HERDGNN: Hybrid Error-Guided Regularization with Deviation for Highly Imbalanced Classification in Dynamic Graphs

Vahid Shahrivari Joghan
Utrecht University
Utrecht, The Netherlands
v.shahrivarigjoghan@uu.nl

Ramon Rico
Utrecht University
Utrecht, The Netherlands
r.ricocuevas@uu.nl

Ioana Hulpus
Utrecht University
Utrecht, The Netherlands
i.r.karnstedt-hulpus@uu.nl

Yannis Velegrakis
Utrecht University and University of Trento
Utrecht, The Netherlands
i.velegrakis@uu.nl

Abstract

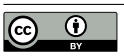
Risk monitoring tasks like financial fraud detection, money laundering detection, and cybersecurity often involve graph-native datasets that are characterized by extreme class imbalance and evolving structure, as well as deployment constraints that require incremental model updates under bounded temporal memory. Despite recent efforts in graph representation learning, the state-of-the-art Dynamic Graph Neural Networks (DGNNs) have several limitations: They are either designed for class-balanced settings and (some) offer a bounded temporal memory footprint, or are designed for class-imbalanced settings but do not offer a bounded temporal memory footprint. To bridge this gap, we propose HERDGNN: the first DGNN tailored for class-imbalanced settings while keeping a bounded temporal memory footprint. To address class imbalance, HERDGNN integrates deviation-aware regularization pushing abnormal entities away from the distribution of normal ones. To maintain bounded temporal memory, HERDGNN captures temporal information through a compact hierarchical memory whose size is fixed and independent of graph size, enabling live-update training without storing node-level temporal states. We evaluate the effectiveness of HERDGNN on 7 large-scale real and synthetic financial networks, under a live-update protocol, against 3 GNN baselines and 5 state-of-the-art DGNNs. The results show that HERDGNN achieves competitive or superior performance compared to the competing methods, and overall strikes the best balance between classification performance and memory consumption.

CCS Concepts

• **Computing methodologies** → **Supervised learning by classification**; **Neural networks**; • **Mathematics of computing** → *Graph algorithms.*

Keywords

Dynamic Graphs, Dynamic Graph Neural Networks, Highly Imbalanced Classification on Graphs, Graph Anomaly Detection



CIKM '26, Rome, Italy

ACM ISBN 979-8-4007-2539-5/2026/11
https://doi.org/10.1145/3799682.3840793

ACM Reference Format:
Vahid Shahrivari Joghan, Ramon Rico, Ioana Hulpus, and Yannis Velegrakis. 2026. HERDGNN: Hybrid Error-Guided Regularization with Deviation for Highly Imbalanced Classification in Dynamic Graphs. In *Proceedings of the 35th ACM International Conference on Information and Knowledge Management (CIKM '26), November 07–11, 2026, Rome, Italy*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3799682.3840793

1 Introduction

Graph learning finds applications in many risk monitoring scenarios, ranging from financial fraud and money laundering detection, to cybersecurity, or online platform security [31, 35, 52, 57, 58]. Nodes represent entities such as accounts, users, or devices, while edges model interactions between those entities, such as, transactions, communications, or events. In these scenarios, the ability to identify nodes or edges that correspond to malicious entities or interactions is of paramount importance, yet significantly challenging, for a number of reasons. The first is the extreme class imbalance of the data [18, 55]. Due to the cost and latency of verification processes, only a very small fraction of nodes or edges are typically labeled as positive, whereas the large unlabeled pool is often used as negative supervision in practice despite possible label contamination. The second reason is the dynamic nature of the data [28]. Financial transaction data evolves over time as new nodes and edges appear and interaction patterns shift. The third is that many applications require live-update learning, where models must be updated incrementally as new data-points arrive without repeatedly revisiting historical data [61]. Furthermore, deployment often imposes strict compute and memory budgets, making node/edge-level temporal representation histories difficult to maintain at scale [63].

Dynamic Graph Neural Networks (DGNNs) provide a natural framework for representation learning on evolving graphs. However, as summarized in Table 1, existing approaches model temporal information through mechanisms that do not simultaneously satisfy the requirements of bounded temporal memory, live-update training, and class-imbalance-aware classification. EvolveGCN [45] and ROLAND [61] capture temporal evolution by recurrently updating GNN parameters or node-level representations. However, their temporal mechanisms introduce scalability limitations: EvolveGCN-H and ROLAND maintain node-level temporal states whose memory cost grows with the number of nodes, while EvolveGCN-O

Table 1: High-level comparison of the state-of-the-art DGNNs and HERDGNN.

Model	Temporal modeling	Avoids storing node states over time?	Scalable w.r.t. $\lvert\mathcal{V}\rvert$ and/or $\lvert\mathcal{E}\rvert$?	Live-update?	Anomaly-aware?
WinGNN [63]	Meta-learning over sliding windows	✓	✓	✗	✗
EvolveGCN-O [45]	Evolving GCN weights over time	✓	✗	✗	✗
EvolveGCN-H [45]	Evolving node embedding over time	✗	✓	✗	✗
ROLAND [61]	Recurrent updates of node embeddings	✗	✗	✓	✗
HawkGNN [47]	Snapshot fusion + Hawkes time decay	✓	✗	✗	✗
ABNet [27]	Temporal feature-difference learning + meta-learning	✗	✗	✗	✓
HERDGNN (Ours)	Snapshot GNN + Hierarchical memory	✓	✓	✓	✓

evolves model parameters and is not designed for class-imbalance-aware anomaly detection. SFDyG/HawkGNN [47] avoids persistent node-level histories by fusing snapshots within a temporal window and applying Hawkes-process-inspired time decay, but its temporal computation still depends on the windowed graph structure. WinGNN [63] similarly does not maintain explicit node embedding histories, but it captures temporal dependencies through randomized sliding-window meta-learning and multi-step gradient updates. Finally, ABNet [27] explicitly targets anomaly detection under sample imbalance, but its temporal feature processing, anomaly augmentation, and meta-learning components do not yield a bounded temporal memory footprint. Therefore, existing DGNNs do not jointly provide bounded temporal memory, live-update learning, and class-imbalance-aware classification.

To address these limitations, the first DGNN that is tailored for class-imbalanced classification on discrete-time dynamic graphs that combines live-update learning with graph-size-independent persistent temporal memory. Our model, named HERDGNN (Hybrid Error-Guided Regularization with Deviation GNN) handles class imbalance by exploiting a deviation-inspired regularizer for the cross entropy loss that draws inspiration from the *deviation networks* [44]. Specifically, in addition to cross entropy supervision, HERDGNN maintains online estimates of the distribution of negative instances and encourages positive instances to deviate toward the tail of said distribution.

To maintain a bounded temporal memory overhead, HERDGNN captures temporal information through a compact hierarchical memory whose size is fixed and independent of the graph size, enabling live-update training without storing node-level temporal states. HERDGNN is trained in a streaming fashion. At a time step t, HERDGNN processes the t-th graph snapshot using a static GNN and models temporal evolution through a fixed-size layer-wise memory that modulates the GNN parameters across snapshots. It does not require storage of node-level temporal histories nor revisiting past snapshots, yielding a temporal overhead that is independent of the dynamic graph size.

To evaluate the effectiveness of HERDGNN, we compare it to three static GNN baselines and five state-of-the-art DGNN families, comprising eleven system variants in total, on seven large-scale real-world and synthetic graph benchmarks. The results show that HERDGNN strikes the best balance between classification performance and memory consumption. Overall, the contributions of this work are as follows:

- We propose HERDGNN, a DGNN for class-imbalanced classification on discrete-time dynamic graphs, with a fixed-size layer-wise temporal memory that generates GNN parameters across snapshots, enabling live-update learning with graph-size-independent persistent memory.
- We integrate deviation-aware regularization to address extreme class imbalance without storing historical risk scores.
- We provide an extensive evaluation showing that HERDGNN achieves a favorable trade-off between classification performance and memory consumption.

In what follows, after positioning our approach with respect to the related work (Section 2), we present our solution (Section 3 and 4), and its performance evaluation (Section 5).

2 Related Work

Anomaly detection in graphs is a well-established research area motivated by high-impact applications in cybersecurity, financial fraud monitoring, and social platforms [17, 39]. In *dynamic* graphs, some works target the streaming setting, prioritizing throughput and bounded memory via randomized and sketch-based techniques [1, 4, 5, 12, 20, 21, 30, 40, 48, 60]. Ranshous et al. [48] introduced one of the earliest sketch-driven approaches for edge streams, using Count-Min Sketch [15] to approximate structural statistics with bounded error and enable memory-efficient scoring at high throughput. Building on this paradigm, MIDAS [4] detects abrupt temporal bursts in node/edge activity by providing theoretical control over false-positive rates, while AnoEdge [5] generalizes sketching to higher-order patterns by representing dense interaction structures as submatrices. SedanSpot scores incoming edges using randomized random-walk views [20]. SpotLight employs sketching with random projections and streaming histograms to track the emergence and disappearance of dense substructures [21]. StreamSpot detects anomalous subgraphs via shingling and locality sensitive hashing [29, 40]. AnomRank ranks anomalies based on temporal changes in personalized PageRank centrality [43, 60]. Despite their efficiency and favorable theoretical properties, these methods largely rely on hand-crafted statistics and provide limited capacity for representation learning.

In addition to sketch-based streaming methods, many recent approaches rely on GNNs as learned encoders for graph anomaly detection, since neighborhood aggregation can capture higher-order structural and attribute patterns beyond hand-crafted statistics. Typical backbones include GIN, GAT, and GraphSAGE [24, 54, 59],

which can be used with snapshot-level encoders or as the structural backbone inside temporal GNN frameworks.

For discrete-time dynamic graphs, a common formulation represents the evolution as a sequence of snapshots. Discrete-time based GNNs such as EvolveGCN [45] and ROLAND [61] model temporal evolution by updating GNN parameters or node representations across time. However, many such designs maintain node-level temporal states or cached embeddings, which can induce substantial memory growth and limit scalability in large graphs. HawkGNN (SFDyG) [47] mitigates explicit history storage by fusing multiple snapshots within a sliding window and applying time-decayed message passing motivated by the Hawkes point process [26]. Nevertheless, the fused representation and associated computation can still scale with graph size and window edge size. WinGNN [63] proposes an encoder-free alternative based on meta-learning over randomized sliding windows inspired by MAML [23], where gradients from consecutive snapshots are aggregated to enable temporal adaptation.

Several discrete-time based GNNs are tailored specifically to unsupervised anomaly detection in evolving graphs. AddGraph [62] combines GNN embeddings with recurrent updates [13] and attention [53] to capture spatiotemporal patterns, but requires retaining historical embeddings across time steps. StrGNN [10] improves edge-level detection through informative subgraph extraction and spatiotemporal prediction. Unsupervised approaches such as AddAG-AE [41] and FuSAGNet [25] use reconstruction and forecasting objectives to detect anomalies as deviations from expected evolution, while transformer-based designs such as TADDY [37] model long-range dependencies with higher computational cost. More recently, ABNet [27] addresses anomalous node detection in dynamic graphs under severe sample imbalance. It introduces a temporal feature-difference extractor to reduce representation bias toward normal nodes, an anomaly augmentation module based on discrete wavelet transforms and autoencoder perturbations to enrich rare anomalous samples in evolving graph structures. Although ABNet is explicitly anomaly-aware, its reliance on multi-snapshot feature processing, anomaly augmentation, and meta-learning introduces extra runtime memory and computation, which may limit its applicability to large-scale dynamic graph settings.

Continuous-time based GNNs operate on fine-grained, timestamped interaction streams, learning temporal embeddings by sampling historical neighborhood events and aggregating them with time-aware encoders or event-driven message passing [14, 16, 49]. Recent works such as SAD [51] and GADY [38] adopt this paradigm but differ in learning strategy. SAD leverages limited supervision via a temporal encoder with a time-aware memory bank and pseudo-label contrastive learning via deviation loss [44], whereas GADY is fully unsupervised and augments continuous-time message passing with adversarially generated negatives. A key limitation is scalability since the set of relevant historical interactions can grow rapidly, causing temporal neighborhood explosion and substantially increasing both memory footprint and computation.

Existing Dynamic GNNs [45, 49, 61, 62] vary in the handling of temporal dependencies, memory, and supervision, which either incur graph-size-dependent memory costs or rely on fully supervised settings that are impractical for large, evolving graphs. We address these limitations by combining snapshot-based learning with a bounded-parameter temporal mechanism and a classification objective augmented with anomaly-detection–inspired regularization, enabling scalable live-update learning under extreme class imbalance.

3 Preliminaries and Problem Statement

We consider both node-level and edge-level classification tasks on a discrete-time dynamic graph under extreme class imbalance and limited memory. Formally,

Definition 3.1. A *discrete-time dynamic graph* is a collection $\mathcal{G} = \{G_t\}_{t=1}^{T}$, where $G_t = (\mathcal{V}_t, \mathcal{E}_t)$ is a graph with node set $\mathcal{V}_t$ and edge set $\mathcal{E}_t \subseteq \mathcal{V}_t \times \mathcal{V}_t$. An edge $e_{i,j}^t = (v_i^t, v_j^t) \in \mathcal{E}_t$ indicates an interaction between nodes $v_i \in \mathcal{V}_t$ and $v_j \in \mathcal{V}_t$ during time window $t \in \{1, \ldots, T\}$. Nodes and edges may have attributes. Fixed a time window t, the node and edge attribute matrices are denoted as $\mathbf{X}_{\mathcal{V}_t}$ and $\mathbf{X}_{\mathcal{E}_t}$, respectively.

Problem 1 (Node/Edge-level classification on a discrete–time dynamic graph under extreme class imbalance and limited memory). *Let $\mathcal{G} = \{G_t\}_{t=1}^{T}$ be a discrete-time dynamic graph and $\mathbf{Y} = \{\mathbf{Y}_t\}_{t=1}^{T}$ be a collection of vectors where $\mathbf{Y}_t \in \{0,1\}^{|\mathcal{V}_t|}$ (resp., $\mathbf{Y}_t \in \{0,1\}^{|\mathcal{E}_t|}$) contains the labels of the nodes (resp., edges) of graph G_t. Each component y_v^t (resp., y_e^t) of vector $\mathbf{Y}_t$ indicates whether a node v (resp., edge e) is malicious at time t, denoted $y_v^t = 1$, (resp., $y_e^t = 1$) or if it has no verified positive label $y_v^t = 0$ (resp., $y_e^t = 0$). We operate under the extreme class imbalance assumption. That is, the number of malicious entities is significantly lower than the number of normal entities. Let M be the number of available memory slots and d_m their memory size. The goal is to learn a scoring function $s_\theta(\cdot)$, while restricting the temporal memory overhead carried across snapshots to a budget of $M \cdot d_m$ during the learning process, such that given an unlabeled node v^{T+1} (resp., edge e^{T+1}) that belongs to the unseen graph G_{T+1}, we can predict its label as $\hat{y}(v^{T+1}) = \mathbb{I}\{s_\theta(v^{T+1}) \geq \tau\}$, (resp., $\hat{y}(e^{T+1}) = \mathbb{I}\{s_\theta(e^{T+1}) \geq \tau\}$). Function $\mathbb{I}(\cdot)$ denotes the indicator function and $\tau \in \mathbb{R}_+$ is a threshold.*

Problem 1 is also referred to as node/edge-level *anomaly detection* on a discrete-time dynamic graph under limited memory. Under this naming convention, the anomalies refer to the malicious entities. In what follows, we will use these names interchangeably.

4 A DGNN for class-imbalanced classification on discrete-time dynamic graphs

We describe the HERDGNN architecture, its training and inference modes, and, finally, we provide a comparative memory complexity analysis against the state-of-the-art DGNNs.

4.1 The HERDGNN Architecture

4.1.1 The Snapshot-based GNN Backbone. Let $\mathcal{G} = \{G_t\}_{t=1}^{T}$ be a discrete-time dynamic graph. For each snapshot G_t, HERDGNN exploits an L-layer message-passing GNN to compute node embeddings. Formally,

$$\mathbf{H}_t^{(l)} = \left[h_{v_1}^{(l)}, \ldots, h_{v_{|\mathcal{V}_t|}}^{(l)} \right], \quad l \in \{0, \ldots, L\}, \tag{1}$$

denotes the matrix of node embeddings at layer l, where $\mathbf{H}_t^{(0)} = \mathbf{X}_{\mathcal{V}_t}$. The final node embedding matrix corresponds to $\mathbf{H}_t := \mathbf{H}_t^{(L)}$. The

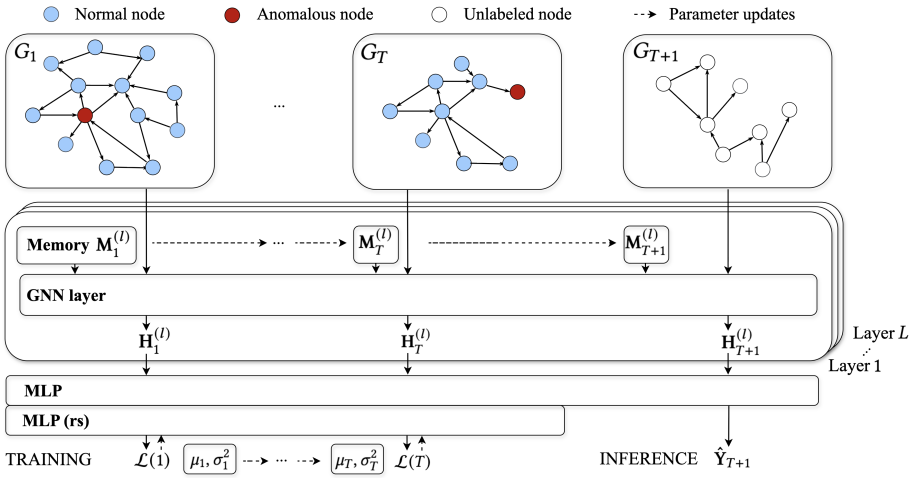


Figure 1: The HERDGNN architecture depicting both the training and the inference phases

hidden representation of node $v \in \mathcal{V}_t$ at layer l, denoted $h_v^{(l)}$, is given by

$$h_v^{(l)} = \sigma\Big(\mathbf{W}_t^{(l)} \cdot \mathrm{AGG}^{(l)}\big(\{h_u^{(l-1)} \mid u \in \mathcal{N}(v) \cup \{v\}\}\big)\Big), \tag{2}$$

where $\mathrm{AGG}^{(l)}(\cdot)$ is a permutation-invariant aggregator, $\mathcal{N}(v)$ denotes the set of neighbors of v in G_t, $\sigma(\cdot)$ is a non-linear activation function, and $\mathbf{W}_t^{(l)}$ are the l-th layer GNN weights used at time t.

4.1.2 The Hierarchical Temporal Memory. To avoid graph-size-dependent temporal storage, HERDGNN models temporal evolution through a fixed-size, layer-wise memory that controls how each GNN layer evolves across snapshots.

Specifically, for each GNN layer $l \in \{1, \ldots, L\}$, we maintain a persistent layer-level memory vector

$$\mathbf{m}_{t-1}^{(l)} \in \mathbb{R}^{d_m}, \quad d_m \in \mathbb{N}. \tag{3}$$

The size of the memory vectors of all layers is the same and is controlled only by a hyperparameter d_m and hence it does not grow with $|\mathcal{V}_t|$ or $|\mathcal{E}_t|$.

For each layer l, the memory is initialized as

$$\mathbf{m}_0^{(l)} = \mathbf{0} \in \mathbb{R}^{d_m}. \tag{4}$$

At snapshot t, the memory is updated through a lightweight recurrent transition:

$$\mathbf{m}_t^{(l)} = \phi_{\mathrm{mem}}^{(l)}\left(\mathbf{m}_{t-1}^{(l)}\right), \tag{5}$$

where $\phi_{\mathrm{mem}}^{(l)}$ denotes the complete recurrent transition applied to the previous memory state and is implemented using a lightweight GRU-based update for layer l.

To generate the time-dependent layer parameters, HERDGNN uses a layer-specific linear weight generator:

$$\boldsymbol{\theta}_t^{(l)} = g_\psi^{(l)}\left(\mathbf{m}_t^{(l)}\right) = \mathbf{A}^{(l)}\mathbf{m}_t^{(l)} + \mathbf{b}^{(l)}, \tag{6}$$

$$\mathbf{W}_t^{(l)} = \mathrm{Reshape}\left(\boldsymbol{\theta}_t^{(l)}\right). \tag{7}$$

Here, $g_\psi^{(l)}$ denotes the layer-specific parameter generator, while $\mathbf{A}^{(l)}$ and $\mathbf{b}^{(l)}$ are its learnable parameters. The operator $\mathrm{Reshape}(\cdot)$ maps the generated vector $\boldsymbol{\theta}_t^{(l)}$ to the weight-tensor shape required by the selected GNN operator. The resulting tensor $\mathbf{W}_t^{(l)}$ is then used in Equation (2) to perform message passing at snapshot t.

4.2 Training and Inference in HERDGNN

HERDGNN has two prediction heads, a classification and a deviation head. The classification head is used both during training and inference, while the deviation head is exclusively used during training. The sole purpose of the deviation-head is shaping the risk-score distribution to better cope with extreme class imbalance. HERDGNN is trained and tested following a *live-update protocol* [61]. In the following, we detail the training and inference phases of HERDGNN.

4.2.1 HERDGNN Training. HERDGNN is trained with two optimization objectives: A weighted binary cross entropy loss and a deviation-based regularizer that is applied to an auxiliary risk score.

For node-level prediction, given the final node embedding of a labeled node u at time t, denoted $\mathbf{h}_{t,u}$, we compute a classification logit denoted by $o_{t,u} \in \mathbb{R}$ and an auxiliary risk logit denoted by $r_{t,u} \in \mathbb{R}$. Formally,

$$o_{t,u} = f_{\mathrm{cls}}^V(\mathbf{h}_{t,u}), \tag{8}$$

$$r_{t,u} = f_{\mathrm{dev}}^V(\mathbf{h}_{t,u}), \tag{9}$$

Similarly, for edge-level prediction, given a labeled edge $e = (u, v)$, we form the edge representation

$$\mathbf{z}_{t,e} = [\mathbf{h}_{t,u} \| \mathbf{h}_{t,v} \| \mathbf{x}_{t,e}], \tag{10}$$

where $\mathbf{x}_{t,e}$ is the edge feature vector and $\|$ denotes vector concatenation. The edge classification logit and risk logit are computed analogously to that of the node-level setting. That is,

$$o_{t,e} = f_{\mathrm{cls}}^E(\mathbf{z}_{t,e}), \tag{11}$$

$$r_{t,e} = f_{\mathrm{dev}}^E(\mathbf{z}_{t,e}). \tag{12}$$

To address class imbalance, we use a positive-class weighted binary cross entropy loss. Formally,

$$\mathcal{L}_{\text{cls}}(t) = -\frac{1}{|\Omega_t|} \sum_{i \in \Omega_t} \left[\beta_t y_{t,i} \log \sigma(o_{t,i}) + (1 - y_{t,i}) \log(1 - \sigma(o_{t,i})) \right], \tag{13}$$

where Ω_t denotes the supervised entities in snapshot G_t. For the node-level task, Ω_t is the set of labeled nodes in $\mathcal{V}_t$, and for edge-level task, Ω_t is the set of labeled edges in $\mathcal{E}_t$. Further, β_t denotes the positive-class weight defined as

$$\beta_t = \frac{N_t^-}{N_t^+ + \epsilon}, \tag{14}$$

where $\epsilon > 0$ is a small constant that is introduced to ease numerical stability and

$$N_t^+ = \sum_{i \in \Omega_t} y_{t,i}, \qquad N_t^- = |\Omega_t| - N_t^+. \tag{15}$$

Even though the weighted cross entropy as defined in Equation (13) addresses class imbalance locally through class weights, it does not explicitly shape the distribution of the anomaly scores. To further separate anomalous entities from the majority normal class, we introduce a deviation-based regularizer that is inspired by the *deviation networks* [44]. The regularizer assumes that the auxiliary risk logits of normal entities follow a (μ_t, σ_t)-Gaussian distribution and encourages anomalous entities to lie in the tails of this distribution[1].

To avoid storing historical risk scores, HERDGNN maintains running estimates of (μ_t, σ_t). These estimates are updated recursively by merging the estimates at time t with those at $t-1$. Formally,

$$\mu_t = (1 - \omega)\mu_{t-1} + \omega \hat{\mu}_t, \tag{16}$$

$$\sigma_t^2 = (1 - \omega) \left[\sigma_{t-1}^2 + (\mu_{t-1} - \mu_t)^2 \right] + \omega \left[\hat{\sigma}_t^2 + (\hat{\mu}_t - \mu_t)^2 \right], \tag{17}$$

where $\omega \in (0, 1)$ is a blending factor controlling the contribution of the current snapshot and

$$\hat{\mu}_t = \frac{1}{|\Omega_t^0|} \sum_{i \in \Omega_t^0} r_{t,i}, \tag{18}$$

$$\hat{\sigma}_t^2 = \frac{1}{|\Omega_t^0|} \sum_{i \in \Omega_t^0} (r_{t,i} - \hat{\mu}_t)^2. \tag{19}$$

Here,

$$\Omega_t^0 = \{i \in \Omega_t : y_{t,i} = 0\}, \quad \text{and} \quad \Omega_t^1 = \{i \in \Omega_t : y_{t,i} = 1\} \tag{20}$$

denote the set of normal and respectively anomalous supervised entities in snapshot G_t, respectively. In our setting, entities without a verified positive label are assigned to Ω_t^0. This set may therefore contain undetected anomalies, but under extreme class imbalance it is assumed to be dominated by normal entities. Further, $r_{t,i}$ denote the risk logit described in Equation (9) for node-level tasks and in Equation (12) for edge-level tasks. The temporal update mechanism described in Equations (16) and (17) is highly efficient as it provides a compact summary of the normal risk-score distribution without the need of retaining past scores.

[1]This approximation is not intended as a strict distributional assumption, rather, it provides a memory-efficient way to standardize risk scores and define an adaptive deviation margin under streaming updates.

Based on the aforementioned estimates, the deviation regularizer penalizes normal entities that deviate far from the normal-score mean and penalizes anomalous entities whose deviation is smaller than a pre-defined margin γ_t. Formally,

$$\mathcal{L}_{\text{dev}}(t) = \frac{1}{|\Omega_t^0|} \sum_{i \in \Omega_t^0} d_{t,i} + \frac{1}{|\Omega_t^1|} \sum_{i \in \Omega_t^1} \max(0, \gamma_t - d_{t,i}), \tag{21}$$

where

$$d_{t,i} = \frac{|r_{t,i} - \mu_t|}{\sigma_t + \epsilon} \tag{22}$$

is the absolute standardized deviation of entity i. The margin γ_t is selected according to the anomaly ratio in the current supervised snapshot. We set

$$\gamma_t = \Phi^{-1} \left(1 - \frac{\pi_t}{2} \right), \tag{23}$$

where $\pi_t = \frac{N_t^+}{|\Omega_t|}$, and Φ^{-1} is the inverse cumulative distribution function of the standard normal distribution. Thus, rarer anomalies require the regularizer to enforce a larger standardized deviation from the normal-score distribution.

The final optimization objective at snapshot t combines the weighted binary cross entropy loss and the deviation-based regularizer via a hyperparameter $\lambda \geq 0$ that controls the strength of the regularization. That is,

$$\mathcal{L}(t) = \mathcal{L}_{\text{cls}}(t) + \lambda \mathcal{L}_{\text{dev}}(t). \tag{24}$$

Algorithm 1 HERDGNN Training and Inference

Require: A discrete-time dynamic graph $\mathcal{G} = \{G_1, \ldots, G_T\}$ with corresponding label vectors $\{\mathbf{Y}_1, \ldots, \mathbf{Y}_T\}$ (at the node or edge level), a GNN f_θ with hierarchical memory $M = \{M^{(1)}, \ldots, M^{(L)}\}$, deviation loss weight λ, parameter ω.
Ensure: The updated GNN f_θ and classification predictions $\hat{\mathbf{Y}}_t$

Initialize deviation statistics (μ, σ) using G_1
for $t \leftarrow 2$ **to** T **do**
  Split the labels of G_{t-1} into $\mathbf{Y}_{t-1}^{\text{train}}$ and $\mathbf{Y}_{t-1}^{\text{val}} \subseteq \mathbf{Y}_{t-1}$
  $best\_score \leftarrow 0$
  repeat ▷ Training phase: Train the HERDGNN model on snapshot G_{t-1}
    $\hat{Y}_{t-1}, r_{t-1}^{\text{train}} \leftarrow f_\theta(G_{t-1}; M)$
    $L_{\text{cls}} \leftarrow \textsc{ClassificationLoss}\big(\hat{Y}_{t-1}^{\text{train}}, Y_{t-1}^{\text{train}}\big)$
    $L_{\text{dev}} \leftarrow \textsc{DeviationLoss}\big(r_{t-1}^{\text{train}}, Y_{t-1}^{\text{train}}; \mu, \sigma\big)$
    $L \leftarrow L_{\text{cls}} + \lambda \cdot L_{\text{dev}}$
    $\theta \leftarrow \textsc{OptimizerStep}(\theta, \nabla_\theta L)$
    $val\_score \leftarrow \textsc{Evaluate}\big(\hat{\mathbf{Y}}_{t-1}^{\text{val}}, \mathbf{Y}_{t-1}^{\text{val}}\big)$ ▷ e.g., AUNPRC
    if $val\_score \leq best\_score$ **then**
      break ▷ early stopping
    end if
    $best\_score \leftarrow val\_score$
  until maximum number of epochs reached
  $\hat{\mathbf{Y}}_t \leftarrow f_\theta(G_t; M)$ ▷ Inference phase: Run HERDGNN on next snapshot G_t
  $(\mu, \sigma) \leftarrow \textsc{UpdateDeviationStats}\big(\mu, \sigma, \omega\big)$
end for

4.2.2 HERDGNN Inference. At inference time, HERDGNN only relies on the classification head to make predictions. Given a target entity i (node or edge depending on the task), the predicted anomaly probability is

$$\hat{p}_{t,i} = \sigma(o_{t,i}). \tag{25}$$

Here, $\sigma(\cdot)$ denotes the sigmoid function and $o_{t,i}$ is the classification logit described in Equation (8) for node-level tasks, and Equation (11) for edge-level tasks. Last, a binary prediction for entity i is obtained as

$$\hat{y}_{t,i} = \mathbb{I}\{\hat{p}_{t,i} \geq \tau\}, \tag{26}$$

where τ is the decision threshold.

The pseudocode of the training and inference routines is provided in Algorithm 1.

4.3 Space Complexity Analysis

HERDGNN maintains a persistent temporal state that is independent of the size of the input graph G. To support long, high-throughput graph streams under tight memory budgets, HERDGNN adopts an incremental (live-update) training protocol inspired by *truncated backpropagation-through-time* [61]. At time t, the model is updated using only the information available at the current snapshot G_t, without storing previous snapshots. The only state that is carried across time consists of the model parameters, the layer-wise memory $\{\mathbf{M}_t^{(l)}\}_{l=1}^{L}$, and the deviation-statistics scalars (μ_t, σ_t) used by the deviation-aware objective.

As mentioned, HERDGNN does not cache historical node embeddings $\{\mathbf{H}_\tau^{(l)}\}_{\tau<t}$. Rather, each layer maintains a fixed-size memory vector $\mathbf{m}_t^{(l)} \in \mathbb{R}^{d_m}$ that is independent of $|\mathcal{V}_t|$ and $|\mathcal{E}_t|$. Therefore, the temporal memory overhead is $O(L \cdot d_m)$ with $d_m \ll n$. Besides, HERDGNN uses $O(Ld_m^2)$ memory for the GRU parameters used to update the hierarchical memory, and $O(Ld_m f^2)$ for the trainable full linear generator in Eq. (7), which maps each memory vector to the GNN layer parameters.

Table 2 presents a comparison between HERDGNN and DGNNs like DySAT [50], EvolveGCN [45], ROLAND [61], SFDyG [47] and ABNet [27]. It summarizes the per-snapshot working memory required for training/update and the size of the persistent temporal state maintained across time. As shown in the last column of Table 2, most DGNN baselines maintain persistent temporal states that scale with the number of nodes n, the number of edges e, or the temporal window length S. In contrast, HERDGNN carries only $O(Ld_m)$ fixed-size layer-wise memory across snapshots. Thus, its persistent temporal state is independent of graph size and window length, making it more suitable for large-scale live-update learning.

5 Experimental Evaluation

We evaluate HERDGNN on 7 dynamic graph benchmarks covering both node-level and edge-level anomaly detection tasks. We compare against 3 static GNN baselines, namely GCN [33], GIN [59], and GAT [54], and several dynamic graph learning baselines, including EvolveGCN [45], ROLAND [61], WinGNN [63], HawkGNN [47], and ABNet [27]. Considering different variants of these methods, our evaluation contains a total of 11 competing baselines.

To reflect a realistic streaming deployment scenario, we adopt a live-update evaluation protocol [61]. At each time step t, each model is updated using only the currently observed snapshot G_t. After this update, the model is evaluated on the next snapshot G_{t+1}. This protocol avoids training on future snapshots and does not require repeatedly revisiting the full historical graph sequence.

All experiments were conducted on a server equipped with Intel(R) Xeon(R) Gold 6238R CPUs @ 2.20GHz and NVIDIA A100-PCIE-40GB GPUs. We implemented all models using PyTorch [46] and PyTorch Geometric [22]. Our code is publicly available at https://github.com/fafal-abnir/HerdGNN

5.1 Experimental Setup

5.1.1 Compared Methods. We compare HERDGNN against both static and dynamic GNN baselines. The static GNN baselines are GCN [33], GIN [59], and GAT [54]. The DGNN baselines include EvolveGCN [45], ROLAND [61], WinGNN [63], HawkGNN [47], and ABNet [27]. In the case of ROLAND, we evaluate its moving-average, GRU, and MLP update variants. For HawkGNN, we evaluate the GCN and GAT backbone variants.

HERDGNN uses a GIN backbone. Further, we set the deviation regularization coefficient to $\lambda = 0.05$ (fixed across all datasets for the main results in Table 4), the number of training epochs per snapshot to 200, dropout to 0.1, the Gaussian blending factor to $\omega = 0.9$, the hidden dimension to 128, and the memory dimension to 256.

For a fair comparison, we use the same hidden dimension across all methods and set the backbone GNN to GIN (HERDGNN's backbone) for all systems except for HawkGNN whose architecture is not straightforwardly transferrable to other encoders. Further, all systems are evaluated under the same live-update protocol, hardware environment, hidden dimension, learning rate, dropout, and other shared training hyperparameters. For method-specific hyperparameters, such as temporal window size or update parameters, we use the recommended settings from the original papers. All models are optimized using Adam [32].

5.1.2 Datasets. We evaluate the effectiveness of HERDGNN on a diverse range of real-world and synthetic dynamic graph datasets spanning social and financial crime domains. The statistics of the considered datasets are summarized in Table 3. Specifically, we conduct our experimental evaluation on the following datasets:

Reddit: We use the Subreddit Hyperlink Network [34], where nodes are subreddits and directed edges are hyperlinks between posts, annotated with timestamps, sentiment labels, and text features. We consider both title and body based networks as edge-level anomaly detection tasks.

DGraphFin: DGraphFin [28] is a large-scale dynamic financial graph from Finvolution, with users as nodes and emergency-contact relations as directed edges. Labels define a highly imbalanced node-level anomaly detection setting.

Elliptic++: Elliptic++ [18] extends Elliptic [56] with improved labels and temporal granularity for crypto transactions. Only a subset of nodes are labeled, reflecting practical AML conditions.

Ethereum Phishing: In the Ethereum Phishing dataset [11] verified phishing addresses are expanded via a second-order BFS and temporally filtered, yielding a node-level anomaly detection task.

SAML: SAML [42] is a large-scale synthetic AML dataset generated by an enhanced simulator modeling laundering typologies. It is used for edge-level anomaly detection under extreme class imbalance.

Table 2: Memory complexity and temporal memory overhead of DGNNs. Here, n is the number of distinct nodes, e is the maximum number of edges per snapshot, f is the hidden dimension, L is the number of GNN layers, and S is the temporal window length.

Model	Memory Complexity	Temporal Memory Overhead
WinGNN	$O(L(nf+ef)+Lf^2+B^*)$	Additional B^* buffers for loss/gradient accumulation/multi-step meta-updates
DySAT	$O(SL(nf+ef)+Lf^2+nSf+nS^2)$	Window buffer of nS node embeddings and nS^2 temporal-attention maps
EvolveGCN-O	$O(L(nf+ef)+Lf^2)$	$O(Lf^2)$ previous GCN weights + $O(Lf^2)$ GRU/LSTM parameters
EvolveGCN-H	$O(L(nf+ef)+Lf^2)$	$O(Lnf)$ hidden node embeddings + $O(Lf^2)$ GRU/LSTM parameters
ROLAND-MLP	$O(L(nf+ef)+Lf^2)$	$O(Lnf)$ hierarchical node embeddings + $O(Lkf^2)$ where k is the MLP layer depth
ROLAND-GRU	$O(L(nf+ef)+Lf^2)$	$O(Lnf)$ hierarchical node embeddings + $O(Lf^2)$ GRU parameters
ROLAND-Moving	$O(L(nf+ef)+Lf^2)$	$O(Lnf)$ hierarchical node embeddings
SFDyG-F	$O(L(nf+ef)+Lf^2+\lvert E^*\rvert)$	$\lvert E^*\rvert$ Hawkes excitation weights where $\lvert E^*\rvert \approx eS$
ABNet	$O(SL(nf+ef)+Snf+S\lvert\mathcal{A}\rvert f+B^*)$	$O(Snf)$ temporal node embeddings across snapshots + B^* for projection-based learning
HERDGNN (Ours)	$O(L(nf+ef)+Ld_m^2+Ld_mf^2)$	$O(Ld_m)$ fixed-size memory cells + $O(Ld_m^2)$ GRU parameters

Table 3: Dataset Characteristics

Dataset Name	# Nodes	# Edges	Snapshot Freq.	# Snapshots	Anomaly Ratio	Anomaly Task
Elliptic++	203,769	234,355	Biweekly	49	2%	Node-level
DGraphFin	3,700,550	4,300,999	Weekly	116	0.4%	Node-level
Ethereum Phishing	2,737,308	12,417,592	Monthly	19	0.043%	Node-level
Reddit-Body	35,776	286,561	Weekly	178	7%	Edge-level
Reddit-Title	54,075	571,927	Weekly	178	11%	Edge-level
AMLWorld	515,080	5,078,345	Daily	10	0.102%	Edge-level
SAML	855,460	9,504,852	Weekly	45	0.1038%	Edge-level

AMLWorld: AMLWorld [2] is a collection of synthetic AML transaction graphs released by IBM Research. We use the Small–HI variant as an edge-level benchmark with rare illicit transactions.

5.1.3 Evaluation Metrics. We measure the performance of the models on classification with AUPRC, the area under the precision recall curve [8] and AUROC, the area under the receiver operating characteristic curve [9]. Both curves allow the evaluation of binary classification performance over a range of thresholds. For a global score of the models' performance, we report mean AUROC and mean AUNPRC. AUNPRC, the Normalized AUPRC [7], accounts for class skew so it can be meaningfully averaged over multiple time steps [2] . We discard the first five windows as a warm-up period before reporting metrics.

5.2 Results

Table 4 reports mean AUROC and mean AUNPRC averaged across all snapshots for every dataset. We make two key observations with respect to HERDGNN. First, we note that HERDGNN is the only approach that remains competitive on all datasets without encountering out-of-memory (OOM) or excessive time (OOT) failures. In fact, HERDGNN strikes the best balance between classification performance and memory consumption.

[2]Some prior studies on money-laundering benchmarks such as AMLWorld and SAML like FraudGT [36] and GFP [6] reported high F1 scores even with standard GNN encoders (GAT, GIN, ...). A key factor is the evaluation setting: these works typically assume access to the entire transaction graph and construct train/validation/test splits by randomly/time partitioning edges(60/20/20). This setup allows message passing to leverage structural signals that would not be available in deployment scenarios where graphs evolve over time or new nodes/edges appear.

Second, HERDGNN achieves the best or near-best performance on most datasets, with particularly strong results on large-scale and highly imbalanced datasets. In terms of mean AUNPRC, HERDGNN achieves the best performance on Elliptic++, DGraphFin, Ethereum-Phishing, SAML, and AMLWorld, and ties the best result on Reddit-Body. In terms of mean AUROC, HERDGNN achieves the best performance on DGraphFin, Ethereum-Phishing, and SAML, ties the best result on Elliptic++, and remains competitive on Reddit-Body and AMLWorld. On Reddit-Title, HawkGCN obtains the strongest performance, while HERDGNN remains among the competitive methods.

Regarding the OOM/OOT issues encountered by competing methods, WinGNN performs multi-step meta-updates over a sliding window and accumulates gradients across snapshots. Hence, the increase in window length results in an increase in the number of inner-loop updates and, consequently, the overall training time. This leads to OOT issues. HawkGNN fuses multiple snapshots and assigns learnable Hawkes excitation weights to the edges within the fusion window. The number of such weights grows approximately linearly with the number of fused edges. Hence, the parameter and activation footprint can become large during training, causing OOM during backpropagation. Finally, all ROLAND's variants maintain a node-level temporal state influenced by the previous snapshot embeddings for all nodes at each GNN layer. This layer-wise storage increases with the number of nodes and the number of layers, triggering OOM on large graphs during the training phase. Additionally, as shown in Figure 2, updating these per-node hidden states at every snapshot introduces a substantial overhead which results in increasing per-epoch training runtime.

Table 4: Experimental results averaged across snapshots. Values are reported as mean ± standard deviation over snapshots. OOM denotes out-of-memory, and OOT indicates that training did not finish within 8 hours.

Model/Dataset	Reddit-Title	Reddit-Body	Elliptic++	DGraphFin	Ethereum-Phishing	SAML	AMLWorld
				Mean AUROC			
GCN	0.71 ± 0.05	0.63 ± 0.05	0.76 ± 0.15	0.75 ± 0.01	0.69 ± 0.13	0.73 ± 0.13	0.85 ±0.02
GIN	0.72 ± 0.06	0.62 ± 0.05	0.77 ± 0.12	0.77 ± 0.01	0.65 ± 0.11	0.72 ± 0.12	0.84 ±0.03
GAT	0.72 ± 0.03	0.64 ± 0.05	0.80 ± 0.16	0.75 ± 0.01	0.66 ± 0.12	0.74 ± 0.03	0.84 ±0.02
EvolveGCN-O	0.81 ± 0.04	0.64 ± 0.06	0.77 ± 0.13	0.78 ± 0.01	0.69 ± 0.10	0.79 ± 0.05	0.82 ± 0.03
ROLAND-mlp	0.82 ± 0.03	0.61 ± 0.06	0.76 ± 0.13	OOM	OOM	OOM	OOT
ROLAND-moving	0.86 ± 0.05	0.63 ± 0.05	0.56 ± 0.14	OOM	OOM	OOM	OOT
ROLAND-gru	0.86 ± 0.02	0.66 ± 0.07	**0.83 ± 0.14**	OOM	OOM	OOM	OOT
HawkGCN	**0.93 ± 0.01**	**0.69 ± 0.05**	0.68 ± 0.12	OOM	OOM	0.77 ± 0.03	**0.88 ± 0.01**
HawkGAT	0.81 ± 0.06	0.62 ± 0.04	0.75 ± 0.11	OOM	OOM	0.84 ± 0.02	0.85 ± 0.01
WinGNN	0.77 ± 0.11	0.57 ± 0.03	0.60 ± 0.13	OOT	OOT	0.68 ± 0.18	0.65 ± 0.06
ABNet	0.90 ± 0.05	0.58 ± 0.06	OOM	OOM	OOM	OOM	OOM
HERDGNN (Ours)	0.88 ± 0.03	0.68 ± 0.05	**0.83 ± 0.12**	**0.84 ± 0.03**	**0.75 ± 0.09**	**0.90 ± 0.02**	0.86 ± 0.03
				Mean AUNPRC			
GCN	0.23 ± 0.06	0.09 ± 0.03	0.19 ± 0.16	0.03 ± 0.002	0.002 ± 0.002	0.004 ± 0.005	0.007 ± 0.004
GIN	0.27 ± 0.07	0.08 ± 0.03	0.18 ± 0.15	0.03 ± 0.003	0.003 ± 0.004	0.004 ± 0.005	0.005 ± 0.003
GAT	0.23 ± 0.04	0.09 ± 0.03	0.22 ± 0.15	0.03 ± 0.001	0.002 ± 0.002	0.005 ± 0.006	0.007 ± 0.004
EvolveGCN-O	0.35 ± 0.05	0.10 ± 0.04	0.19 ±0.13	0.03 ± 0.001	0.002 ± 0.001	0.007 ± 0.003	0.004 ± 0.002
ROLAND-mlp	0.40 ± 0.08	0.08 ± 0.03	0.17 ± 0.15	OOM	OOM	OOM	OOT
ROLAND-moving	0.43 ± 0.1	0.09 ± 0.02	0.05 ± 0.08	OOM	OOM	OOM	OOT
ROLAND-gru	0.45 ± 0.05	0.11 ± 0.03	0.24 ± 0.18	OOM	OOM	OOM	OOT
HawkGCN	**0.56 ± 0.04**	**0.12 ± 0.04**	0.05 ± 0.10	OOM	OOM	0.007 ± 0.005	0.006 ± 0.003
HawkGAT	0.34 ± 0.04	0.09 ± 0.03	0.07 ± 0.11	OOM	OOM	0.010 ± 0.006	0.004 ± 0.002
WinGNN	0.27 ± 0.08	0.06 ± 0.01	0.04 ± 0.08	OOT	OOT	0.003 ± 0.002	0.002 ± 0.002
ABNet	0.49 ± 0.09	0.08 ± 0.04	OOM	OOM	OOM	OOM	OOM
HERDGNN (Ours)	0.48 ± 0.05	**0.12 ± 0.05**	**0.28 ± 0.18**	**0.09 ± 0.03**	**0.005 ± 0.004**	**0.013 ± 0.006**	**0.010 ± 0.007**

Table 5: Ablation study on Elliptic++ and SAML.

Components			Elliptic++		SAML	
M	I	D	AUNPRC	AUROC	AUNPRC	AUROC
✓	✓	✓	**0.28**	**0.83**	**0.013**	**0.90**
✓	✗	✓	0.20	0.78	0.007	0.79
✗	✓	✓	0.24	0.78	0.011	0.87
✓	✓	✗	0.26	0.81	0.012	0.87
✗	✓	✗	0.23	0.78	0.009	0.84
✓	✗	✗	0.20	0.78	0.005	0.76
✗	✗	✓	0.21	0.76	0.005	0.70
✗	✗	✗	0.19	0.77	0.004	0.72

5.3 Ablation Study

To evaluate the contribution of each component in HERDGNN we conduct an ablation study. To this end, we design 7 ablation models in which some of the components of HERDGNN are removed. Specifically, the hierarchical memory module (M), the incremental fine-tuning strategy (I), the deviation-based loss regularization and the blending mechanism (D). The results of the ablation models on Elliptic++ and SAML are reported in Table 5. Marks ✓/✗ indicate whether a component is active/inactive.

Generally, the obtained results confirm our design choices. We observe that the complete model outperforms the ablation models on both datasets, indicating the importance of every component. More specifically, removing incremental training (I) causes a dramatic drop in mean AUNPRC and mean AUROC, showing the importance of live updates. Removing the hierarchical memory module (M) results in a substantial decrease of mean AUNPRC and mean AUROC, indicating the importance of the hierarchical memory. Further, disabling the deviation-aware regularization (D) consistently hurts AUNPRC, confirming that deviation-aware supervision brings value under extreme imbalance.

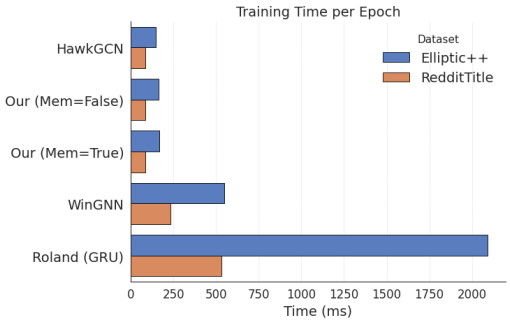


Figure 2: Training time on Elliptic++ and RedditTitle.

Table 6: Total parameters and buffers across datasets and models. Each entry is reported as *parameters / buffers*. Ours-NoMem denotes our method without memory, while Ours-M64, Ours-M128, Ours-M192, and Ours-M256 denote our method with memory dimensions 64, 128, 192, and 256, respectively. Roland-GRU, Roland-MLP, and Roland-Mov denote Roland with different update mechanisms.

Dataset	Ours-NoMem	Ours-M64	Ours-M128	Ours-M192	Ours-M256	Roland-GRU	Roland-MLP	Roland-Mov	HawkGCN	WinGNN	ABNet
RedditTitle	73K / –	2.3M / –	4.5M / –	6.8M / –	9.3M / –	287K / 13.8M	155K / 13.8M	89K / 13.8M	180K / –	89K / –	2.3M / –
RedditBody	73K / –	2.3M / –	4.5M / –	6.8M / –	9.3M / –	287K / 9.2M	155K / 9.2M	89K / 9.2M	144K / –	89K / –	2.3M / –
Elliptic++	58K / –	2.2M / –	4.5M / –	6.8M / –	9.3M / –	271K / 51.9M	139K / 51.9M	73K / 51.9M	462K / –	73K / –	1.3M / –
SAML	35K / –	2.2M / –	4.5M / –	6.8M / –	9.2M / –	249K / 219.0M	116K / 219.0M	50K / 219.0M	1.7M / –	50K / –	2.3M / –
DGraphFin	37K / –	2.2M / –	4.5M / –	6.8M / –	9.2M / –	250K / 947.3M	118K / 947.3M	52K / 947.3M	7.4M / –	52K / –	1.2M / –
EthereumPhishing	35K / –	2.2M / –	4.5M / –	6.8M / –	9.2M / –	248K / 761.2M	116K / 761.2M	50K / 761.2M	6.0M / –	50K / –	1.2M / –
AMLWorldSmall	35K / –	2.2M / –	4.5M / –	6.8M / –	9.2M / –	248K / 131.9M	116K / 131.9M	50K / 131.9M	1.1M / –	50K / –	2.3M / –

– denotes buffer values below 5K, which are relatively negligible in this comparison.

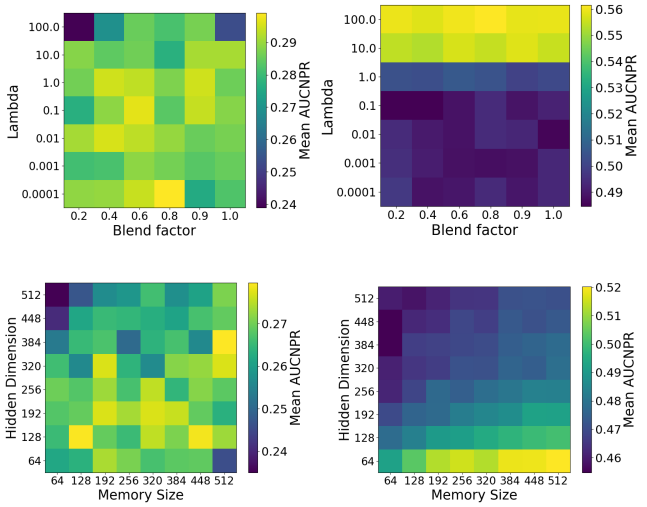


Figure 3: Sensitivity analysis on Elliptic++(first column) and RedditTitle(second column). The heatmap grid reports AUNPRC for each (lambda, blend factor) and (hidden dimension, memory size) combination.

5.4 Sensitivity Analysis

Figure 3 reports the sensitivity analysis of the principal hyperparameters of HERDGNN on the Elliptic++ and RedditTitle datasets in terms of mean AUNPRC. The top row shows multiple combinations of the deviation-loss weight λ and the blending factor ω used to update the Gaussian statistics. We observe the best results in Elliptic++ are attained for small-to-moderate λ, whereas large λ consistently degrades AUNPRC, indicating that $\mathcal{L}_{\text{dev}}$ is most effective as a regularizer rather than a dominant objective. Notably, the optimal λ is dataset-dependent, for example, in our hyperparameter search on RedditTitle we observed the best performance at a much larger value ($\lambda \approx 100$). Across settings, performance is typically strongest for high but non-unit blending (often $\omega \approx 0.8$–0.9), suggesting that our mechanism of temporally updating the deviation loss parameters further boosts the model's ability to transfer learning from the past iterations. The lower heatmaps in Figure 3 examine sensitivity to memory size and hidden dimension. The first thing to note is that despite a broad range of memory size and hidden dimension values we studied, the performance does not decrease by more than 10% of the best performance on any of the datasets. In Elliptic++, a specific trend is difficult to isolate, apart from the conclusion that smallest memory sizes and highest dimensions are consistently detrimental to the performance. In contrast, in RedditTitle, HERDGNN shows a strong trend in favor of high memory size coupled with low hidden dimensionality.

The above show that HERDGNN is stable across a broad range of hyper parameters, but the hyperparameter tuning strategy might need to differ depending on the dataset.

5.5 Memory Footprint

In this section, we complement the theoretical memory complexity analysis in Table 2 with empirical results. Since all models use the same hidden size of 128, the reported values mainly reflect architectural differences in temporal modeling rather than differences in hidden-size configuration.

5.5.1 Number of parameters. We first report the total number of trainable parameters and registered buffers for each method across datasets, in Table 6. As the table shows, ROLAND's high memory consumption is dominated by the buffer data structures and not by its parameters, and it is particularly memory inefficient for the larger networks. The number of parameters of the other related work methods strongly vary depending on the dataset, with WinGNN being particularly efficient. As seen in the table, HERDGNN's parameter size is a function of allocated memory size and is independent from the dataset.

5.5.2 Memory scalability analysis. To assess memory scalability under controlled graph growth, we conduct full-batch training experiments on two families of synthetic graphs: random Erdős–Renyi (ER) graphs [19] and power-law degree distributed Barabási–Albert (BA) graphs [3]. For the ER graphs, we consider two complementary scaling regimes: (i) increasing the number of nodes while fixing the number of edges (5M), and (ii) increasing the number of edges while fixing the number of nodes (500K). As for BA graphs, we generate them with attachment parameter $m = 5$.

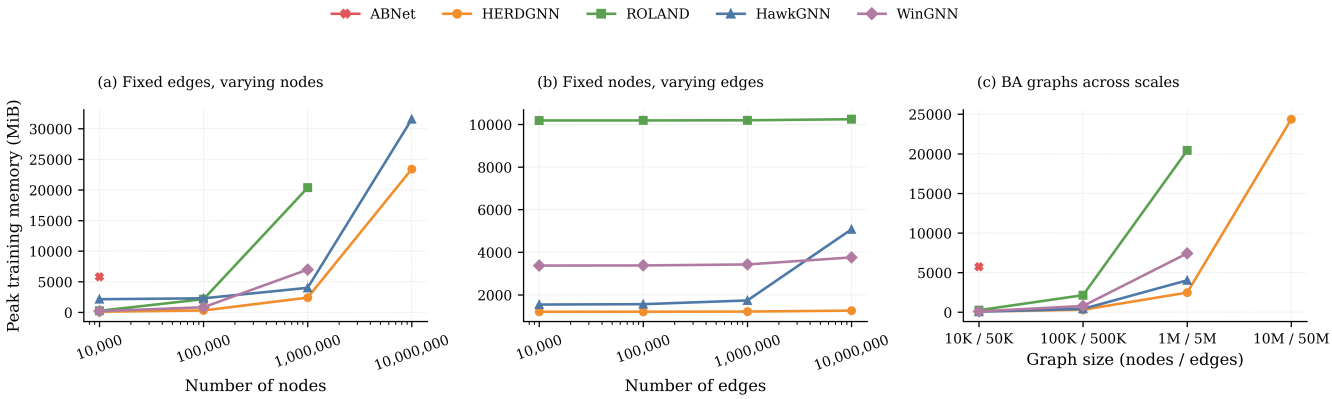


Figure 4: Memory scalability on synthetic dynamic graphs: (a) dynamic Erdős–Renyi graphs with fixed edges and increasing nodes, (b) dynamic Erdős–Rényi graphs with fixed nodes and increasing edges, and (c) dynamic Barabási–Albert graphs of increasing size ($m = 5$).

Table 7: Deviation-based Regularization (Mean AUNPRC)

Model	Elliptic++		RedditTitle		
	$\mathcal{L}_{cls}$	$\mathcal{L}_{cls} + \lambda\mathcal{L}_{dev}$	$\mathcal{L}_{cls}$	$\mathcal{L}_{cls} + \lambda\mathcal{L}_{dev}$	↑ Avg.
Roland-gru	0.25	0.28 (↑ 12%)	0.48	0.53 (↑ 10%)	↑ 11%
HawkGCN	0.05	0.06 (↑ 20%)	0.56	0.56	↑ 10%
WinGNN	0.04	0.04	0.27	0.28 (↑ 4%)	↑ 2%
HERDGNN	0.26	0.28 (↑ 7%)	0.43	0.49 (↑ 14%)	↑ 11%

To create dynamic graphs, we first generate a static graph, and afterwards we randomly assign each of its edges to one out of 30 timestamps. In all settings, each node is assigned 20 features and each edge 10 features. We report peak training memory in MiB for all methods.

Figure 4 shows that HERDGNN exhibits substantially more stable memory scaling than competing methods. ROLAND scales poorly with the number of nodes because it maintains node-level temporal representations, while WinGNN and ABNet incur substantial overhead from multi-step/meta-learning and temporal processing. HawkGNN scales well with nodes but becomes sensitive to the number of edges. On the largest BA graphs, all baselines except HERDGNN encounter OOM, confirming HERDGNN's advantage across both random and scale-free dynamic graphs.

Putting memory complexity and prediction performance together, our experiments show that HERDGNN manages to strike a balance between the two aspects. Among the real-world datasets, this result is particularly reflected on Elliptic++, as seen in Figure 5.

5.6 Deviation Regularization on the Baselines

Last, we assess the value of our proposed deviation-based regularization term $\mathcal{L}_{dev}$ beyond HERDGNN. For this purpose, we trained some of the competitor methods by including the deviation-based regularization into the BCE loss. The classification results in terms of mean AUNPRC are shown in Table 7. The vanilla BCE loss is denoted by $\mathcal{L}_{\text{cls}}$, and the regularized objectives are referred to as $\mathcal{L}_{\text{cls}} + \lambda\mathcal{L}_{\text{dev}}$.

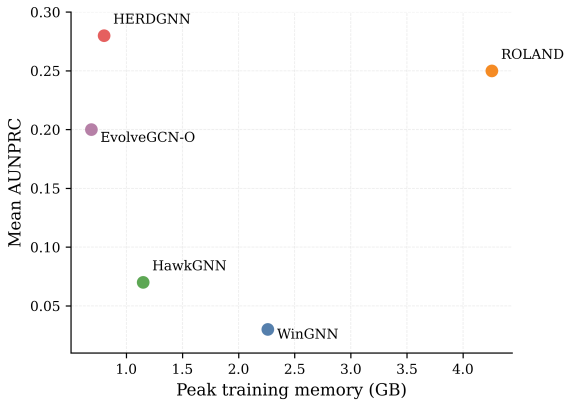


Figure 5: Mean Normalized AUPRC (Mean AUNPRC) versus peak memory consumption during training (GB) on the Elliptic++ dataset for state-of-the-art DGNNs (*High/Left is better*).

We observe that regularizing the vanilla BCE loss via $\mathcal{L}_{\text{dev}}$ improves, on average, the performance in terms of mean AUNPRC for all methods. Overall, these results indicate that $\mathcal{L}_{\text{dev}}$ is an effective regularizer helping models to attain better classification results in cases of extreme class imbalance.

6 Conclusion

In this work we proposed the first DGNN (HERDGNN) tailored for class-imbalanced settings while keeping a bounded temporal memory footprint. Experimentally, HERDGNN achieves competitive or superior performance on 7 large-scale real and synthetic financial networks compared to the state-of-the-art DGNNs, and overall strikes the best balance between classification performance and memory consumption.

References

[1] Charu C Aggarwal, Yuchen Zhao, and S Yu Philip. 2011. Outlier detection in graph streams. In *2011 IEEE 27th international conference on data engineering*. IEEE, 399–409.

[2] Erik Altman, Jovan Blanuša, Luc Von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. 2023. Realistic synthetic financial transactions for anti-money laundering models. *Advances in Neural Information Processing Systems* 36 (2023), 29851–29874.

[3] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[4] Siddharth Bhatia, Rui Liu, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2022. Real-time anomaly detection in edge streams. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 16, 4 (2022), 1–22.

[5] Siddharth Bhatia, Mohit Wadhwa, Kenji Kawaguchi, Neil Shah, Philip S Yu, and Bryan Hooi. 2023. Sketch-based anomaly detection in streaming graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 93–104.

[6] Jovan Blanuša, Maximo Cravero Baraja, Andreea Anghel, Luc Von Niederhäusern, Erik Altman, Haris Pozidis, and Kubilay Atasu. 2024. Graph Feature Preprocessor: Real-time Subgraph-based Feature Extraction for Financial Crime Detection. In *Proceedings of the 5th ACM International Conference on AI in Finance*. 222–230.

[7] Kendrick Boyd, Vitor Santos Costa, Jesse Davis, and C David Page. 2012. Unachievable region in precision-recall space and its effect on empirical evaluation. In *ICML*, Vol. 2012. NIH Public Access, 349.

[8] Kendrick Boyd, Kevin H Eng, and C David Page. 2013. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 451–466.

[9] Andrew P. Bradley. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition* 30, 7 (1997), 1145–1159. doi:10.1016/S0031-3203(96)00142-2

[10] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. 2021. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the 30th ACM international conference on Information & Knowledge Management*. 3747–3756.

[11] Liang Chen, Jiaying Peng, Yang Liu, Jintang Li, Fenfang Xie, and Zibin Zheng. 2021. Phishing scams detection in Ethereum transaction network. *ACM Transactions on Internet Technology (TOIT)* 21, 1 (2021), 1–16. doi:10.1145/3398071

[12] Yuhang Chen, Jiaxin Jiang, Shixuan Sun, Bingsheng He, and Min Chen. 2024. Rush: Real-time burst subgraph detection in dynamic graphs. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3657–3665.

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* 1412 (2014).

[14] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. 2023. Do We Really Need Complicated Model Architectures For Temporal Networks?. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=ayPPc0SyLv1

[15] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[16] da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan. 2020. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*.

[17] Ocheme Anthony Ekle and William Eberle. 2024. Anomaly detection in dynamic graphs: A comprehensive survey. *ACM Transactions on Knowledge Discovery from Data* 18, 8 (2024), 1–44.

[18] Youssef Elmougy and Ling Liu. 2023. Demystifying fraudulent transactions and illicit nodes in the bitcoin network for financial forensics. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3979–3990.

[19] Paul Erdős and Alfréd Rényi. 1959. On random graphs I. *Publicationes Mathematicae Debrecen* 6, 3–4 (1959), 290–297. doi:10.5486/PMD.1959.6.3-4.12

[20] Dhivya Eswaran and Christos Faloutsos. 2018. Sedanspot: Detecting anomalies in edge streams. In *2018 IEEE International conference on data mining (ICDM)*. IEEE, 953–958.

[21] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. 2018. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1378–1386.

[22] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[23] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR, 1126–1135.

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

[25] Siho Han and Simon S Woo. 2022. Learning sparse latent graph representations for anomaly detection in multivariate time series. In *Proceedings of the 28th ACM SIGKDD Conference on knowledge discovery and data mining*. 2977–2986.

[26] Alan G Hawkes. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika* 58, 1 (1971), 83–90.

[27] Yifan Hong, Muhammad Asif Ali, Huan Wang, Junyang Chen, and Di Wang. 2025. ABNet: Mitigating Sample Imbalance in Anomaly Detection Within Dynamic Graphs. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*. 2910–2918.

[28] Xuanwen Huang, Yang Yang, Yang Wang, Chunping Wang, Zhisheng Zhang, Jiarong Xu, Lei Chen, and Michalis Vazirgiannis. 2022. Dgraph: A large-scale financial dataset for graph anomaly detection. *Advances in Neural Information Processing Systems* 35 (2022), 22765–22777.

[29] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[30] Jiaxin Jiang, Yuan Li, Bingsheng He, Bryan Hooi, Jia Chen, and Johan Kok Zhi Kang. 2022. Spade: A real-time fraud detection framework on evolving graphs. *Proceedings of the VLDB Endowment* 16, 3 (2022), 461–469.

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

[32] Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[33] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[34] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. 2018. Community interaction and conflict on the web. In *Proceedings of the 2018 world wide web conference*. 933–943.

[35] Xujia Li, Yuan Li, Xueying Mo, Hebing Xiao, Yanyan Shen, and Lei Chen. 2023. Diga: Guided diffusion model for graph recovery in anti-money laundering. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4404–4413.

[36] Junhong Lin, Xiaojie Guo, Yada Zhu, Samuel Mitchell, Erik Altman, and Julian Shun. 2024. FraudGT: a simple, effective, and efficient graph transformer for financial fraud detection. In *Proceedings of the 5th ACM International Conference on AI in Finance*. 292–300.

[37] Yixin Liu, Shirui Pan, Yu Guang Wang, Fei Xiong, Liang Wang, Qingfeng Chen, and Vincent CS Lee. 2021. Anomaly detection in dynamic graphs via transformer. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2021), 12081–12094.

[38] Shiqi Lou, Qingyue Zhang, Shujie Yang, Yuyang Tian, Zhaoxuan Tan, and Minnan Luo. 2023. GADY: Unsupervised anomaly detection on dynamic graphs. *arXiv preprint arXiv:2310.16376*.

[39] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. 2021. A comprehensive survey on graph anomaly detection with deep learning. *IEEE transactions on knowledge and data engineering* 35, 12 (2021), 12012–12038.

[40] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. 2016. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1035–1044.

[41] Gongxun Miao, Guohua Wu, Zhen Zhang, Yongjie Tong, and Bing Lu. 2023. Addag-ae: Anomaly detection in dynamic attributed graph based on graph attention network and lstm autoencoder. *Electronics* 12, 13 (2023), 2763.

[42] Berkan Oztas, Deniz Cetinkaya, Festus Adedoyin, Marcin Budka, Huseyin Dogan, and Gokhan Aksu. 2023. Enhancing Anti-Money Laundering: Development of a Synthetic Transaction Monitoring Dataset. In *2023 IEEE International Conference on e-Business Engineering (ICEBE)*. IEEE, 47–54.

[43] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford infolab.

[44] Guansong Pang, Chunhua Shen, and Anton Van Den Hengel. 2019. Deep anomaly detection with deviation networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 353–362.

[45] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 5363–5370.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[47] QingGuo Qi, Hongyang Chen, Minhao Cheng, and Han Liu. 2025. Input Snapshots Fusion for Scalable Discrete-Time Dynamic Graph Neural Networks. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*. 1138–1149.

[48] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F Samatova. 2016. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *Proceedings of the 2016 SIAM international conference on data mining*. SIAM, 189–197.

[49] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.

[50] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 519–527.

[51] Sheng Tian, Jihai Dong, Jintang Li, Wenlong Zhao, Xiaolong Xu, Baokun Wang, Bowen Song, Changhua Meng, Tianyi Zhang, and Liang Chen. 2023. SAD: semi-supervised anomaly detection on dynamic graphs *(IJCAI '23)*. Article 256, 9 pages. doi:10.24963/ijcai.2023/256

[52] Rafaël Van Belle, Bart Baesens, and Jochen De Weerdt. 2023. CATCHM: A novel network-based credit card fraud detection method using node representation learning. *Decision Support Systems* 164 (2023), 113866.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[54] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

[55] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[56] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[57] Tianpeng Wei, Biyang Zeng, Wenqi Guo, Zhenyu Guo, Shikui Tu, and Lei Xu. 2023. A dynamic graph convolutional network for anti-money laundering. In *International Conference on Intelligent Computing*. Springer, 493–502.

[58] Jiajing Wu, Qi Yuan, Dan Lin, Wei You, Weili Chen, Chuan Chen, and Zibin Zheng. 2020. Who are the phishers? phishing scam detection on ethereum via network embedding. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, 2 (2020), 1156–1166.

[59] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[60] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. 2019. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 647–657.

[61] Jiaxuan You, Tianyu Du, and Jure Leskovec. 2022. ROLAND: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*. 2358–2366.

[62] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. 2019. AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN.. In *IJCAI*, Vol. 3. 7.

[63] Yifan Zhu, Fangpeng Cong, Dan Zhang, Wenwen Gong, Qika Lin, Wenzheng Feng, Yuxiao Dong, and Jie Tang. 2023. Wingnn: Dynamic graph neural networks with random gradient aggregation window. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*. 3650–3662.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009