-views derived from its
    original graph view to capture higher-order fraud patterns:
- Time hyper-view: groups users with temporally close interactions to
    capture short-term patterns.
- Frequency hyper-view: groups users with similar interaction
    frequencies to capture long-term patterns.

Using the current graph statistics, the previous task's hyper-view
    parameters, and the estimated embedding drift between tasks,
    determine:
1. The initial time tolerance delta_t and its exploration bounds.
2. The initial frequency tolerance delta_f and its exploration bounds.

Use the estimated embedding drift to determine how far the current
    parameters should deviate from the previous task's parameters.
The resulting bounds will define the search space in which reinforcement
     learning subsequently optimizes the tolerances.

Interpret the tolerances using the following dataset-specific
    definitions and units:
- delta_t: <temporal_tolerance_convention>
- delta_f: <frequency_tolerance_convention>

Return only a JSON object containing six numeric fields and a brief
    analysis. All numeric values must be finite and use the stated units.
     For each tolerance, the initial value must lie between its lower
    and upper bounds. In the analysis, briefly explain how the current
    graph statistics, previous parameters, and estimated embedding drift
     informed the initial values and exploration bounds.
{
  "delta_t": <initial_temporal_tolerance>,
  "delta_f": <initial_frequency_tolerance>,
  "delta_t_min": <temporal_lower_bound>,
  "delta_t_max": <temporal_upper_bound>,
  "delta_f_min": <frequency_lower_bound>,
  "delta_f_max": <frequency_upper_bound>,
  "analysis": "<brief_analysis_and_justification>"
}

[Input]
Serialized state of the current graph task:
Task index: <task_index>
```

Table 7: Mapping between RL-AHC prompt fields and model quantities.

Input fields	Corresponding quantities
`task_index`	Task index i (starting from 0)
`num_nodes`, `num_edges`	$\lvert\mathcal{N}_i\rvert$, $\lvert\mathcal{E}_i\rvert$
`fraud_ratio`, `embedding_drift`	ρ_i, $\mathrm{Drift}(\mathcal{G}_i, \mathcal{G}_{i-1})$
`previous_delta_t`	$\Delta t_{i-1}^{\mathrm{end}}$, or 0 for task 0
`previous_delta_f`	$\Delta f_{i-1}^{\mathrm{end}}$, or 0 for task 0

```
Number of nodes: <num_nodes>
Number of edges: <num_edges>
Fraud edge ratio: <fraud_ratio>
Estimated embedding drift from the previous task: <embedding_drift>
Temporal tolerance at the end of the previous task (obtained through RL
    exploration, or 0 for task 0): <previous_delta_t>
Frequency tolerance at the end of the previous task (obtained through RL
     exploration, or 0 for task 0): <previous_delta_f>
```

The placeholders are filled with the dataset context (Appendix H) and task-specific inputs summarized in Table 7. The numeric JSON fields define $a_{i,0} = [\Delta t, \Delta f]$ and $\mathcal{A}_i = [\Delta t_{\min}, \Delta t_{\max}] \times [\Delta f_{\min}, \Delta f_{\max}]$. The former constructs the initial hyper-views, and the latter bounds subsequent PPO exploration.

M.2 Prompt for Multi-view Semantic Encoding

MPPG uses the LLM to encode each target node's local context in the initial transaction, time, and frequency views into semantic embeddings $\mathbf{s}_u^v$ (Eq. (20)). The prompt template is provided below.

```
[Task Definition]
You are an expert in fraud detection and streaming graph modeling.

Dataset: <dataset_name>
Description:
<dataset_description>
The dataset is chronologically divided into 10 sequential graph tasks.

For the current task (task <task_index>), analyze the behavioral
    patterns of a target user from one of the following views:
- Transaction view: describes the target user's direct interactions with
     neighboring users.
- Time hyper-view: groups users with temporally close interactions to
    capture short-term patterns.
- Frequency hyper-view: groups users with similar interaction
    frequencies to capture long-term patterns.

Analyze the target user's behavioral patterns based on its attributes
    and the supplied neighbor context in the selected view. Focus on
    patterns that are informative and distinctive for the target user.

Use only the supplied context. Do not infer meanings for anonymized
    features, invent missing attributes, or predict the fraud label.

Return only one concise paragraph describing the target user's
    behavioral patterns. Do not include headings, bullet points, or
    additional explanations. The resulting analysis will be used to
    derive the node's semantic embedding for personalized prompt
    generation.

[Input]
Task index: <task_index>
```

Table 8: Mapping between MPPG prompt fields and model quantities.

Input fields	Corresponding quantities
`task_index`	Task index i (starting from 0)
`view_type`	View $v \in \{\mathcal{G}, \mathcal{HG}^T, \mathcal{HG}^F\}$
`target_node_id`	Target node u
`target_node_attributes`	Target node feature $\mathbf{x}_u$
`neighbor_context`	Direct neighbors and their features in $\mathcal{G}$, or incident hyperedges $\mathcal{H}^v(u)$ and the features of their member neighbors in $\mathcal{HG}^T$ and $\mathcal{HG}^F$

```
View: <view_type>
Target:
- Node ID: <target_node_id>
- Attribute: <target_node_attributes>
Neighbor context:
<neighbor_context>
```

We summarize the placeholders used in this template in Table 8. `neighbor_context` is instantiated according to the view type. For the transaction view, direct neighbors are serialized as:

```
- Neighbor #1:
  - Node ID: <neighbor_1_id>
  - Attribute: <neighbor_1_attributes>
- Neighbor #2:
  - Node ID: <neighbor_2_id>
  - Attribute: <neighbor_2_attributes>
...
```

For both the time and frequency hyper-views, neighbors are grouped by the incident hyperedges of the target node:

```
- Hyperedge #1:
  - Neighbor #1.1:
    - Node ID: <neighbor_1_1_id>
    - Attribute: <neighbor_1_1_attributes>
  - Neighbor #1.2:
    - Node ID: <neighbor_1_2_id>
    - Attribute: <neighbor_1_2_attributes>
- Hyperedge #2:
  - Neighbor #2.1:
    - Node ID: <neighbor_2_1_id>
    - Attribute: <neighbor_2_1_attributes>
...
```

Within each hyperedge, the target node itself is excluded from the neighbor list. If the target node has no direct neighbors or incident hyperedges in the selected view, `neighbor_context` is set to `None`.

N Extended Related Work

Fraud Detection on Graphs. Graph neural networks (GNNs) have become a dominant paradigm for graph-based fraud detection in financial transaction networks (Cheng et al., 2025; Ou et al., 2026b) and social networks (Qiao et al., 2025; Xu et al., 2024). Representative methods include PC-GNN (Liu et al., 2021). While early studies focus on static settings (Zou & Cheng, 2024; Pan et al., 2025), recent work has emphasized temporal dynamics in fraud detection (Ho et al., 2025; Tian et al., 2023). For example, DGNN-SR (Yuan et al., 2025) designs temporal encoding to capture

both relative and absolute temporal patterns in fraud behavior. RGTAN (Xiang et al., 2025) further incorporates temporal risk propagation to capture collusive fraudsters in dynamic transaction networks. And CamFD (Ou et al., 2026b) leverages contrastive learning to detect temporal inconsistencies in users' transaction histories. Despite these advances, most methods still rely on local neighborhood aggregation, which limits their ability to model evolving high-order collusive fraud patterns in streaming settings.

Streaming Fraud Detection and Graph Continual Learning. Real-world anti-fraud systems are inherently streaming, where transactions arrive continuously and fraud patterns drift over time (Motie & Raahemi, 2024; Khodabandehlou & Golpayegani, 2024). Existing graph continual learning methods mainly focus on mitigating catastrophic forgetting (Tian et al., 2024; Zhang et al., 2024b). Many studies leverage experience replay to preserve historical knowledge. ER-GNN (Zhou & Cao, 2021) introduces an early sampling-based replay framework for streaming graph learning by storing historical graph samples. SGNN-GR (Wang et al., 2022) generates random-walk sequences to synthesize historical graph samples, reducing the memory cost of storing real samples. DSLR (Choi et al., 2024) further improves rehearsal by jointly enhancing replay diversity and structure learning. RL-GNN (Song et al., 2025) uses rationale learning to preserve transferable task-relevant substructures and improve adaptation stability across tasks. In addition, some studies adopt parameter isolation to retain task-specific knowledge. PI-GNN (Zhang et al., 2023) reduces cross-task interference on dynamic graphs by isolating task-specific parameters, while MSCGL (Cai et al., 2022) combines multimodal continual graph learning with neural architecture search to adapt model structures. However, these methods mainly operate on low-order graph structures and provide limited theoretical analysis of embedding drift for high-order fraud under streaming dynamics.

High-order Graph Learning. High-order graph learning aims to model complex relations beyond pairwise links (Antelmi et al., 2023; Lee et al., 2025). Hypergraph neural networks (HGNNs) provide a effective solution by replacing pairwise edges with hyperedges that can connect multiple related nodes (Feng et al., 2019; Gao et al., 2022). Building on this paradigm, T-HyperGNN (Wang et al., 2024b) further introduces tensorized hypergraph representations to model high-order patterns. MHGNN (Ji et al., 2025) and MH-LGC (Ou et al., 2026a) improve high-order representation quality through multi-scale or multi-view hypergraph aggregation designs. TROPICAL (Haghighi et al., 2024) emphasizes robust high-order structure learning under complex graph conditions and improves downstream representation quality. Despite these advances, most existing high-order methods still depend on static or heuristic hypergraph construction and are mainly designed for non-streaming settings, which limits their ability to track rapidly evolving fraud patterns over time.

Large Language Models on Graphs. Large language models (LLMs) have shown strong performance in graph learning tasks (Chai et al., 2026; Yu et al., 2026). Existing methods that integrate graphs and LLMs can be grouped into three categories (Li et al., 2024b). ENGINE (Zhu et al., 2024) and GAugLLM (Fang et al., 2024) introduce LLMs as enhancers, using LLM-derived semantic priors to augment node and edge features, thereby producing richer inputs for downstream GNN predictors. GraphGPT (Tang et al., 2024) and GraphWiz (Chen et al., 2024) use LLMs as downstream predictors, where the LLM serves as the primary decision module for graph tasks based on graph-derived context and task prompts. Also, TEA-GLM (Wang et al., 2024a) and GraphTranslator (Zhang et al., 2024a) integrate LLMs as alignment components, aligning LLM-derived semantic embeddings and GNN-derived graph embeddings to bridge modality gaps and improve cross-modal consistency. However, most existing studies are designed for static settings and do not explicitly address continual adaptation under streaming concept drift. In contrast, we use LLMs to provide auxiliary signals that assist effective hypergraph construction and task adaptation in streaming fraud detection.