WinGNN: Dynamic Graph Neural Networks with Random Gradient Aggregation Window

Yifan Zhu
Tsinghua University
zhuyifan@tsinghua.edu.cn

Fangpeng Cong
Yanshan University
pnp@stumail.ysu.edu.cn

Dan Zhang
Tsinghua University
zd21@mails.tsinghua.edu.cn

Wenwen Gong
Tsinghua University
wenweng@mail.tsinghua.edu.cn

Qika Lin
Xi'an Jiaotong University
qikalin@foxmail.com

Wenzheng Feng
Tsinghua University
wenzhengfeng96@gmail.com

Yuxiao Dong
Tsinghua University
yuxiaod@tsinghua.edu.cn

Jie Tang*
Tsinghua University
jietang@tsinghua.edu.cn

ABSTRACT

Modeling the dynamics into graph neural networks (GNNs) contributes to the understanding of evolution in dynamic graphs, which helps optimize temporal-spatial representations for real-world dynamic network problems. Empirically, dynamic GNN embedding requires additional temporal encoders, which inevitably introduces additional learning parameters to make dynamic GNNs oversized and inefficient. Furthermore, previous dynamic GNN models are under the same fixed temporal term, which causes the short-temporal optimum. To address these issues, we propose the WinGNN framework to model dynamic graphs, which is realized by a simple GNN model with the meta-learning strategy and a novel mechanism of random gradient aggregation. WinGNN calculates the frame-wise loss of the current snapshot and passes the loss gradient to the next to model graph dynamics without temporal encoders. Then it introduces the randomized sliding-window to acquire the window-aware gradienton consecutive snapshots, and the calculated two types of gradient are aggregated to update the GNN, thereby reducing the parameter size and improving the robustness. Experiments on six public datasets show the advantage of our WinGNN compared with existing baselines, where it has reached the optimum in twenty-two out of twenty-four performance metrics.

CCS CONCEPTS

• **Information systems → Data mining**; • **Theory of computation → Dynamic graph algorithms**.

KEYWORDS

Temporal encoder-free GNN, multi gradient aggregation, dynamic graph neural networks, dynamic graph representation learning.

*Corresponding author.

KDD '23, August 6–10, 2023, Long Beach, CA, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599551

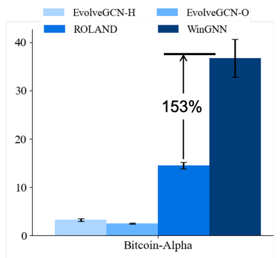
ACM Reference Format:
Yifan Zhu, Fangpeng Cong, Dan Zhang, Wenwen Gong, Qika Lin, Wenzheng Feng, Yuxiao Dong, and Jie Tang. 2023. WinGNN: Dynamic Graph Neural Networks with Random Gradient Aggregation Window. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3580305.3599551

1 INTRODUCTION

Most graphs in real-world scenarios are essentially dynamic networks, such as social networks, traffic networks, online behaviors, etc., which characterize the variation of entities and relations in the network over time [12, 30]. Capturing and modeling such dynamics provides a unique temporal view of the network variability to better understand network evolution, which has been studied extensively in decades [1, 42]. Recently, graph neural networks (GNNs) have produced rapid progress in representing the graph by propagating nodal information via edges to obtain low-dimensional embedding [7, 14, 35]. Under these circumstances, incorporating graph evolution information into GNNs (i.e., dynamic GNNs) shows a prominent prospect and has attracted considerable attention in very recent years [16, 44]. Despite the great success, there are still two following limitations.

Model	TE-Free	LSTS	Meta	LG
EvolveGCN	×	√	×	△
dyngraph2vec	×	√	×	×
DGNN	×	√	×	×
DEFT	√	×	×	△
DDGCL	√	×	×	×
ROLAND	×	×	√	√
WinGNN	√	√	√	√

(a) Categorization comparison



(b) Performance comparison

Figure 1: Comparison between WinGNN and recent dynamic GNNs. (a) Categorization comparison. TE-Free denotes temporal-free design, LSTS denotes the model considers long and short temporal scale feature explicitly, Meta denotes the model adapts a meta-learning strategy, and LG denotes the model is capable of large dynamic graph with more than 50 million edges (△ denotes extremely poor performance). (b) MRR performance improved by WinGNN on Bitcoin-Alpha.

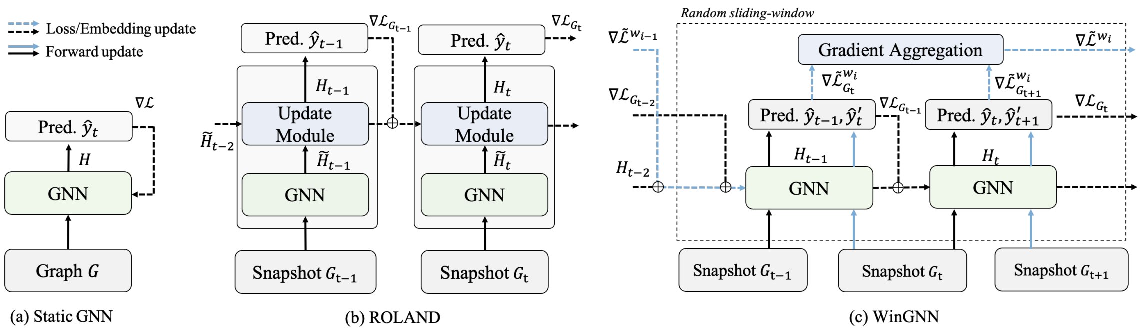

Loss/Embedding update
Forward update
Pred. $\hat{y}_t$
$\nabla\mathcal{L}$
H
GNN
Graph G
(a) Static GNN
Pred. $\hat{y}_{t-1}$
$\nabla\mathcal{L}_{G_{t-1}}$
Pred. $\hat{y}_t$
$\nabla\mathcal{L}_{G_t}$
H_{t-1}
H_t
Update Module
$\widetilde{H}_{t-2}$
$\widetilde{H}_{t-1}$
$\widetilde{H}_t$
Snapshot G_{t-1}
Snapshot G_t
(b) ROLAND
Random sliding-window
$\nabla\tilde{\mathcal{L}}^{w_{i-1}}$
Gradient Aggregation
$\nabla\tilde{\mathcal{L}}^{w_i}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_t}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t+1}}$
$\nabla\mathcal{L}_{G_{t-2}}$
$\nabla\mathcal{L}_{G_{t-1}}$
$\nabla\mathcal{L}_{G_t}$
Pred. $\hat{y}_{t-1}, \hat{y}'_t$
Pred. $\hat{y}_t, \hat{y}'_{t+1}$
H_{t-2}
Snapshot G_{t+1}
(c) WinGNN

Figure 2: Propagation of loss gradient between static GNN, ROLAND [44], and the proposed WinGNN. The black dotted arrows and blue dotted arrows in (c) denote the frame-wise and window-aware loss gradient, respectively.

Limitation 1: the *temporal encoder-dependent* issue. It indicates that many existing dynamic methods first utilize the GNN to model the internal information of the static subgraph at a specific time (noted as a snapshot). Then the dynamics among consecutive snapshots are acquired by previous GNN embeddings using some temporal sequence encoders, e.g., long short-term memory (LSTM) [24] and Transformer [26]. Although these methods are simple and intuitive, they inevitably introduce additional learning parameters to the whole model and have the risk of an over-fitting problem. Therefore, realizing a temporal encoder-free model to remove the use of these temporal encoders can improve the parameter efficiency of the model, thus resulting in performance increment. However, it is challenging because the direct removal of the temporal encoder will cause the intractable modeling of graph dynamics.
Limitation 2: the *short-temporal optimum* issue. Meta-learning, as a transfer learning method, is able to model the temporal dynamics as proved from both theoretical and experimental perspectives [22, 23]. The idea of meta-learning provides a new perspective to reduce the extra temporal encoding cost of model parameters. For example, MetaDyGNN [43] updates its global parameter for each temporal interval via hierarchical adaptions. ROLAND [44] utilizes a hybrid framework that forwards the learned embedding and loss on the snapshot to the next one as Figure 2-(b) shows. It only integrates the information of two adjacent time steps. Generally, these models neglect temporal information with long-term dependence, which could lead to the short-temporal optimum and cause poor performance. Therefore, it is essential to design a robust dynamic GNN model with a more comprehensive meta-learning strategy.

Taking the above limitations together, we study the key research question ***Is there a temporal encoder-free GNN design that can learn representations from dynamic graphs efficiently and robustly?*** In this paper, as illustrated in Figure 2-(c), we propose a novel dynamic GNN framework called WinGNN (abbreviation of window GNN) to establish efficient and robust dynamic graph representation learning. First, we incorporate a meta-learning strategy into a simple GNN structure, where the frame-wise loss is calculated on the current snapshot and then the loss gradient is passed to the next snapshot. Thus WinGNN establishes a representation association between two adjacent snapshots and removes all temporal-specific encoders, thereby improving the generalization and parameter efficiency. Second, we propose a novel mechanism of random gradient aggregation to model the temporal information with long-term dependence. It introduces the randomized sliding-window to fuse the gradient of long-term consecutive snapshots, where an adaptive gradient aggregation is proposed to determine which snapshot to keep or ignore, and acquire the window-aware gradient. Meanwhile, the randomness of these sliding-windows brings different temporal scales, making WinGNN robustly fine-tune its parameters for graph dynamics. Finally, the frame-wise and window-aware gradients are integrated together to optimize the GNN of the next snapshot, forming an iterative calculation process with time increasing. We compare WinGNN with other dynamic GNN studies from four categorized aspects in Figure 1-(a), which intuitively shows the advantages of our WinGNN.

We summarize the main contribution in our study as follows:

- A unified framework WinGNN is proposed that realizes a temporal encoder-free GNN design efficiently and robustly. It utilizes the meta-learning strategy to model the association of snapshots that are adjacent and consecutive in a sliding-window.
- To acquire robust representations, a novel mechanism of random gradient aggregation is proposed by introducing the randomized sliding-window and adaptive gradient aggregation. We also reveal the significance of modeling different temporal scales in temporal encoder-free meta-learning for dynamic GNNs.
- Extensive experiments on six public datasets are performed, and the superiority of WinGNN is demonstrated by the results, which can achieve the maximum improvement of 153% on MRR metric as Figure 1-(b) shows. Meanwhile, WinGNN can outperform the state-of-the-art baseline ROLAND even on a very large dynamic graph with more than 60 million edges.

2 PRELIMINARIES

2.1 Problem Formulation

Let $\mathcal{G} = \{G_t\}_{t=1}^T$ denote a dynamic graph that contains a list of graph snapshots. $\mathcal{V} = \{v_1, ...v_n\}$ is a finite set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. Each snapshot is a static graph $G_t = (V_t, E_t)$ where V_t and E_t are the node set and edge set of the snapshot. They form $\mathcal{V} = \bigcup_{t=1}^T V_t$ and $\mathcal{E} = \bigcup_{t=1}^T E_t$. The adjacent matrix of a snapshot at the time t is denoted as $A_t \in \mathbb{R}^{n\times n}$. If $v_i, v_j \in V_t$ and $(v_i, v_j) \in E_t$, then $A_{i,j}^t = 1$ and otherwise it is 0. A feature matrix $\boldsymbol{X} \in \mathbb{R}^{n\times k} = \{\boldsymbol{x}_v | v \in \mathcal{V}\}$ is usually provided for nodes, where k is the feature dimension. Similar to many graph learning studies, link prediction is an ideal task for evaluating dynamic graph representation learning. It requires the model to learn a binary classification function f to predict whether there is an unobserved edge at time $t+1$ by giving two nodes at previous snapshots:

$$f : (G_{1:t}, \boldsymbol{X}) \longrightarrow E_{t+1}. \tag{1}$$

2.2 Graph Neural Networks

The essence of GNNs is to learn the node representations via iteratively aggregating information from neighbor nodes. By applying a l-layer GNN, we denote the representation (also known as the embedding) of node u as $\boldsymbol{h}_u$, which is calculated by:

$$\boldsymbol{h}_u^{(l)} = \text{UPDATE}(\boldsymbol{h}_u^{(l-1)}, \text{AGG}(\{\boldsymbol{h}_v^{(l-1)}, \forall v \in \mathcal{N}_u\})), \tag{2}$$

where $\boldsymbol{h}_u^{(l)}$ is the node representation after a l-layer GNN, $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$, $\mathcal{N}_u$ is the direct neighbors of u. AGG$(\cdot)$ and UPDATE$(\cdot)$ stand for aggregation and update functions which can have different designs in different studies. Take GCN [14] as an example, this process is defined in a matrix form:

$$\boldsymbol{H}^{(l)} = \sigma(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)}), \tag{3}$$

where $\boldsymbol{H}^{(l)} \in \mathbb{R}^{n \times d}$ is the representation of the nodes derived at l-th layer, $\hat{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ denotes the normalized adjacent matrix given that $\boldsymbol{A}$ is the adjacent matrix with self-loops and $\boldsymbol{D}$ is the corresponding diagonal degree matrix. $\boldsymbol{W}^{(l)}$ is a learnable parameter matrix at l-th layer and σ denotes a non-linear activation function. Additional weighets could also be added durting the propagation, such as attention and diffusion [46]. However, this coupled operation could cause over-smoothing issues and redundant training costs. Thus, decoupling this process would further improve parameter efficiency [2, 39]. Generally, a decoupled GCN could be expressed as:

$$\boldsymbol{H} = g_{\theta_1}(\boldsymbol{H}^{(0)}, ..., \boldsymbol{H}^{(l)}),\ \boldsymbol{H}^{(l)} = \hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)},\ \boldsymbol{H}^{(0)} = g_{\theta_2}(\boldsymbol{X}), \tag{4}$$

where g_{θ_1} and g_{θ_2} are parameter-learnable neural networks, and $\boldsymbol{Z}$ is the model outputted representation.

Tgeneralizing to dynamic graphs, dynamic GNNs need to learn the graph structure under different time units through different information propagation $(\boldsymbol{A}_1, \boldsymbol{A}_2, ..., \boldsymbol{A}_T)$, and finally generate node representations under different time units $(\boldsymbol{H}_1, \boldsymbol{H}_2, ..., \boldsymbol{H}_T)$. Therefore, dynamic GNNs require additional operations and components to be capable of the loss calculated by Eq. (1).

2.3 Meta-Learning

Inspired by the idea of learning to learn, the goal of meta-learning is to train a model that is able to be quickly adapted to novel tasks by using a few new data records and training iterations. Typically, the model or learner is preliminarily trained on a set of specific tasks to derive the initial embedding and model parameters. Then, facing a new task that has similar informative embedding requirements to the preliminary tasks, the meta-learning model only needs to fine-tune its parameters by using a small number of samples. Formally, each task $\mathcal{T} = \{\mathcal{L}(x_1, a_1, \ldots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$ consists of loss function $\mathcal{L}$. The distribution on the initial observation $q(x_1)$, transition distribution $q(x_{t+1}|x_t, a_t)$, and event length H. Meta-learning defines a two-stage learning paradigm, namely inner loop and outer loop. In the inner loop phase, the model is trained based on each task to produce a list of parameters $\theta_{0:l} \equiv \{\theta_0, \cdots, \theta_{l-1}\}$. Then, the parameter is tuned by comprehensively optimizing all tasks $P_{\theta_{0:l},w}$ in the outer loop, such as $P_{\theta_{0:l},w}(\cdot) = \sum_{j=0}^{l} \omega_j P_{\theta_j,w}(\cdot)$.

The tasks in meta-learning can also be regarded as states at different times. Under this setting, the model learns embedding at each temporal unit as an inner loop stage with preliminarily trained parameters, and updates it by fine-tuning the model's parameters as an outer loop stage according to the optimization on all following temporal units.

3 THE WinGNN MODEL

3.1 Overview of WinGNN

We introduce the proposed WinGNN model in this section, as the overall framework is presented in Figure 3. Inspired by the observation that current dynamic GNN models suffer from short-temporal optimum, we take the advantage of meta-learning to jointly optimize parameters based on multi-snapshot gradients. In particular, WinGNN has removed all temporal-specific encoders to learn from the evolution of dynamic graphs. Then, we incorporate randomized window gradient propagation and adaptive gradient aggregation strategy into sliding-window-based multi-snapshot gradient aggregation, thereby establishing both effective and robust dynamic graph representation learning.

3.2 The Frame-Wise Dynamic GNN Unit

Revisiting loss gradient propagation. We revisit the propagation of loss gradient in static GNNs and existing dynamic GNNs. Illustrated in Figure 2-(a), a static GNN learns data on a single graph, and the model loss is generated from the label and prediction. The generated loss is then back-propagated to update the parameters of the GNN layer and other parameter-learnable modules. It is obvious that such a design makes the GNN only learn from the intrinsic graph structure without any dynamic information, and therefore it cannot be directly applied to a dynamic graph without considering the dynamics of graphs.

As an intuitive solution, early dynamic GNNs added temporal encoders before or after aggregating multiple graph encoders to capture the graph dynamics [17, 21]. However, such a process usually incorporates temporal encoders such as LSTM [24] and Transformer [26], and therefore has much more model parameters to aggravate the training difficulty and over-fitting risk. Recent studies have designed more efficient strategies to relieve such problems. Take ROLAND [44] for example, as illustrated in Figure 2-(b), it introduces a more dynamics-compatible GNN design with BatchNorm [10] as well as residual connection, and incorporates two length-limited GRU layers to enhance the parameter efficiency. From the view of the loss gradient, it generates loss on each snapshot, and back-propagates to update model parameters for the next snapshot. However, as we show in Section 1, this snapshot-by-snapshot design could still mislead the optimization into short-temporal optimum, which will degrade the performance when edge distribution changes dramatically over the snapshots.

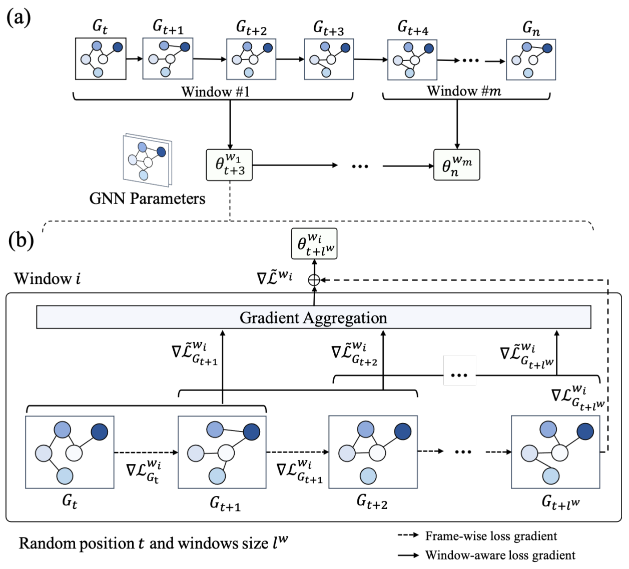

(a)
G_t
G_{t+1}
G_{t+2}
G_{t+3}
G_{t+4}
G_n
Window #1
Window #m
$\theta^{w_1}_{t+3}$
$\theta^{w_m}_{n}$
GNN Parameters
(b)
$\theta^{w_i}_{t+l^w}$
Window i
$\nabla\tilde{\mathcal{L}}^{w_i}$
Gradient Aggregation
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t+1}}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t+2}}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t+l^w}}$
$\nabla\mathcal{L}^{w_i}_{G_{t+l^w}}$
$\nabla\mathcal{L}^{w_i}_{G_t}$
$\nabla\mathcal{L}^{w_i}_{G_{t+1}}$
G_t
G_{t+1}
G_{t+2}
G_{t+l^w}
Random position t and windows size l^w
Frame-wise loss gradient
Window-aware loss gradient

Figure 3: The pipeline of WinGNN. (a) WinGNN uses a sliding-window strategy with random starting position t and window size l^w to capture graph dynamics under different temporal scales and then tunes the model parameter through the aggregation of window-aware loss gradients. (b) For each random sliding-window, WinGNN back-propagates the frame-wise loss gradient snapshot-by-snapshot and collects window-aware loss gradients for aggregation. When the iteration on this window is finished, the aggregated loss gradient is used to update the model parameter for the next sliding-window.

Temporal encoder-free dynamic GNN unit. Inspired by the meta-learning on multiple tasks [3], this study introduces a randomized sliding-window-based multi-snapshot gradient optimization strategy. Correspondingly, we design a simple but effective dynamic GNN architecture that removes the heavy temporal encoder. Generally, for snapshot G_t during the training process, the basic WinGNN unit is trained on the current snapshot G_t, and directly conducts prediction on the next snapshot G_{t+1} without loss back-propagation. There are two types of loss, namely frame-aware loss (black arrow in Figure 2) and window-aware loss (blue arrow in Figure 2), which are generated for each learning step. Then, we back-propagate the gradient of frame-aware loss to update model parameters for the next training step on G_{t+1} and collect window-aware loss for a long-term window-aware gradient optimization. When all window-aware losses are collected from the beginning to the end snapshot in a sliding-window, a such window-aware gradient is optimized and back-propagated to the model parameters for the next window. Formally, given a snapshot G_t, WinGNN encodes it via a l-layer decoupled GCN:

$$\boldsymbol{h}^{(l)}_{u,t} = \hat{\boldsymbol{A}}\boldsymbol{h}^{l-1}\boldsymbol{W}^{(l)}_h. \tag{5}$$

Then, WinGNN predicts the probability of edge from node u to node v through a multi-layer perceptron (MLP):

$$\hat{y}^t_{u,v} = \mathrm{MLP}(\boldsymbol{h}^{(l)}_{u,t} || \boldsymbol{h}^{(l)}_{v,t}), \tag{6}$$

where || denotes the concatenation operation. Based on the label data $\boldsymbol{Y}^t$, we can obtain the train loss via cross-entropy loss:

$$\mathcal{L}^{w_i}_{G_t} = -\sum_{(u,v)\in\mathcal{E}_t} y^t_{u,v}\log\hat{y}^t_{u,v} + (1-y^t_{u,v})\log(1-\hat{y}^t_{u,v}), \tag{7}$$

where $\mathcal{L}^{w_i}_{G_t}$ denotes the train loss at snapshot under the sliding-window w_i which is introduced in the Section 3.3 in detail. Note that such loss can also be replaced by other loss functions depending on the down-streaming tasks. Here, at each training step, we denote the model parameter as $\theta^{w_i}_t$, and the frame-wise loss gradient is formulated as the following approximation:

$$\nabla\mathcal{L}^{w_i}_{G_t} \simeq \frac{\partial\mathcal{L}^{w_i}_{G_t}}{\partial\theta^{w_i}_t}. \tag{8}$$

This gradient is then utilized to update the model parameters in the next snapshot with the general learning rate τ:

$$\theta^{w_i}_{t+1} \longleftarrow (\nabla\mathcal{L}^{w_i}_{G_t})^\tau + \theta^{w_i}_t. \tag{9}$$

3.3 Randomized Sliding-window Based Multi-snapshot Gradient Optimization

Multi-snapshot loss gradient. Furthermore, we make the model comprehensively learn from different periodical graph dynamics in a robust way. To achieve this, we define a random sliding-window strategy inspired by MAML [50], where each window is defined as a consecutive sub-list of snapshots with a random window size l^w as well as step length l^s. Note that the starting position of the next random window should always be within the previous window, and the ending position should be outside after the previous window. The size of the random window should also not exceed one-tenth of the total number of snapshots to preserve as much as possible short-term changes in the dynamic graphs [29]. We refer the i-th window from snapshot G_t to G_{t+l^w} as w_i. For each snapshot in this window, the window-aware loss is collected based on the prediction on the edges of the next snapshot, that is, the GNN unit uses training loss on the current snapshot G_t to update parameters to the next snapshot, and generates an additional loss on the next snapshot G_{t+1} for the periodical multi-snapshot gradient optimization. Thus, we have:

$$\nabla\hat{\mathcal{L}}^{w_i}_{G_t} \simeq \frac{\partial\mathcal{L}^{w_i}_{G_{t+1}}}{\partial\theta^{w_i}_t}. \tag{10}$$

Adaptive loss gradient aggregation. After the dynamic GNN unit finishes the training from the first to the last snapshot in the sliding-window w_i, we totally collected l^w loss gradients. These gradients are aggregated together to be optimized for a more comprehensive model parameter tuning than using gradients from single snapshots. Such process is denoted as $\phi(\cdot)$:

$$\nabla\hat{\mathcal{L}}^{w_i} = \phi(\nabla\hat{\mathcal{L}}^{w_i}_{G_t}, ..., \nabla\hat{\mathcal{L}}^{w_i}_{G_{t+l^w}}), \tag{11}$$

where $\nabla\hat{\mathcal{L}}^{w_i}$ is the aggregated gradient of the entire sliding-windows. The simplest way is to sum all gradients together (i.e., $\sum_{j=t}^{t+l^w}\nabla\hat{\mathcal{L}}^{w_i}_{G_j}$).

However, simply using a sum would be inevitably vulnerable to local optimum in short temporal terms. To overcome this, we design an adaptive loss gradient aggregation layer to enhance the robustness of dynamics learning. As presented in Figure 4, the

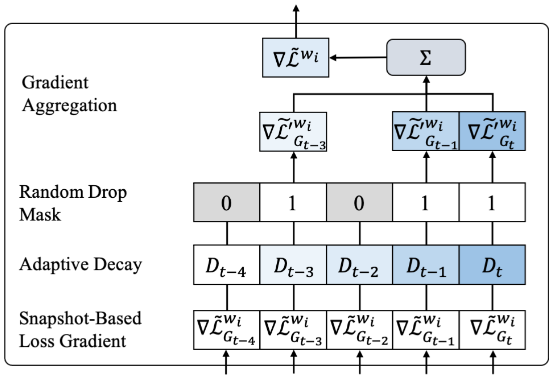

Gradient Aggregation
Random Drop Mask
Adaptive Decay
Snapshot-Based Loss Gradient
$\nabla\hat{\mathcal{L}}^{w_i}$
Σ
$\nabla\tilde{\mathcal{L}}'^{w_i}_{G_{t-3}}$
$\nabla\tilde{\mathcal{L}}'^{w_i}_{G_{t-1}}$
$\nabla\tilde{\mathcal{L}}'^{w_i}_{G_t}$
0
1
0
1
1
D_{t-4}
D_{t-3}
D_{t-2}
D_{t-1}
D_t
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t-4}}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t-3}}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t-2}}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_{t-1}}$
$\nabla\tilde{\mathcal{L}}^{w_i}_{G_t}$

Figure 4: Illustration of adaptive loss gradient aggregation at window w_i, where the window size is set to 5.

loss gradients of snapshots in the same sliding-window are first gathered. Assume there are l^w snapshots in the window which are numbered as $1, ..., t, ..., l^w$, and then we assign adaptive decay D_t to each gradient, which is calculated by:

$$\nabla\hat{\mathcal{L}}'^{w_i}_{G_t} = D_t \odot \nabla\hat{\mathcal{L}}^{w_i}_{G_t}, \tag{12}$$

$$D_t = -\frac{\tau}{\sqrt{\delta + \boldsymbol{r}_t}}, \tag{13}$$

where τ is the general learning rate of the model training, δ is a small constant, $\odot$ is the element-wise dot production, and $\boldsymbol{r}_t$ is the accumulated gradient which is calculated as:

$$\boldsymbol{r}_t = \rho\boldsymbol{r}_{t-1} + (1-\rho)\nabla\hat{\mathcal{L}}^{w_i}_{G_t} \odot \nabla\hat{\mathcal{L}}^{w_i}_{G_t}, \quad \boldsymbol{r}_0 = \boldsymbol{0}, \tag{14}$$

where ρ determines the trade-off between the previously accumulated gradient and the current loss gradient.

Afterward, to further reduce the effects from the local optimum, we introduce a snapshot-wise random loss drop called *Snapshot-Drop*. To implement this, a binary random mask vector is generated to perform element-wise product with adaptively adjusted loss gradients, and then all results are summed as the final window-aware loss gradient of the current sliding-window:

$$\nabla\hat{\mathcal{L}}^{w_i} = \sum_{t=1}^{l^w} M_t \nabla\hat{\mathcal{L}}'^{w_i}_{G_t}, \tag{15}$$

where M is the SnapshotDrop mask. With this approach, WinGNN can randomly select batches of snapshots in a dynamic graph and adaptively aggregate all loss gradients of snapshots on the batch into one comprehensive loss gradient.

For the next sliding-window, we refine the Eq.(9) by combining the window-aware multi-snapshot gradient from the previous window and frame-wise single gradient from the previous snapshot:

$$\theta^{w_{i+1}}_{t+1} \longleftarrow (\nabla\mathcal{L}^{w_i}_{G_t})^{\tau} + (\nabla\hat{\mathcal{L}}^{w_i})^{\tau} + \theta^{w_{i+1}}_{t+1}. \tag{16}$$

Then the parameters of WinGNN are iteratively updated with time increasing. Finally, we utilize the learned node representations to calculate the logits of the dynamic link prediction through an extra MLP layer in the inference stage.

3.4 Training and Evaluation

We implement the WinGNN and summarize the training algorithm in Appendix A.1. The source code of implemented WinGNN is publicly available[1], and the experimental setup is presented in Appendix A.5. The evaluation of effectiveness in this study is performed by measuring the performance of link prediction on the $t+1$ snapshot by giving the first t snapshots. We split the first 70% snapshots of each dataset as the training set, and then regard the rest 30% data as the test set. In particular, the dynamic link prediction task can be evaluated with two settings, which are classification and ranking[32]. Under the classification setting, the model is evaluated to predict whether there exists an edge between the given head and tail nodes. We randomly sample one negative edge sample for every single edge in the test snapshots and then measure the average accuracy and macro-AUC for evaluation. Under the ranking setting, the dynamic graph model is evaluated to predict which node is the tail node with a given head node. We randomly sample 1,000 negative nodes as candidates for each positive tail node in all test snapshots. However, for the large dynamic graph such as Stack Overflow which has more than 60 million edges, we only sampled 100 negative edges due to memory constraints. To avoid data leakage problems, the randomized sliding-window mechanism is only used in the training stage, and we also follow the live update setting of ROLAND [44].

4 EXPERIMENT

4.1 Datasets

We perform the experiment on six typical public datasets which are widely evaluated by existing dynamic graph representation learning studies. The datasets are selected with different edge densities and distributions, including Bitcoin-Alpha, Bitcoin-OTC, DBLP, Reddit-title, Stack Overflow and UCI. Detailed descriptions are presented in Appendix A.2. The Stack Overflow dataset includes a very large dynamic graph with more than 60 million edges, so the efficiency and performance on it can better reflect the application potential of the model. To make a fair comparison of reproducible results, we adopt the standard partition of snapshots generated by GraphGym [45]. We show the basic statistics of these six datasets in Table 1.

Table 1: The basic statistics of six datasets in the experiment.

Dataset	# Nodes	# Edges	# Snapshots	Avg. Density
Bitcoin-Alpha	3,783	24,186	226	2.5890×10^{-3}
Bitcoin-OTC	5,881	35,592	262	1.7396×10^{-4}
DBLP	28,086	162,451	27	9.5423×10^{-5}
Reddit-title	54,075	571,927	178	1.9592×10^{-5}
Stack Overflow	2,601,977	63,497,050	92	2.5503×10^{-6}
UCI	1,899	59,835	28	1.1191×10^{-3}

4.2 Baselines and Evaluation Metrics

We evaluate our proposed WinGNN by comparing the performance with various dynamic GNN baselines, including EvolveGCN [24], DGNN [21], dyngraph2vec [6], and ROLAND [44]. The detailed description of these baselines could be found in the Appendix A.3.

[1]https://github.com/thudm/WinGNN

We apply four commonly-used metrics, namely accuracy, macro-AUC, MRR, and recall@K to testify the superiority of WinGNN againest other baselines, and describe these metrics in Appendix A.4.

4.3 Main Results

The overall dynamic link prediction performance results of the proposed WinGNN and other baseline models are presented in Table 2. The improvement (%Improv.) is calculated according to the relative improvement of sub-optimal models. The model efficiency is also illustrated in Figure 5.

Dynamic link prediction under the classification setting. The classification setting evaluates the performance of dynamic link prediction via accuracy and AUC. The results show that many existing baselines, such as EvolveGCN, DGNN, and dyngraph2vec vary greatly on dynamic graph datasets with different density distributions. In other words, these methods are not effective in fully encoding spatiotemporal features when using embedded representations of the past to predict links that will occur in the future. The result shows that our proposed WinGNN model outperforms all baselines in terms of accuracy with a maximum improvement of 22.59% (Bitcoin-Alpha). It also shows similar results on AUC among all datasets. WinGNN outperforms all baselines by improving AUC from 0.1% to 2.43%. Additionally, ROLAND shows the second-best performance on most datasets, only lagging behind EvolevGCN by about 4 percent measured by accuracy on the DBLP dataset.

It is worth mentioning that when evaluating the performance on large dynamic graphs (i.e., Stack Overflow), we found that all models except WinGNN and ROLAND had out-of-memory failures, which demonstrates the importance of parameter efficiency for the generalization of dynamic graph representation learning. Furthermore, ROLAND's performance on the Stack Overflow is somehow unstable, where the results from multiple runs vary dramatically, thereby resulting in non-statistical significance between ROLAND and WinGNN. Thus, taking the above aspects together, we derive the observation that WinGNN has a better ability in classifying dynamic edges compared to the existing baselines.

Dynamic link prediction under the ranking setting. For the ranking setting, the result also shows that the proposed WinGNN outperforms all baselines in most datasets (Bitcoin-Alpha, Bitcoin-OTC, DBLP, and UCI). Especially in the dataset with more snapshots such as Bitcoin-Alpha and Bitcoin-OTC, WinGNN can significantly improve the MRR and recall@10 metric values by up to 153.03% and 106.56% (On Bitcoin-Alpha), respectively. This result shows that the graph in the long-term evolution process has adequate long-term and short-term temporal information, with more snapshots leading to the local optimum. Therefore, the randomized sliding-window-based gradient aggregation can reduce the impact of a single snapshot by randomly dropping loss gradient on unspecific snapshots. Additionally, on the large graph, similar results to the classification setting are observed as well. WinGNN shows better stability and predictive performance than ROLAND.

However, when the evolution length of the dynamic graph decreases and sparsity increases, the performance advantage brought by WinGNN falls. When the average density was reduced to the level of 10^{-5}, the WinGNN model only increased by 13.03% and 23.26% compared with the second-best model in the DBLP dataset in terms of MRR and recall@10. It is also worth mentioning that ROLAND achieves the best performance among the baselines in the majority of scenarios (Bitcoin-Alpha, DBLP, Reddit-title), but on smaller datasets, such as UCI, the dyngraph2vec model achieves even better performance with a thousand times more model parameters. For the Reddit-tittle dataset, WinGNN performs even worse than ROLAND does. This result is also related to the distribution of edges in dynamic graphs. The long evolution and the relatively dense connection make the dynamic graph itself have certain redundancy so that the randomized sliding-window can support robust learning without losing too much key information. Additionally, the gradient aggregation of multiple snapshots can also provide stable long-term evolution information. Taking these aspects together, we summarize that the proposed WinGNN has a very competitive ability in ranking potential tail nodes by giving the head nodes in dynamic graphs.

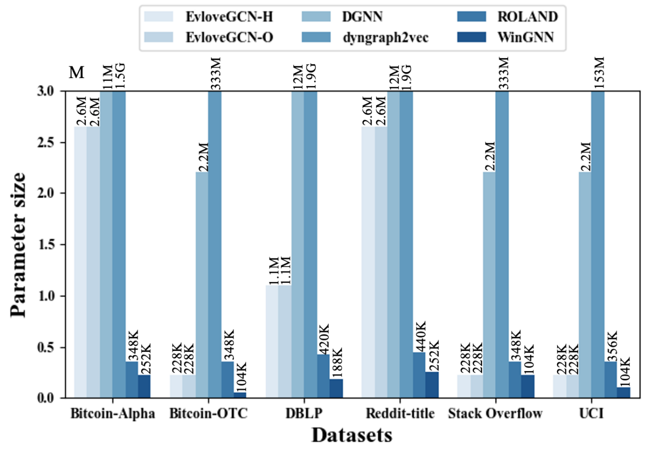
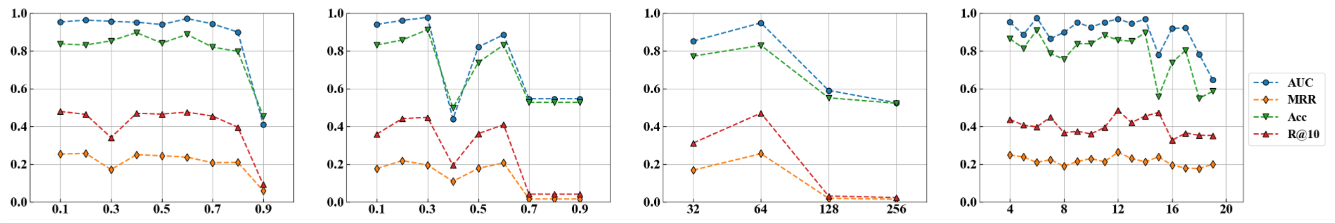


Figure 5: Model parameter size of WinGNN and other baselines. Note that the dimension size is determined by the model with the best performance.

Comparison on model parameter size. In addition to the prediction performance, we also show the advantage of WinGNN in model efficiency. In particular, we compare the parameter size of the model, which directly reflects the efficiency of encoding and prediction when the trained model is applied in real-world applications, such as online recommendation and anomaly detection. As we discussed, the parameter size of the dynamic GNN model should be an important indicator to evaluate the model's efficiency during inference. Empirically, a larger model with more parameters can have more space for encoding information, but at the same time, it increases training difficulty and time, so as to break away from the requirement of practical applications. By using our proposed temporal encoder-free strategy, we markedly reduce parameter spaces as shown in Figure 5. WinGNN significantly reduces the number of parameters in the model compared to all baselines with the same embedding dimension setting.

Table 2: Overall performance comparison on six datasets (% is omitted). The best results and second best are highlighted in bold and underline, respectively. We repeat the experiment with 10 random seeds (3 seeds for Stack Overflow) and report the average metrics with standard deviation, note that * denotes the statistical significance (p-value < 0.05) between WinGNN and the second-best baseline, and OOM. denotes the out-of-memory error when we tried to run the model on our environment.

Dataset	Metric	EvloveGCN-H	EvloveGCN-O	DGNN	dyngraph2vec	ROLAND	WinGNN	%Improv.
Bitcoin-Alpha	Accuracy	51.99±0.2546	57.44±0.4096	OOM.	OOM.	66.21±2.7566	***81.17±0.5058**	22.59%
	AUC	63.71±1.0318	68.93±0.9144	OOM.	OOM.	90.21±1.1762	**91.43±0.3259**	1.35%
	MRR	3.28±0.2845	2.52±0.1014	OOM.	OOM.	14.52±0.6506	***36.74±3.9389**	153.03%
	Recall@10	7.06±1.1900	5.27±0.5093	OOM.	OOM.	31.25±2.2782	***64.55±3.6126**	106.56%
Bitcoin-OTC	Accuracy	50.48±0.0321	50.56±1.5719	54.08±0.6755	58.29±4.5547	86.60±0.5233	**87.14±1.2408**	0.62%
	AUC	55.38±1.6617	59.82±2.5744	59.13±6.4914	62.12±10.7457	90.07±1.2998	**91.64±0.6178**	1.74%
	MRR	11.27±0.5793	11.44±0.4986	15.16±0.5773	35.39±2.5046	16.54±1.2191	***37.94±1.7019**	7.21%
	Recall@10	20.58±1.6515	26.40±2.1204	31.09±2.1594	58.29±6.7410	41.77±3.3926	***73.96±1.4569**	26.88%
DBLP	Accuracy	63.17±0.4138	65.24±0.5294	OOM.	OOM.	62.87±0.5908	***68.43±0.4135**	4.88%
	AUC	70.91±0.3823	72.64±0.4697	OOM.	OOM.	77.79±0.1689	**77.87±0.3050**	0.10%
	MRR	2.55±0.0032	2.48±0.0038	OOM.	OOM.	6.60±0.0047	*** 7.46±0.0020**	13.03%
	Recall@10	5.12±0.0310	4.84±0.0023	OOM.	OOM.	13.48±0.0132	***16.63±0.0299**	23.36%
Reddit-title	Accuracy	85.85±0.0164	77.46±1.2696	OOM.	OOM.	93.42±0.0073	***99.55±0.0009**	6.56%
	AUC	93.87±0.0054	97.17±0.2683	OOM.	OOM.	97.90±0.0001	***99.87±0.0002**	2.01%
	MRR	3.28±0.0198	1.31±0.0213	OOM.	OOM.	**35.11±0.0928**	29.91±0.0829	–
	Recall@10	5.05±0.6796	1.81±0.2453	OOM.	OOM.	**61.13±0.0970**	60.46±0.2910	–
Stack Overflow	Accuracy	OOM.	OOM.	OOM.	OOM.	62.94±18.3068	**96.91±0.1423**	53.97%
	AUC	OOM.	OOM.	OOM.	OOM.	74.56±17.9382	**99.63±0.0176**	33.62%
	MRR	OOM.	OOM.	OOM.	OOM.	27.55±21.0300	**32.51±1.5775**	18.00%
	Recall@10	OOM.	OOM.	OOM.	OOM.	41.46±32.3883	**64.71±1.7591**	56.08%
UCI	Accuracy	59.85±2.5388	49.91±1.4492	50.91±0.0510	50.88±3.1146	81.83±0.6433	***86.70±1.1867**	5.95%
	AUC	71.99±1.8252	62.05±3.8124	52.19±0.5604	54.30±1.1352	91.81±0.3052	**94.05±0.4679**	2.43%
	MRR	8.17±0.2284	10.81±0.5327	1.52±0.0016	17.84±0.4917	11.84±0.2561	***21.69±0.3383**	21.58%
	Recall@10	14.37±0.4915	16.94±0.9584	4.56±0.7313	36.22±1.6716	25.14±0.9237	***40.62±0.9364**	12.14%



(a) τ **(b) Drop rate** **(c) Embedding size** **(d) Window size**

Figure 6: Performance of WinGNN on UCI dataset with different hyper-parameter settings, including the general learning rate, drop rate, embedding size and window size.

In actual training, the GPU needs to load the model itself (including parameter matrices) and the snapshot (including adjacency and embedding matrices). The parameter matrices of the dynamic GNN are directly related to its own structural design and the dimension of embedding space, but not to the scale of the graph. The scale of the graph affects the size of adjacency matrices and embedding matrices. However, different models need to load different numbers of snapshots according to training strategies. For example, although the DGNN model only requires 12M spaces, it still meets the out-of-memory error since the temporal feature passing mechanism in its model design requires all snapshot data to be inputted into the model at the same time. WinGNN needs to load snapshots into GPU memory each time due to its sliding window design, and its video memory occupancy has a certain randomness. Therefore, we do not compare the GPU usage of each model in this paper.

Table 3: Ablation performance of WinGNN on UCI dataset, where S.D. and G.A. denote the SnapshotDrop and gradient aggregation, respectively.

Model	Accuracy	AUC	MRR	Recall@10
WinGNN	86.70±1.1867	94.05±0.4679	21.69±0.3383	40.62±0.9364
-Decay	81.44±1.2862	93.42±0.3205	23.87±0.2911	44.05±0.8395
-S.D.	85.86±1.6178	93.49±0.9019	20.70±0.4617	40.44±1.1575
-Window	58.06±2.4384	61.06±3.3366	5.01±0.7599	9.80±3.1910
-G.A.	51.55±0.0515	55.35±1.0041	1.36±0.0050	1.63±0.0186

Considering the model parameter size and dynamic link prediction performance, we show that the proposed WinGNN model has addressed the research question discussed in the Introduction section, that is, the proposed WinGNN can learn representations in dynamic graphs robustly and parameter-efficiently without using temporal encoders.

4.4 Exploration of WinGNN

After verifying the effectiveness of the proposed WinGNN, we perform the exploration study to figure out the necessity of its various components and parameter settings on the performance. Due to space limitation, we selected UCI as the standard dataset, and conduct analysis from two aspects: ablation experiment and hyper-parameter setting.

Model ablation study. As presented in Table 3, we explored the contribution of four key components in WinGNN to the final performance. In particular, "-Decay" denotes the model without the adaptive decay (i.e., $D_t = 1$ in Eq.12). "-S.D." stands for the model by removing the random drop mask (i.e., $M_t = 1$ in Eq.15). The "-Window" variant directly fixes the window size to the single evolution frame for the entire dynamic graph. It should be noted that the performance with fixed window size is further analyzed in hyper-parameter analysis, which is equivalent to WinGNN without a random window size setting. The "-G.A." variant removes the entire multi-snapshot gradient optimization strategy, which only keeps the simple GNN structure.

Based on the results, it is observed that the adaptive loss gradient aggregation has made a significant contribution to WinGNN. When we remove the whole module, the performance of WinGNN is reduced by almost half in both accuracy and AUC, and reduced by more than 90% in MRR and Recall@10. The gradient aggregation strategy without the sliding-window can provide a few performance improvements with the help of the adaptive loss gradient, from 1.36% to 5.01% in terms of the MRR metric, but still underperforms most dynamic GNN models. This observation has shown that both capturing temporal information in a term-based approach and comprehensively aggregating them are vital to the temporal-encoder-free dynamic GNN models. In addition, after we removed the adaptive decay or random SnapshotDrop, WinGNN would encounter a slight performance loss in accuracy and AUC, which also suggests the robustness brought by these two mechanisms. However, there is also a slight increment after we remove the adaptive decay of WinGNN and this phenomenon may be caused by the density distribution of the UCI dataset changing dramatically. Therefore, future dynamic GNN designs need to further consider the behavior change patterns of nodes and edges in dynamic graphs.

Hyper-parameter sensitivity analysis. In the WinGNN model, there are four major hyper-parameters which are the general learning rate τ, the drop rate of SnapshotDrop, embedding size, and window size. To verify the degree to which the model is affected by these four hyper-parameters, different settings are used to observe changes in performance, and the result is shown in Figure 6.

In Figure 6-(a), the learning rate τ determines how aggressively the model updates its existing parameters and embeddings with the incoming new loss gradient. We observe that when τ is not set to an extreme value, the performance of WinGNN is not very sensitive, and the performance of the WinGNN is relatively stable. When τ reaches 0.9, which means that the model extremely focuses on preserving the information of the current snapshot, the performance declines rapidly, which is consistent with the phenomenon that the window reduction significantly reduces the performance observed in the ablation study.

The drop rate of SnapshotDrop decides the ratio of snapshot loss to be discarded in a window. As shown in Figure 6-(b), the optimal performance is derived when the drop rate is set to 0.3. As the drop rate further increases, key dynamic information in a window can be lost, resulting in a downward trend in performance.

The effect of node embedding dimensions is also studied. As shown in Figure 6-(c), the model achieves optimal performance when the embedding dimension is selected as 64. This result also suggests that too small dimension could lead to insufficient representation for both dynamic and structural information, while too large dimension could cause over-fitting issues.

Although WinGNN uses randomized window size and step size, Figure 6-(d) shows the impact of using a fixed time window of different sizes on predicting performance. Surprisingly, when the window length is less than half of the total length (i.e., 14), the performance of WinGNN fluctuates but does not decrease significantly. However, as the window length increases further, the performance will show a decreasing trend. Thus, when there are too many snapshots in the same window compared to the entire dynamic graph, the aggregated global loss gradient could lose specificity. To summarize, we have examined the effectiveness of each component in WinGNN through the ablation study, and also shown the influence on performance caused by the hyper-parameters.

5 RELATED WORKS

5.1 Dynamic GNNs

Incorporating temporal information and graph dynamics into graph representation learning has drawn more and more attention which is motivated by the massive requirements of analyzing real-world dynamically generated graphs [36, 47]. Based on the types of temporal information to be encoded, dynamic GNN models are designed for either discrete-time or continuous-time [12]. Discrete-time dynamic graphs usually consist of discrete snapshots which reflects the periodic changes of the dynamic network, while may lose the connection information between two different snapshots [40]. Continuous-time dynamic graphs keep all edges in one graph with a timestamp label, which can store the whole dynamic network efficiently and completely, but it brings high requirements for continuous temporal information encoding [25]. However, the essence of both dynamic GNNs is to encode the temporal dynamics of the graph into updatable vector representations of nodes, which is also known as dynamic node embeddings [33], and the learned embedding can be directly used for many down-streaming tasks such as recommendation [36] and link prediction [25]. To achieve such node embedding on dynamic graphs, the most intuitive approach is using a time-aware encoder to capture graph dynamics and encode them into node embeddings [15]. These time-aware encoders could be designed to synthesize node representations at different times along with the GNN modules. For example, DGNN [21] introduces an LSTM layer after the graph convolution layer to directly pass the nodal feature in a sequential manner. EvolveGCN [24] uses LSTM and GRU to dynamically update weights of internal GNNs structure, and similar designs of synthesizing feature information could also be found in GCRN [28] and T-GCN [48]. Not only limited to recurrent neural networks (RNNs) including LSTM and GRU [24], it has also been proved that other time-aware encoders such as Transformer [26] and autoencoder [6, 31] are also useful in dynamic node embedding learning.

Modifying the mechanism of message-passing in GNNs is another feasible approach to learning graph dynamics. Instead of stacking multiple GNN layers after aggregating and propagating information through multi-hop nodes, message-passing in the dynamic graph is designed to propagate stationary distribution over dynamic path variation [34]. And such message-passing modification is also diverse, such as changing the process of propagation [4], constructing collaborative loss [37], and adjusting the scale of local structures [11]. However, capturing graph dynamics with temporal-specific encoders is a resource-consuming operation. These encoders significantly increase the parameter size of the model. Because of the changes in the temporal dimension, the number of edges in dynamic graphs is usually much larger than that in static graphs with the same scale of nodes. This problem once again exacerbates the scalability and efficiency shortcomings introduced by these temporal-specific encoders. Additionally, self-supervised contrastive learning is also a useful paradigm to learn dynamic node embedding by considering more appropriate relative distribution in the temporal dimension, such as DDGCL [33] and DySubC [11]. However, these methods are difficult to run on large dynamic graphs due to the high overhead of sampling strategies.

5.2 Meta-Learning on GNNs

Although GNNs have proved to be very powerful for a variety of graph mining tasks, it still faces the challenge of tasks with very limited labels. Meta-learning, based on the idea of model transfer, utilizes the learning experience in previous tasks to quickly adapt to a new task, which has the potential to solve the problem of label shortage faced by GNNs [20]. The purpose of the early meta-learning-based GNN was to use a graph neural network to transmit label information to unlabeled samples by message-passing through graph inference [27]. Meta-learning-based GNNs mainly focus on how to determine the representation type shared across tasks and how to design effective transfer training strategies. It has been reported that such designs improve the overall performance under limited sample size in graphs, such as node classification [49], node embedding [18], link prediction [9], and graph classification [19]. Specifically, following the general design of Model-Agnostic Meta-Learning(MAML) [3], the meta-learning-based GNN usually has an inner-loop strategy to perform task-specific parameter update, and then shares its partial parameters as common knowledge to the outer-loop for task transfer learning [8, 38].

Recent meta-learning-based studies have been devoted to model temporal factors [22, 23]. In this setting, the meta-learning-based approach regards the data on the existing temporal sequence as the preliminary task to learn the initial embedding. When the new time data arrives, the meta-learning model adjusts the model parameters quickly according to the fine-tuning strategy. Furthermore, recently, meta-learning is used to capture dynamic topological information on graphs [41]. ROLAND framework [44] adopts hierarchically update modules to update node embedding produced by the GNN layer through meta live-update. MetaDyGNN [43] utilizes a meta-learner to model hierarchical time with interval- as well as node-wise adaptions, extract knowledge for link prediction, and regard the dynamic networks as few-shot learning task. However, it should be noted that current studies update model embedding in a snapshot-by-snapshot manner, which still neglects dynamic information under different temporal scales and face the over-fitting issue caused by local optimum.

6 CONCLUSION AND FUTURE WORK

In this study, we propose a simple but effective dynamic GNN model, WinGNN, by removing parameters for temporal encoding. Particularly, WinGNN utilizes a randomized sliding-window strategy to learn the graph dynamics at different periodical terms and uses multi-snapshot gradient optimization to adaptively propagate periodical loss gradients among dynamic GNN parameters. Then the frame-wise and window-aware loss gradients are aggregated to update the GNN, thereby establishing comprehensive dynamic representation learning of nodes. Based on experiments with six public dynamic link prediction datasets, we show the proposed WinGNN achieves state-of-the-art performance by significantly reducing model parameters compared to existing baselines. Further exploration study also suggests the effectiveness and robustness of introducing random sliding window and adaptive loss gradient aggregation strategy to overcome local optimum caused by single snapshots in dynamic graphs. Future research directions are twofold. First, periodic patterns including "far-away" timestamps needs to be further extracted. Second, WinGNN's strategy also needs to be further applied in continuous-time dynamic graph modeling.

Acknowledgments. This work was supported by Technology and Innovation Major Project of the Ministry of Science and Technology of China under Grant 2020AAA0108400 and 2020AAA0108402, NSF of China for Distinguished Young Scholars (61825602), NSF of China (62276148), China Postdoctoral Science Foundation (2022M711814), Tsinghua-Bosch Joint ML Center, and a research fund from Zhipu.AI.

REFERENCES

[1] Charu C. Aggarwal and Karthik Subbian. 2014. Evolutionary Network Analysis: A Survey. *Comput. Surveys* 47, 1 (2014), 10:1–10:36.

[2] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *9th International Conference on Learning Representations, ICLR 2021*.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, Vol. 70. PMLR, 1126–1135.

[4] Dongqi Fu and Jingrui He. 2021. SDG: A Simplified and Dynamic Graph Neural Network. In *The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2021*. ACM, 2273–2277.

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010*, Vol. 9. JMLR.org, 249–256.

[6] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* 187 (2020), 104816.

[7] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. 1024–1034.

[8] Kexin Huang and Marinka Zitnik. 2020. Graph Meta Learning via Local Subgraphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

[9] Dasol Hwang, Jinyoung Park, Sunyoung Kwon, Kyung-Min Kim, Jung-Woo Ha, and Hyunwoo J. Kim. 2021. Self-supervised Auxiliary Learning for Graph Neural Networks via Meta-Learning. *CoRR* abs/2103.00771 (2021). arXiv:2103.00771 https://arxiv.org/abs/2103.00771

[10] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015,*, Vol. 37. JMLR, 448–456.

[11] Linpu Jiang, Ke-Jia Chen, and Jingqiang Chen. 2021. Self-Supervised Dynamic Graph Representation Learning via Temporal Subgraph Contrast. *CoRR* abs/2112.08733 (2021). arXiv:2112.08733 https://arxiv.org/abs/2112.08733

[12] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research* 21 (2020), 70:1–70:73.

[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015*.

[14] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017*.

[15] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019*. ACM, 1269–1278.

[16] Haoyang Li and Lei Chen. 2021. Cache-based GNN System for Dynamic Graphs. In *The 30th ACM International Conference on Information and Knowledge Management, CIKM 2021*. ACM, 937–946.

[17] Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. 2019. Predicting Path Failure In Time-Evolving Graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019*. ACM, 1279–1289.

[18] Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven C. H. Hoi. 2020. Towards Locality-Aware Meta-Learning of Tail Node Embeddings on Networks. In *The 29th ACM International Conference on Information and Knowledge Management, CIKM 2020*. ACM, 975–984.

[19] Ning Ma, Jiajun Bu, Jieyu Yang, Zhen Zhang, Chengwei Yao, Zhi Yu, Sheng Zhou, and Xifeng Yan. 2020. Adaptive-Step Graph Meta-Learner for Few-Shot Graph Classification. In *The 29th ACM International Conference on Information and Knowledge Management, CIKM 2020*. ACM, 1055–1064.

[20] Debmalya Mandal, Sourav Medya, Brian Uzzi, and Charu Aggarwal. 2021. Meta-Learning with Graph Neural Networks: Methods and Applications. *CoRR* abs/2103.00137 (2021). arXiv:2103.00137 https://arxiv.org/abs/2103.00137

[21] Franco Manessi, Alessandro Rozza, and Mario Manzo. 2020. Dynamic graph convolutional networks. *Pattern Recognition* 97 (2020).

[22] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2020. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *8th International Conference on Learning Representations, ICLR 2020*.

[23] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2021. Meta-Learning Framework with Applications to Zero-Shot Time-Series Forecasting. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 9242–9250.

[24] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*. AAAI Press, 5363–5370.

[25] Liang Qu, Huaisheng Zhu, Qiqi Duan, and Yuhui Shi. 2020. Continuous-Time Link Prediction via Temporal Dependent Graph Neural Network. In *The Web Conference 2020, WWW'20*. 3026–3032.

[26] 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 *The Thirteenth ACM International Conference on Web Search and Data Mining, WSDM 2020*. ACM, 519–527.

[27] Victor Garcia Satorras and Joan Bruna Estrach. 2018. Few-Shot Learning with Graph Neural Networks. In *6th International Conference on Learning Representations, ICLR 2018*.

[28] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured Sequence Modeling with Graph Convolutional Recurrent Networks. In *Neural Information Processing - 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I*, Vol. 11301. Springer, 362–373.

[29] Sadia Shakil, Chin-Hui Lee, and Shella Dawn Keilholz. 2016. Evaluation of sliding window correlation performance for characterizing dynamic functional connectivity and brain states. *NeuroImage* 133 (2016), 111–128.

[30] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. 2021. Foundations and Modeling of Dynamic Networks Using Dynamic Graph Neural Networks: A Survey. *IEEE Access* 9 (2021), 79143–79168.

[31] Li Sun, Zhongbao Zhang, Jiawei Zhang, Feiyang Wang, Hao Peng, Sen Su, and Philip S. Yu. 2021. Hyperbolic Variational Graph Neural Network for Modeling Dynamic Graphs. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 4375–4383.

[32] Komal K. Teru, Etienne G. Denis, and William L. Hamilton. 2020. Inductive Relation Prediction by Subgraph Reasoning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Vol. 119. PMLR, 9448–9457.

[33] Sheng Tian, Ruofan Wu, Leilei Shi, Liang Zhu, and Tao Xiong. 2021. Self-supervised Representation Learning on Dynamic Graphs. In *The 30th ACM International Conference on Information and Knowledge Management, CIKM 2021*. ACM, 1814–1823.

[34] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *7th International Conference on Learning Representations, ICLR 2019*.

[35] Xiyuan Wang and Muhan Zhang. 2022. How Powerful are Spectral Graph Neural Networks. In *International Conference on Machine Learning, ICML 2022*, Vol. 162. PMLR, 23341–23362.

[36] Yifan Wang, Yifang Qin, Fang Sun, Bo Zhang, Xuyang Hou, Ke Hu, Jia Cheng, Jun Lei, and Ming Zhang. 2022. DisenCTR: Dynamic Graph-based Disentangled Representation for Click-Through Rate Prediction. In *The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2022*. ACM, 2314–2318.

[37] Zhihao Wen and Yuan Fang. 2022. TREND: TempoRal Event and Node Dynamics for Graph Representation Learning. In *The ACM Web Conference 2022, WWW 2022*. ACM, 1159–1169.

[38] Zhihao Wen, Yuan Fang, and Zemin Liu. 2021. Meta-Inductive Node Classification across Graphs. In *The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2021*. ACM, 1219–1228.

[39] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, Vol. 97. PMLR, 6861–6871.

[40] Jiayun Wu, Tao Jia, Yansong Wang, and Li Tao. 2022. Significant Ties Graph Neural Networks for Continuous-Time Temporal Networks Modeling. *CoRR* abs/2211.06590 (2022). arXiv:2211.06590

[41] Xintao Xiang, Tiancheng Huang, and Donglin Wang. 2022. Learning to Evolve on Dynamic Graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*. AAAI Press, 13091–13092.

[42] Yu Xie, Chunyi Li, Bin Yu, Chen Zhang, and Zhouhua Tang. 2020. A Survey on Dynamic Network Embedding. *CoRR* abs/2006.08093 (2020). arXiv:2006.08093 https://arxiv.org/abs/2006.08093

[43] Cheng Yang, Chunchen Wang, Yuanfu Lu, Xumeng Gong, Chuan Shi, Wei Wang, and Xu Zhang. 2022. Few-shot Link Prediction in Dynamic Networks. In *The Fifteenth ACM International Conference on Web Search and Data Mining, WSDM 2022*. ACM, 1245–1255.

[44] Jiaxuan You, Tianyu Du, and Jure Leskovec. 2022. ROLAND: Graph Learning Framework for Dynamic Graphs. In *The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2022:*. ACM, 2358–2366.

[45] Jiaxuan You, Rex Ying, and Jure Leskovec. 2020. Design Space for Graph Neural Networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

[46] Dan Zhang, Yifan Zhu, Yuxiao Dong, Yuandong Wang, Wenzheng Feng, Evgeny Kharlamov, and Jie Tang. 2023. ApeGNN: Node-Wise Adaptive Aggregation in GNNs for Recommendation. In *Proceedings of the ACM Web Conference 2023, WWW 2023*. 759–769.
[47] Zhaoli Zhang, Zhifei Li, Hai Liu, and Neal N. Xiong. 2022. Multi-Scale Dynamic Convolutional Network for Knowledge Graph Embedding. *IEEE Transactions on Knowledge and Data Engineering* 34, 5 (2022), 2335–2347.
[48] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2020. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transpartation Systems* 21, 9 (2020), 3848–3858.
[49] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. 2019. Meta-GNN: On Few-shot Node Classification in Graph Meta-learning. In *The 28th ACM International Conference on Information and Knowledge Management, CIKM 2019*. ACM, 2357–2360.
[50] Yifan Zhu, Xuesong Li, Yufei Qiao, Ruihong Shang, Gen Shi, Yingying Shang, and Hua Guo. 2021. Widespread plasticity of cognition-related brain networks in single-sided deafness revealed by randomized window-based dynamic functional connectivity. *Medical Image Analysis* 73 (2021), 102163.

Algorithm 1: WinGNN training algorithm

```
Input  : Dynamic graph $G$, Node feature $X$
Output : Optimized model parameters $\Theta$
1  Initialize $\Theta$ ;
2  while $\Theta$ does not converge do
3      Randomly generate $m$ sliding-windows on $G$ with different
         window size $l^w$ and step size $l^s$ ;
4      for $i : 1 \to m$ do
5          for $t : idx(i) + 1 \to idx(i) + l^w$ do
               // GNN propagation with $\Theta$
6              Calculate the prediction loss: $\mathcal{L}_{G_t}^{w_i} \leftarrow$ Eq. (5)-(7) ;
               // Frame-wise parameter update
7              Obtain frame-wise gradient: $\nabla \mathcal{L}_{G_t}^{w_i} \leftarrow$ Eq. (8) ;
8              Update on the next snapshot: $\theta_{t+1}^{w_i} \leftarrow$ Eq. (9) ;
               // Multi-snapshot gradient collection
                  within sliding-window
9              Obtain window-aware gradient: $\nabla \hat{\mathcal{L}}_{G_t}^{w_i} \leftarrow$ Eq. (10) ;
           // Multi-snapshot gradient aggregation
10         Aggregate window-aware gradient: $\nabla \hat{\mathcal{L}}^{w_i} \leftarrow$ Eq.(15) ;
11         Update WinGNN: $\Theta = \theta_{t+1}^{w_{i+1}} \leftarrow$ Eq. (16) ;
12 Return $\Theta$.
```

A IMPLEMENTATION DETAILS

A.1 Training algorithm

The training process of WinGNN is summarized as Algorithm 1, where $idx(i)$ is a function that returns the index of the i-th window.

Complexity analysis. The time complexity of WinGNN is at the same level as existing dynamic GNN models. Suppose $|U|$ and $|\mathcal{E}|$ are the total number of nodes and edges in the dynamic graph, d denotes the dimension of the input embedding of nodes, F denotes the dimension of node's embedding, L denotes the layers of GNN, $|S|$ is the average size of the sliding window, and $|T|$ is the total amount of snapshots. Here we only consider the worst situation. During the training process, the time complexity is the complexity of GCN ($O(\mathcal{E}|FL)$) multiplied by the length of windows as well as the number of windows, i.e., $O(|\mathcal{E}|FL|S||T|)$. In contrast, EvolveGCN uses RNN to learn the evolution of GCN parameters, thus the complexity is $O(|T||\mathcal{E}|FL + |T|(dF + d^2 + d))$. In terms of ROLAND, the time complexity is the $O(|T||\mathcal{E}|FL + |T|LP)$, where P depends on the selection of the embedding update module. Additionally, the running time of WinGNN is observed to be markedly smaller than EvolveGCN and slightly larger than ROLAND. For example, the running time of WinGNN on the UCI dataset is 4m26s, which is between the EvolveGCN (15m37s) and ROLAND (3m3s). However, it should be noted that running time is a volatile number that can be affected by a variety of environmental factors.

A.2 Dataset Description

In this paper, we conduct the experiment on six typical public datasets to perform a comparison between WinGNN and other baselines. All of these datasets can be obtained from the SNAP website (https://snap.stanford.edu/) A brief description of these six datasets is as follows:

- **Bitcoin-Alpha** and **Bitcoin-OTC**: These datasets describe directed graphs on anonymous bitcoin transaction platform OTC and Alpha, where nodes represent user accounts and edges are the ratings made between users.
- **DBLP**: The DBLP dataset provides a directed graph to describe an academic collaboration network, where the nodes are authors and edges represent co-authored papers.
- **Reddit-title**: This dataset denotes a directed graph on the Reddit platform where the node represents the subreddit and the edges are set according to the items that add hyperlinks linking two subreddits. The hyperlink in Reddit-title is presented in the title of the post.
- **Stack Overflow**: This dataset describes the interactions on Stack Overflow platform. In this dataset, the nodes represent users and directed edges denote the answer activity from one user to another.
- **UCI**: This dataset presents a directed graph describing message flow on an online social network at the University of California, Irvine. In the UCI dataset, nodes denote users and edges are messages sent between users.

In particular, since different dynamic graphs have different density distributions on the list snapshots, we show the density variation of the datasets used in the experiment through Figure 7.

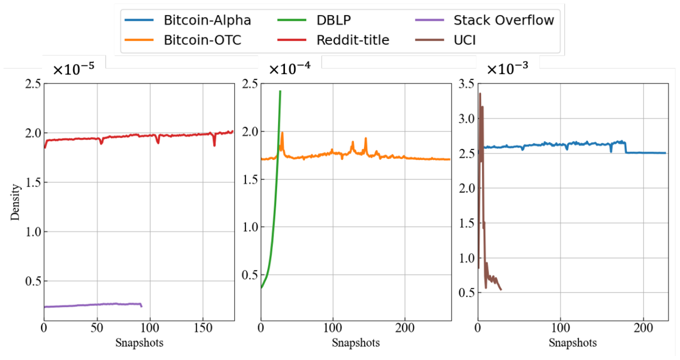


Figure 7: The density of each snapshot on the six datasets.

A.3 Description of Baselines

We performn experiments by comparing the performance with the following dynamic GNN baselines,

- **EvolveGCN** [24]: uses an RNN to update internal GNN parameters between snapshots, thereby endowing GNN with the ability to encode partial temporal information. In this experiment, we select both versions with different temporal encoders (i.e. LSTM vs. GRU) and denote them as EvloveGCN-O and EvloveGCN-H.
- **DGNN** [21]: utilizes a stacked encoder to capture the dynamics of nodes through LSTM over the encoded representation by GNNs.
- **dyngraph2vec** [6]: learns the temporal transitions in a dynamic graph by using an auto-encoder architecture composed of dense and recurrent layers.

Table 4: Comparison of recall@1 and recall@3 on six datasets (% is omitted). The best results and second best are highlighted in bold and underline, respectively. We repeat the experiment with 10 random seeds and report the average metrics with standard deviation, noting that OOM. denotes the out-of-memory error when we tried to run the model on our environment.

Dataset	Metric	EvloveGCN-H	EvloveGCN-O	DGNN	dyngraph2vec	ROLAND	WinGNN
Bitcoin-Alpha	Recall@1	1.20±0.0985	0.79±0.0326	OOM.	OOM.	6.89±0.4815	**23.26±4.2389**
	Recall@3	3.13±0.4014	2.11±0.1310	OOM.	OOM.	14.78±1.1743	**43.21±6.4565**
Bitcoin-OTC	Recall@1	6.39±0.4020	4.88±0.3454	7.32±0.1588	**23.55±1.9817**	10.67±0.9296	21.70±0.2256
	Recall@3	14.65±0.8830	15.07±1.1589	20.72±1.1342	44.17±4.1239	23.91±2.0352	**49.96±3.1181**
DBLP	Recall@1	0.35±0.0001	0.56±0.0005	OOM.	OOM.	**2.60±0.0002**	**2.60±0.0003**
	Recall@3	1.23±0.0001	1.59±0.0004	OOM.	OOM.	6.02±0.0006	**6.68±0.0012**
Reddit-title	Recall@1	0.01±0.0000	0.00±0.0000	OOM.	OOM.	**22.36±0.0960**	16.13±0.0016
	Recall@3	0.30±0.0006	0.02±0.0000	OOM.	OOM.	**40.34±0.1271**	33.11±0.1689
Stack Overflow	Recall@1	OOM.	OOM.	OOM.	OOM.	19.01±15.1597	**18.42±1.3139**
	Recall@3	OOM.	OOM.	OOM.	OOM.	33.31±26.4391	**36.05±2.1176**
UCI	Recall@1	3.82±0.2325	6.90±0.4602	0.28±0.0002	10.05±0.3551	5.21±1.1867	**12.49±0.3434**
	Recall@3	6.90±0.4603	10.57±0.6043	0.95±0.0005	18.40±0.6450	10.60±0.4220	**22.20±0.5227**

- **ROLAND** [44]: stacks an update module (such as MLP and GRU) with live update mechanism on the traditional GNN layer which is shown in Figure 2-(B), thereby achieving the SOTA performance.

A.4 Evaluation Metrics

We testify the performance of WinGNN and other baselines by using four metrics with both classification and ranking. A brief description of these metrics is as follows:

- **Accuracy** indicates the basic classification ability of the model by predicting whether there is a link (i.e. edge) between the two given nodes.
$$Accuracy = \frac{|T(\mathcal{E}_{\pm}') \cap R(\mathcal{E}_{\pm}')|}{|T(\mathcal{E}_{\pm}')|}, \tag{17}$$
where $T(\mathcal{E}_{\pm}')$ denotes all edges (including both positive and negative) in the test set and $R(\mathcal{E}_{\pm}')$ is the predicted positive and negative edges by given the head and tail nodes.
- **Area Under Curve (AUC)** uses the area of receiver operating characteristic curve (ROC) to evaluate the classification quality.
- **Mean Reciprocal Rank (MRR)** is a measure of predicting list which counts reciprocal ranks of the correctly-predicted node in the whole list.
$$MRR@K = \frac{1}{N} \sum_{i \le K} \frac{1}{Rank(i)}, \tag{18}$$
where N is the node's number in the test set, and $Rank(i)$ indicates the position of the correct tail node in the list ordered by the predicting logits.
- **Recall** measures the extent to which top-K link prediction accurately identifies true nodes.
$$Recall@K = \frac{1}{|\mathcal{N}|} \sum_{u_i \in \mathcal{N}} \frac{|T(u_i)| \cap |R(u_i)|}{|T(u_i)|}, s.t. |R(u_i)| = K, \tag{19}$$
where $T(u_i)$ and $R(u_i)$ are the labeled and predicted tail node set by given the head node u_i, respectively.

A.5 Experiment Setup

Running Environment. We perform our comparsions on a Ubuntu 18.04.2 LTS server with AMD EPYC 7642 48-Core Processor, 1008 GB RAM, and an NVIDIA A100 Tensor Core GPU. WinGNN is implemented with Python 3.9.7. WinGNN by using the PyTorch 1.9.1 and DGL 0.9.1 framework. We also utilize the modules for data processing as well as evaluation from GraphGym [45].

Hyper-parameter Settings. There are four major hyper-parameters in the WinGNN model, which are: the selection of τ, snapshot drop rate, embedding size, and sliding-window size. We derive the optimal performance by searching different parameter setting as follows: $\tau \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$; snapshot drop rate $\in \{0.1, 0.3, 0.5, 0.7, 0.9\}$; embedding size $\in \{32, 64, 128, 256\}$; window size $\in \{l^w \in \mathbb{N} | 4 \le l^w \le 19\}$. Additionally, the Adam optimizer [13] is used as the optimizer for gradient descent and the Xavier [5] is utilized as the basic initializer for each parameter matrix.

B ADDITIONAL RESULTS

We also show the comparison in terms of recall@1 and recall@3 between WinGNN and other baselines, and the results are presented in Table 4.