DyGSSM: Multi-view Dynamic Graph Embeddings with SSM Gradient Update

Bizhan Alipour Pijani
University of North Texas
pijanialipourpijani@my.unt.edu

Serdar Bozdag
University of North Texas
serdar.bozdag@unt.edu

Abstract

Dynamic graphs whose topology and nodes evolve over time are ubiquitous in multiple real world domains such as social networks, finance, and healthcare. Traditional graph learning methods fail to capture structural changes and temporal patterns in dynamic graphs. Recent advances in dynamic graph representation learning, such as meta-learning-based approaches, have addressed some of these challenges. However, existing methods still face three key limitations. First, most approaches capture either local or global structures of the graphs, neglecting to model both simultaneously. Second, meta-learning models often depend on user-specific window size, which must be carefully tuned for each dataset. A short window size may miss trends, and a long window size may blur recent updates. Third, most methods work on only discrete-time or continuous-time dynamic graphs, resulting in suboptimal performance across different temporal settings. To address these limitations in dynamic graph representation learning, we propose a novel method called DyGSSM (Multi-view Dynamic Graph Embeddings with SSM Gradient Update). We extract local and global features at each snapshot and fuse them using a lightweight attention mechanism for link prediction. To capture long-term dependencies when updating model parameters, we incorporate HiPPO (High-order Polynomial Projection Operators) algorithm, which has gained attention for its ability to efficiently optimize and preserve sequence history in State Space Models (SSMs). DyGSSM is designed to handle both discrete-time and continuous-time dynamic graphs. Parameter comparisons show that DyGSSM requires substantially fewer parameters than the other methods. Extensive experiments on 12 public datasets demonstrate that DyGSSM outperforms baselines and State-Of-The-Art (SOTA) methods in 32 out of 36 evaluation metrics. The source code and datasets are available at https://github.com/bozdaglab/DyGSSM.

1 Introduction

Dynamic graphs are ubiquitous as many real-world interactions and relationships are dynamic in nature, such as social networks [1], transportation networks [2], transaction networks [3], and trade networks [4]. Unlike static graphs, dynamic graphs evolve over time, with changes occurring in their topology or edge/node attributes. Message passing-based graph representation learning models [5–9] have achieved significant success in graph analysis tasks. These models are effective in capturing local structural information due to the inherent locality of the message-passing mechanism. These models have recently been extended to dynamic graphs for representation learning [10–14]. For example, in [13, 15, 16] "snapshot" of the dynamic graph at each time point is processed using a graph representation learning approach and sequence encoders such as Long Short Term Memory (LSTM) [17] or Transformers [18] are utilized to capture the temporal evolution of the graph over time. Although these approaches have shown promising results, they have low expressive power, as an independent GCN is trained for each snapshot of the graph. As a result, these models may fail to extract historical structural information. To address this, researchers have integrated the sequence

Pijani et al., DyGSSM: Multi-view Dynamic Graph Embeddings with SSM Gradient Update. *Proceedings of the Fourth Learning on Graphs Conference (LoG 2025)*, PMLR 269, Hybrid Event, December 10–12, 2025.

encoder into the GCN layer to update the parameters of the GCN model [17, 18] over time. However, these methods have another limitation. In dynamic graphs, changes occurring in multiple hops away from a source node could still influence the source node in subsequent snapshots. However, these methods fail to extract signal from such distant nodes, thereby reducing their expressive power.

Meta-learning has emerged as an effective approach for modeling temporal dynamics. In meta-learning, which is based on the idea of model transfer, the model parameters are updated based on previous time steps and then passed to the next time steps. In this setting, ROLAND [10] updated node embeddings generated by the GNN layer utilizing adjacent time snapshots. However, this approach only aggregates two adjacent snapshots and neglects temporal information with long-term dependencies. WinGNN [11] introduced a sliding window approach to update the model parameters. WinGNN is mainly designed for discrete-time snapshots and may not handle irregularly timed events. It relies on several hyperparameters, such as window size, beta value, and meta-learning rate, which require careful tuning for each dataset. Additionally, if the window size is too small, it may miss trends, and if too large, it may blur recent updates. Moreover, these methods are GNN-based and typically focus on local neighbors (local view), while ignoring the global structure of the graph (global view). Both views provide complementary information: local views capture fine details of immediate interactions, while global views capture long-range changes that may affect future states. Long-distance information is especially important in time series data, as dynamic graphs evolve over time and interactions can occur at any step [19, 20]. Changes in distant hops may also influence the source node in later snapshots. Therefore, it is important to extract both local and global features, which together can be regarded as a multi-view representation of the same snapshots.

Recently, state space models (SSMs) have become a popular and powerful tool for sequence modeling. Some SSM-based methods have been proposed for dynamic graphs [21, 22]. For example, GraphSSM [23] leverages SSMs to capture continuous-time dynamics in dynamic graphs for node classification tasks. DyGMamba [21] uses two types of SSM, a node- and time-level SSM. DyGMamba encodes one-hop temporal neighbors of nodes; as a result, it may miss higher-order structural dependencies. Additionally, the node-level SSM still sequentially processes interaction histories for each node. In very dense graphs or when histories are extremely long, this sequential processing can become computationally expensive and slow. As shown in the results section, dynamic SSM-based models often require a large number of parameters, which makes them less practical for large datasets.

To address these challenges, we introduce Multi-view Dynamic Graph Embeddings with State Space Model Gradient Update (DyGSSM). While traditional SSMs model time-series data directly, DyGSSM leverages SSM to update model parameters. To initialize the SSM state, we utilized High-order Polynomial Projection Operators (HiPPO) [24, 25]. To make this process computationally tractable, we introduce a compressed parameter-space representation: instead of flattening every scalar weight into the SSM state, we treat each parameter tensor as a single entity, maintaining one HiPPO state per tensor. This drastically reduces memory and computational overhead and reduces the number of hyperparameters (such as window size). The SSM in DyGSSM considers the loss of each snapshot during parameter updates, which facilitates smoother updates and introduces a mechanism for *forgetting* less relevant information from the past time while *remembering* critical patterns from earlier ones. This design enables DyGSSM to encode both discrete-time (snapshot-based) and continuous-time dynamic graphs. Additionally, DyGSSM introduces a lightweight attention-fusion mechanism that departs from conventional multi-head attention. We summarize our main contributions as follows:

- To the best of our knowledge, we are the first to integrate SSM into the meta-learning strategy to update model parameters. We introduce an SSM-based method to effectively capture long-term dependencies when updating model parameters. This approach avoids the need for numerous hyperparameters, which can increase the model's sensitivity to specific datasets.
- We introduce a compressed HiPPO formulation that maintains one SSM state per parameter tensor rather than per-weight, which significantly reduces computational and memory costs.
- DyGSSM is designed to handle both discrete-time (snapshot-based) and continuous-time dynamic graphs, extending its applicability to a broader range of temporal graph scenarios.
- We introduce a lightweight, and scalable attention mechanism that fuses local and global embeddings efficiently.
- Extensive experiments on multiple datasets demonstrate the superiority of DyGSSM over State-Of-The-Art (SOTA) models, while having the lowest number of parameters.

2 Related Work

Dynamic graph representation models can broadly be categorized into the following three groups.

2.1 Sequence-Based Models

Sequence-based models follow the message passing and temporal encoder to capture time dependencies [12]. Researchers have utilized GCN with RNN variants to capture time dependencies. For example, CD-GCN [26] is a combination of GCN and LSTM. They apply GCN to obtain the embeddings of each snapshot and pass the embeddings to LSTM for time sequence dependencies. EvolveGCN [17] and GC-GCN-N [27] integrate GCN and GRU for tasks such as link prediction, edge classification, node classification, and landslide displacement forecasting. GC-GCN-N captures spatial dependencies among monitoring stations through a weighted adjacency matrix and temporal patterns from time-series data using GRU. PoGeVon [19] introduces an encoder-decoder architecture for dynamic graph representation. They utilize a novel node position embedding derived from the random walk with restart (RWR) approach. In addition, they use the concept of the sliding window with a Lagrange multiplier to control the amount of information that can be transmitted through the latent representation. They use a 2-layer GRU to capture the dynamic information in networked time series. These models have two main limitations. First, they do not share parameters across time steps; instead, each time step trains an independent GCN, which restricts the model's ability to leverage historical structural information. Second, they require a large number of parameters because they rely on sequence-based models (i.e., GRU and LSTM) to capture the graph's temporal evolution.

2.2 Meta-Learner-Based Models

Meta-learning is based on the idea of transfer learning, where previous experience is used to quickly adapt to a new task. In dynamic graphs, meta-learning–based models extend static GNNs by learning model parameter initializations for the next time steps. ROLAND [10] extends static graphs to dynamic ones with minimal extra computational cost. They use a two-layer GNN, where each layer updates its parameters and passes them to the adjacent snapshot. WinGNN [11] proposes a framework that combines GNN with a meta-learning strategy and a novel random gradient aggregation mechanism. Instead of relying on temporal encoders, WinGNN models graph dynamics by introducing a randomized sliding-window strategy that computes loss on each snapshot and propagates updated model parameters to the next snapshot. They perform backpropagation only at the end of the window. MetaDyGNN [28] leverages a meta-learning strategy for few-shot link prediction in dynamic graphs. They introduce time interval-wise and node-wise adaptations to encompass time dependencies and node dependency features, and update the global parameters. These models suffer from one main limitation. The meta-learning parameters, such as the meta-learning rate and window size, must be carefully tuned for each dataset, which adds extra complexity. For example, short window size (or a ROLAND-based parameter update) may fail to capture long-term trends, while a long window size may obscure recent updates.

2.3 SSM- and Transformers-based methods

Many researchers have used transformers instead of LSTM or GRU to capture the temporal evolution of dynamic graphs [29]. For example, Dysat [18] employs self-attention in two different aspects. First, attending to structural neighborhoods at each time point. Second, attending to previous historical representations to conduct link prediction. Graph Transformers (GT) have gained popularity in the field of graph representation [30–32]. For example, TransformerG2G [33] utilizes transformer for learning temporal graphs. They use only transformer encoder to learn intermediate node representations from all the previous snapshots up to the current snapshot. They use two projection heads (linear mapping and non-linear mapping) to generate low dimensional latent embedding at different snapshots. DTFormer [34] collects all the first-hop neighbors of source and destination nodes. Then, it maps these neighbor features into a sequence to be processed by transformer. Despite the effectiveness of transformer on graph-structured data, it suffers from having a quadratic computational cost and lack of inductive biases on graph structures.

Recent successes of SSM-based models (such as Mamba) in computer vision and natural language processing tasks have motivated researchers to apply SSM-based models to graphs. For example, DyGMamba [21] introduces two levels of SSM: a node-level SSM to encode node interactions and

a time-level SSM to exploit temporal patterns. DG-Mamba [22] treats a dynamic graph as a self-contained system, using an SSM to capture global intrinsic dynamics. It discretizes the system state according to cross-snapshot graph adjacency, enabling the model to capture long-range dependencies through a selective snapshot scanning strategy. Dyg-mamba [35] proposes a new continuous SSM for dynamic graph learning. They consider irregular time spans as control signals for SSM to have robust and generalizable model. Although these methods achieve good performance, their high computational and memory costs make it difficult to scale them to large dynamic graphs. Moreover, none of the SSM-based models integrate SSM and meta-learning to address the limitations of meta-learning while enabling the model to distinguish which information to forget and which information to remember from the past.

3 Preliminaries

In this section, we introduce the notion of a discrete and continuous-time dynamic graph and other important components that are adopted in DyGSSM.

3.1 Problem Formulation

Let $\mathcal{V}$ be a set of nodes and $\mathcal{E}$ be the set of edges that connect the nodes in $\mathcal{V}$. A graph consists of three components $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{X} \in \mathbb{R}^{n \times m}$ is a node feature matrix, $n = |\mathcal{V}|$ and m is the dimension size of the feature. For a graph $\mathcal{G}$, we can create an adjacency matrix $A \in \mathbb{R}^{n \times n}$, that represents local neighbors of each node as follow:

$$\mathcal{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

To study a discrete-time dynamic graph, we let $G = \{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ be a sequence of graphs for discrete snapshots $t = 1, \ldots, T$. Here, each $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{X}_t)$ represents a snapshot of the dynamic graph with adjacency matrix $\mathcal{A}_t$ at time t. The local neighbors of a node i at time t denoted as $\mathcal{N}_{t,i}^{local} = \{v_j \mid (v_i, v_j) \in \mathcal{E}_t\}$ and the global neighbors of a node i at time t denoted as $\mathcal{N}_{t,i}^{global} = \{v_j \mid v_j \in \mathcal{RW}_t(v_i)\}$ where $\mathcal{RW}_t(v_i)$ denotes the set of nodes visited by random walk ($\mathcal{RW}$) starting from v_i in snapshot t. In contrast, a continuous-time dynamic graph models interactions as an event stream. We represent the graph as a sequence of non-decreasing chronological interactions

$$G = \{(u_1, v_1, t_1), (u_2, v_2, t_2), \ldots\},$$

with $0 \leq t_1 \leq t_2 \leq \ldots$, where $u_i, v_i \in \mathcal{V}$ denote the source and destination nodes of the i-th interaction at timestamp t. Each node $u \in \mathcal{V}$ may be associated with a feature vector $x_u \in \mathbb{R}^{d_N}$, and each interaction (u, v, t) may carry an edge feature $e_{u,v}^t \in \mathbb{R}^{d_E}$, where d_N and d_E denote the dimensions of the node feature and link feature. To evaluate DyGSSM, we consider a link prediction task in discrete- and continuous-time dynamic graphs. The model takes the local ($\mathcal{X}_t^{local}$) and global ($\mathcal{X}_t^{global}$) node embeddings of nodes v_i and v_j in discrete/continuous time, and outputs the probability of a connection between them in the next time frame or event.

3.2 State Space Model

A SSM frames a discrete-time system by a linear mapping from a discrete input u_t at time t to a discrete output y_t through a state variable s_t and three matrices, namely K, B, and C as follows:

$$s_t = Ks_{t-1} + Bu_t \tag{2}$$

$$y_t = Cs_t \tag{3}$$

The structure of the matrix K is important when building an SSM, as this matrix determines which part of the previous state can be passed to the current state. In continuous time, the hidden state ($s(t)$) evolves as follows:

$$\frac{ds(t)}{dt} = Ks(t) + Bu(t) \tag{4}$$

$$y(t) = Cs(t) \tag{5}$$

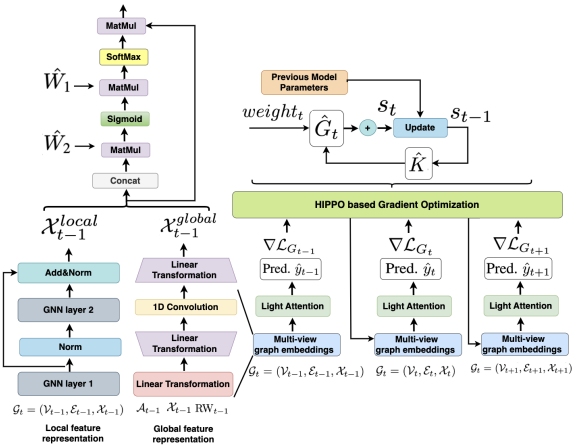

Figure 1: DyGSSM architecture.

where $u(t)$ is a continuous input signal and $y(t)$ is the corresponding output. To use the SSM for meta-learning-based parameter updates without relying on a window size, we start from Equation 2 and modify it slightly to obtain

$$s_t = \hat{K} s_{t-1} + \hat{G}_t \cdot weight_t \tag{6}$$

where s_{t-1} is the state vector at time $t-1$, initialized to zero, $\hat{G}_t$ is the gradient vector of the model parameters at time t, $weight_t$ is the reciprocal of the loss (see Section 4.3), and $\hat{K}$ is the projection matrix. Using HiPPO to initialize the matrix K was shown to perform better than initializing it as a random matrix. The HiPPO matrix is designed to generate a hidden state that can memorize the past inputs with no tunable hyperparameters. We initialized $\hat{K}$ using the HiPPO algorithm [24, 25] as follow:

$$\hat{K}_{i,j} = \begin{cases} (-1)^{i-j}(2i+1) & \text{if i > j} \\ 2 & \text{if i= j} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

For entries where the row index is greater than the column index ($i > j$), the equation above fills the values along the lower diagonal. $(2i+1)$ acts as linear function of the row index i and takes a positive or negative sign depending on whether $i-j$ in $(-1)^{i-j}$ is even or odd. If $i-j$ is even, the output is positive; otherwise, it is negative. On the diagonal ($i = j$), this matrix has a value of 2, and above the diagonal ($i < j$), the values are set to 0. The Equation 6 in continuous time becomes as follows:

$$s(t) = \hat{K} s(t) + \hat{G}(t) \cdot weight(t) \tag{8}$$

4 Method

In this section, we describe the DyGSSM architecture. We start by explaining how we compute local and global node embeddings. Next, we show how these two types of embeddings are combined. Finally, we discuss how the model optimizes its parameters using the HiPPO method. Fig. 1 shows DyGSSM architecture and Fig. 2 (see Appendix A.2) represents the parameter update mechanisms in ROLAND, WinGNN, and DyGSSM. Note that we only show local and global representations in the discrete-time setting here, but the same process can be applied to the continuous-time case as well.

4.1 Node Embeddings

Here, we discuss local and global node embeddings.

4.1.1 Local Embeddings

Graph representation techniques update the embedding of each node u in $\mathcal{G}_t$ by performing message passing between neighbor nodes as follows:

$$h_{t,u}^{(l)} = UPDATE(h_{t,u}^{(l-1)}, AGG(h_{t,v}^{(l-1)}, \forall v \in \mathcal{N}_{t,u}^{local})), \tag{9}$$

where $h_{t,u}^{(l)}$, the embedding of node u at layer l, is computed by aggregating information from all its local neighbors at time point t, denoted by $\mathcal{N}_{t,u}^{local}$. This process can be expressed in matrix form as $H_t^{(l)} = \sigma\left(\hat{A}_t H_t^{(l-1)} W_t^{(l)}\right)$ where $H_t^{(l)} \in \mathbb{R}^{n \times d}$ represents the embeddings of all nodes in $\mathcal{G}_t$ at layer l, $\hat{A}_t = D_t^{-\frac{1}{2}} A_t D_t^{-\frac{1}{2}}$ is the normalized adjacency matrix with A_t being the adjacency matrix of $\mathcal{G}_t$ and D_t being the corresponding diagonal degree matrix. Here, $W_t^{(l)}$ is the learnable weight matrix of the l-th layer at time t, and σ denotes a non-linear activation function. We used two-layer message passing to compute local node embeddings ($\mathcal{X}_t^{local}$).

4.1.2 Global Embeddings

Given a snapshot of a graph at time t, $\mathcal{G}_t$, to compute global embeddings for nodes, we computed a biased $\mathcal{RW}$-node embedding as follows: For each node in $\mathcal{G}_t$, we customized the $\mathcal{RW}$ to explore far-away nodes from the source node. To make sure that we are not selecting an arbitrary global neighbor for each node, we ran the biased $\mathcal{RW}$ for each node 50 times with a path length of 5. This results in a list of 250 nodes for each source node. We selected the five most frequent nodes in the sequence for each source node.

To increase the efficiency of the training phase, we precomputed the global neighbors of each node before the training phase. However, $\mathcal{RW}$ can still be costly on large graphs as the biased $\mathcal{RW}$ must be performed for each node. Alternatively, $\mathcal{RW}$ can be recalculated for the entire graph only when new data arrives, which can increase the overall inference time. To mitigate these cost-prohibitive scenarios, we introduced a caching mechanism that stores node edges from prior snapshots. At each snapshot, we identify newly introduced nodes or those whose topology has changed, and recompute the neighbors for only those nodes. For the remaining snapshots, we reuse the cached distributions to avoid redundant computation. This would allow the model to reuse previously computed information and update embeddings incrementally, rather than recomputing them entirely, which reduces $\mathcal{RW}$ computational cost (see Fig. 3 in Appendix D). After computing the $\mathcal{RW}$ for the source node, we obtain a sequence of five nodes, including the source node i as follows, $\mathcal{RW}_t^i = [v_t^i, v_t^1, v_t^2, v_t^3, v_t^4]$. The embeddings of the generated sequence are passed through a linear transformation:

$$z_t^i = W_1 \, \mathcal{RW}_t^i + b_1,$$

where W_1 and b_1 are learnable parameters that expand the embedding dimension. Next, a 1D convolution is applied to generate embeddings for each source node:

$$e_t^i = Conv1D(z_t^i),$$

where $Conv1D(\cdot)$ denotes the 1D convolution operation. Finally, another linear transformation reduces the dimension back to the original size:

$$x_t^{global} = W_2 e_t^i + b_2,$$

with learnable parameters W_2 and b_2. The final global node embeddings for all nodes in $\mathcal{G}_t$ are denoted as $\mathcal{X}_t^{global}$. Note that this global node embedding approach is highly parameter-efficient compared to GRU, LSTM, or transformer encoders.

4.2 Integration of Local and Global Node Embeddings

To fuse the local and global embeddings, we first concatenate $\mathcal{X}_t^{local}$ and $\mathcal{X}_t^{global}$ as follows:

$$h_t^{concat} = concat(\mathcal{X}_t^{local}, \mathcal{X}_t^{global}) \tag{10}$$

Next, we compute attention weights for each embedding as follows:

$$a_t = \text{softmax}\left(\hat{W}_1 \cdot \sigma(\hat{W}_2 \cdot h_t^{concat})\right) \tag{11}$$

where $\hat{W}_1$ and $\hat{W}_2$ are two learnable attention matrices with shapes $2 \times d \times d_a$ and $2 \times d_a \times 2$, respectively. Here, d_a denotes the size of attention weights computed for each embedding, and the 2 corresponds to the number of embeddings (local and global). a_t acts as a gate that controls the amount of information that $\mathcal{X}_t^{local}$ and $\mathcal{X}_t^{global}$ can transmit to the final representation of the graph at time t. This single lightweight gating operation has no head count or depth parameters. As a result, the performance is not sensitive to some parameters (e.g., the number of attention layers). The final representation is given as follows:

$$h_t^{fused} = a_t \cdot h_t^{concat} \tag{12}$$

4.3 Gradient Optimization using HiPPO

Consistent with existing research, we use the cross-entropy loss as our loss function:

$$\mathcal{L}_t = -\frac{1}{M} \sum_{(u,v)\in\mathcal{E}_t}^{M} y_{u,v}^t \cdot log(\hat{y}_{u,v}^t) + (1 - y_{u,v}^t) \cdot log(1 - \hat{y}_{u,v}^t) \tag{13}$$

where $\hat{y}_{u,v}^t$ is calculated as follow

$$\hat{y}_{u,v}^t = MLP(concat(h_{u,t}^{fused}, h_{v,t}^{fused}))$$

The main objective of this step is to calculate the gradient of the current snapshot t with respect to $\mathcal{L}_t$, efficiently optimize model parameters, and propagate it to the next snapshot. To do that, we first calculate the reciprocal of the loss, acting as a simple dynamic weighting mechanism to adjust the influence of each snapshot during parameter updates, as presented below:

$$weight_t = \frac{1}{\mathcal{L}_t + \epsilon} \tag{14}$$

This dynamic weighting helps the model prioritize parameter updates from snapshots where it performs well (i.e., when the loss $\mathcal{L}_t$ is small), while reducing the influence of updates from snapshots with poor performance. The small constant ϵ prevents division by zero and ensures numerical stability. We followed Equation 6 to update the SSM state, and performed model parameter updates as follows

$$loss\_scale = \min(loss, max\_loss\_scale) \tag{15}$$

$$gate = tanh(\Theta_t \times s_t) \tag{16}$$

$$scaled\_gate = clamp(gate, -max\_gate, max\_gate) \tag{17}$$

$$\Theta_{t+1} \leftarrow \Theta_t - loss\_scale \times scaled\_gate \tag{18}$$

In the update equation, Θ_t denotes the model parameter in the current snapshot, and Θ_{t+1} is the updated model parameter for the snapshot $t + 1$. The loss value capped by a constant value, max_loss_scale, to produce the loss scaling factor $loss\_scale$, which prevents excessively large gradients or NaN values during training. The raw gradients (Θ_{t+1}) are then elementwise multiplied by the SSM state (s_t), passed through a hyperbolic tangent function to bound their range, and then clamped to the interval ($[-max\_gate, max\_gate]$), producing $scaled\_gate$ value. Finally, the update step multiplies this loss scaling factor ($loss\_scale$) by the scaled gate ($scaled\_gate$) and subtracts the result from the current parameters (Θ_t), ensuring stable and bounded parameter updates. In our implementation, we set $max\_gate = 1.0$ and $max\_loss\_scale = 0.1$. Both constants are non-learnable and were not subject to hyperparameter tuning. These two factors aim to mitigate instability caused by spiky or large loss values.

In continuous dynamic graphs, each interaction (edge event) is processed sequentially according to its timestamp rather than aggregated into fixed snapshots. When a new event (u, v, t) arrives, DyGSSM uses the cached $\mathcal{RW}$ representation and runs $\mathcal{RW}$ only for the affected nodes, ensuring that computation scales linearly with the number of active nodes. The HiPPO-based SSM state maintains a continuous temporal memory between irregular events, updating the parameter trajectory which preserves long-range dependencies even under uneven event intervals. This design allows DyGSSM to operate efficiently in event-driven streams without reconstructing full snapshots or retraining the model.

5 Experiments

Datasets. We evaluated DyGSSM on 12 different datasets, commonly employed in dynamic graph representation studies. All datasets are available on the SNAP website (https://snap.stanford.edu/) and DyGFormer paper. We provide detailed descriptions of each dataset in Appendix A.1. To ensure a fair comparison with existing methods, we follow two settings in the discrete dynamic graph. In the WinGNN setting, we used the evaluation code (https://github.com/pursuecong/WinGNN.git) provided by the WinGNN authors, while in the HawkesGNN [36] setting, we used the evaluation code (https://github.com/oncemoe/hawkesGNN.git) provided by the HawkesGNN authors. For example,

WinGNN employed 1000 negative samples when computing MRR, whereas HawkesGNN used 100. For continuous dynamic graphs, we followed the evaluation code (https://github.com/yule-BUAA/DyGLib.git) released with DyGFormer. In all cases, we adopted the preprocessing and data splits from the respective repositories. ***Baselines.*** To verify the superiority of DyGSSM, we compared its results with various recent dynamic graph models on the link prediction task in discrete-time dynamic graphs, including EvolveGCN, DGNN [26], dyngraph2vec [37], ROLAND, WinGNN, TransformerG2G [33], DTFormer [34], DySAT [18], VGRNN [38], HTGN [39], M2DNE [40], GHP [41], HawkesGNN and DG-Mamba [22], and continuous-time dynamic graphs, including JODIE [42], DyRep [43], TGAT [44], TGN [45], CAWN [46], EdgeBank [4], TCL [47], GraphMixer [48], DyGFormer [31], and FreeDyG [49]. We did not compare DyGSSM with GraphSSM because GraphSSM focuses on node classification, which differs from our task. We explain each of the baseline methods in Appendix A.2. We describe the evaluation metrics and implementation details in Appendix B.

5.1 Link Prediction Results

Table 1 presents the link prediction results for DyGSSM and SOTA models in WinGNN settings (see Table 5 in Appendix A.1 for the dataset statistics). In this table, the results for EvolveGCN-H, EvolveGCN-O, DGNN, Dyngraph2vec, ROLAND, and WinGNN were taken directly from the WinGNN paper, while we ran the remaining models using the same train/test split and random seed. As shown, DyGSSM outperformed SOTA models in 18 out of 20 cases with substantial improvements. For instance, on Bitcoin-Alpha, DyGSSM boosted accuracy to 92.33% and AUC to 96.71%, far exceeding the second best baselines. Similar trends were observed on DBLP and Reddit-Title, where DyGSSM achieved high accuracy (97.63% and 99.79%, respectively) and substantially higher MRR and Recall@10 compared to the second best methods. On the UCI dataset, DyGSSM delivered the best performance across all metrics, improving accuracy to 96.82%, AUC to 98.56%, and Recall@10 to 58.43%, consistently surpassing competing models. On Bitcoin-OTC, while DTFormer and WinGNN achieved the highest AUC and Recall@10, respectively, DyGSSM slightly improved accuracy while substantially improving MRR. Importantly, across datasets where older baselines struggled with scalability or memory issues, DyGSSM did not encounter Out-Of-Memory (OOM) errors and consistently provided large performance margins, particularly in MRR and Recall@10. A key observation is that in datasets with large snapshots, such as Bitcoin-Alpha and Reddit-Title, the improvement was promising. This suggests that as the dataset gets bigger in terms of snapshots, SSM can effectively capture long-range dependencies in those datasets.

Table 1: Link prediction performance comparison on five datasets. The best and second best results are shown in **bold** and underlined, respectively. We repeated the experiment with 10 random seeds and reported the average metrics with standard deviation. The * indicates that due to memory constraints, the number of negative samples was reduced from 1000 to 50. OOM indicates that OOM occurred when we attempted to run the model in our environment, even with smaller negative samples.

Dataset	Metric	EvloveGCN-H	EvloveGCN-O	DGNN	dyngraph2vec	ROLAND	WinGNN	TransformerG2G	DTFormer	DG-Mamba	DyGSSM
Bitcoin-Alpha	Accuracy	51.99±0.2546	57.44±0.4096	OOM	OOM	66.21±2.7566	81.17±0.5058	OOM	80.44±0.0238	OOM	**92.33±0.0013**
	AUC	63.71±1.0318	68.93±0.9144	OOM	OOM	90.21±1.1762	91.43±0.3259	OOM	95.62±0.0174	OOM	**96.71±0.0276**
	MRR	3.28±0.2845	2.52±0.1014	OOM	OOM	14.52±0.6506	36.74±3.9389	OOM	*	OOM	**62.97±0.0281**
	Recall@10	7.06±1.1900	5.27±0.5093	OOM	OOM	31.25±2.2782	64.55±3.6126	OOM	*	OOM	**88.69±0.0618**
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	0.5±0.0000	77.49±0.0266	OOM	**88.45±0.0027**
	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	58.43±0.0594	**97.59±0.0034**	OOM	94.37±0.0478
	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	*	*	OOM	**52.49±0.0165**
	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**	*	*	OOM	70.47±0.0281
DBLP	Accuracy	63.17±0.4138	65.24±0.5294	OOM	OOM	62.87±0.5908	68.43±0.4135	49.99±0.0000	70.07±0.0130	51.97±0.0211	**97.19±0.0002**
	AUC	70.91±0.3823	72.64±0.4697	OOM	OOM	77.79±0.1689	77.87±0.3050	53.01±0.00946	77.80±0.0167	52.17±0.0293	**99.37±0.0005**
	MRR	2.55±0.0032	2.48±0.0038	OOM	OOM	6.60±0.0047	7.46±0.0020	3.42±0.0028	*	*	**27.90±0.0449**
	Recall@10	5.12±0.0310	4.84±0.0023	OOM	OOM	13.48±0.0132	16.63±0.0299	6.87±0.0046	*	*	**62.11±0.0732**
Reddit-Title	Accuracy	85.85±0.0164	77.46±1.2696	OOM	OOM	93.42±0.0073	99.55±0.0009	OOM	82.74±0.0008	OOM	**99.79±0.0004**
	AUC	93.87±0.0054	97.17±0.2683	OOM	OOM	97.90±0.0001	99.87±0.0002	OOM	94.25±0.0004	OOM	**99.99±0.0000**
	MRR	3.28±0.0198	1.31±0.0213	OOM	OOM	35.11±0.0928	29.91±0.0829	OOM	*	OOM	**66.62±0.0211**
	Recall@10	5.05±0.6796	1.81±0.2453	OOM	OOM	61.13±0.0970	60.46±0.2910	OOM	*	OOM	**97.47±0.0205**
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	50.00±0.0000	78.84±0.0160	63.52±0.0040	**96.82±0.0148**
	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	65.32±0.0809	87.10±0.0149	58.88±0.0040	**98.56±0.0093**
	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	17.46±0.0422	19.36±0.0769	17.14±0.0095	**25.95±0.0570**
	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	29.92±0.0487	29.30±0.1027	28.91±0.0108	**58.43±0.1529**

Table 2 shows the comparison of DyGSSM with SOTA models in HawkesGNN settings (see Table 6 in Appendix A.1 for the dataset statistics). We obtained all the results from the HawkesGNN paper and ran DyGSSM under the same settings. As presented, DyGSSM outperformed all the models on five out of seven datasets. Particularly, on the Reddit-Title and Reddit-Body datasets, DyGSSM outperformed the second-best model by 17%. We also compared DyGSSM with models designed for

continuous-time dynamic graphs under both inductive and transductive settings. As shown in Table 3 (see Table 7 in Appendix A.1 for the dataset statistics), DyGSSM outperformed all competing models across five datasets in the transductive and inductive setting with respect to Average Precision (AP). Particularly, DyGSSM outperformed the second-best method by about 50% in UN Trade dataset both in inductive and transductive settings. We also compared the AP of our model with several baseline models on the discrete datasets, as presented in Table 8 (see Appendix C). As shown, our model outperformed other models except on Bitcoin-OTC dataset.

We encountered an OOM issue (Table 1) when sampling 1000 negative edges (following WinGNN settings) to compute MRR and Recall@10 for DTFormer, DGMamba, and TransformerG2G. To address this, we reduced the number of negative samples to 50 edges for these calculations. We observed that DyGSSM outperformed all the models on DBLP, Reddit-Title, and Bitcoin-Alpha (see Table 9 in Appendix C). On Bitcoin-OTC, DTFormer achieved better MRR and Recall@10, while DyGSSM was the second-best model. If a model still encounters an OOM issue there, we report it as OOM in Table 1; otherwise, we mark it with $*$. Finally, Table 10 (see Appendix C) presents comparisons of AUC for transductive and inductive continuous-time dynamic graph link prediction tasks, where DyGSSM again consistently outperformed baselines.

Table 2: Overall performance (MRR@100) comparison on seven datasets. Each experiment was conducted using three random seeds, and the average performance is reported along with the standard error.

Methods	Bitcoin-OTC	Bitcoin-Alpha	UCI	Reddit-Title	Reddit-Body	AS733	StackOverflow
DySAT	21.39 ± 2.79	19.16 ± 2.21	23.31 ± 9.42	17.46 ± 4.18	13.87 ± 3.90	25.10 ± 1.71	OOM
EvolveGCN	7.84 ± 0.09	6.65 ± 0.55	7.33 ± 0.15	30.67 ± 0.00	18.55 ± 0.02	42.06 ± 0.00	31.21 ± 0.48
Roland	30.94 ± 0.70	32.97 ± 1.78	17.04 ± 2.30	46.33 ± 0.27	38.57 ± 0.42	21.21 ± 5.73	38.57 ± 1.44
WinGNN	3.86 ± 1.26	3.90 ± 0.84	2.37 ± 0.13	4.19 ± 1.25	2.69 ± 0.38	4.29 ± 2.10	7.51 ± 0.67
VGRNN	6.62 ± 0.10	6.49 ± 0.29	6.96 ± 0.08	OOM	17.19 ± 0.14	41.94 ± 2.04	OOM
HTGN	6.36 ± 0.06	7.72 ± 0.66	8.67 ± 0.43	11.50 ± 0.98	10.70 ± 0.52	13.86 ± 0.58	OOM
GraphMixer	43.67 ± 0.25	35.72 ± 0.41	33.63 ± 0.02	38.32 ± 0.01	33.15 ± 0.02	28.86 ± 0.00	OOM
M2DNE	7.82 ± 1.05	5.49 ± 0.29	8.86 ± 0.44	5.40 ± 0.05	6.03 ± 0.38	19.43 ± 0.12	OOM
GHP	3.40 ± 0.41	3.40 ± 0.46	4.15 ± 0.14	16.00 ± 2.32	8.33 ± 2.00	22.15 ± 4.88	OOM
Hawkes-GCN	46.16 ± 0.45	**47.87 ± 5.85**	35.61 ± 0.06	47.44 ± 0.20	36.44 ± 0.42	44.34 ± 0.41	46.41 ± 0.31
Hawkes-GAT	**51.34 ± 0.07**	40.66 ± 0.25	35.59 ± 1.58	50.84 ± 0.05	40.97 ± 0.47	45.95 ± 0.79	48.83 ± 0.14
DyGSSM	35.75±0.00	30.22 ± 0.00	**36.08 ± 0.03**	**59.38 ± 0.00**	**48.04 ± 0.00**	**52.64 ± 0.00**	**52.43±0.00**

Table 3: Average Precision (AP) for transductive and inductive dynamic link prediction with random negative sampling strategies. The best and second best results are shown in **bold** and underlined, respectively. The results are taken from the DyGFormer and FreeDyG papers. Since the FreeDyG authors did not evaluate their model on Can. Parl, US Legist, and UN Trade, we used the results reported for DyGFormer on these datasets and marked FreeDyG with "–". Inductive results for EdgeBank were not reported by either DyGFormer or FreeDyG and are marked as "–". .

Settings	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	FreeDyG	DyG-Mamba	DyGSSM
Transduction	Enron	79.10±0.85	82.02±3.07	72.58±0.79	85.33±1.05	89.56±0.09	83.53±0.00	79.70±0.71	81.08±0.73	92.47±0.12	92.51±0.05	93.14±0.08	**94.64 ± 0.01**
	UCI	87.65±1.85	70.24±0.32	79.55±0.83	90.69±0.45	94.35±0.11	76.20±0.00	88.12±2.73	93.50±0.49	95.76±0.15	96.28±0.11	96.14±0.14	**98.95 ± 0.00**
	Can. Parl.	69.26±0.31	66.54±2.76	70.73±0.72	70.88±2.34	69.82±2.34	64.55±0.00	68.67±2.67	77.04±0.46	97.36±0.45	-	98.20±0.52	**99.99 ± 0.00**
	US Legis.	75.05±1.52	75.34±0.39	68.52±3.16	75.99±0.58	70.58±0.48	58.39±0.00	69.59±0.48	70.74±1.02	71.11±0.59	-	73.66±1.13	**92.91 ± 0.04**
	UN Trade	64.94±0.31	63.21±0.93	61.47±0.18	65.03±1.37	65.39±0.12	60.41±0.00	62.21±0.03	62.61±0.27	66.46±1.29	-	68.51±0.17	**99.99 ± 0.00**
Inductive	Enron	80.72 ± 1.39	74.55 ± 3.95	67.05 ± 1.51	77.94 ± 1.02	86.35 ± 0.51	-	76.14 ± 0.79	75.88 ± 0.48	89.76 ± 0.34	89.69 ± 0.17	91.14 ± 0.07	**97.58 ± 0.00**
	UCI	79.86 ± 1.48	57.48 ± 1.87	79.54 ± 0.48	88.12 ± 2.05	92.73 ± 0.06	-	87.36 ± 2.03	91.19 ± 0.42	94.54 ± 0.12	94.85 ± 0.10	94.15 ± 0.04	**97.06 ± 0.00**
	Can. Parl.	53.92 ± 0.94	54.02 ± 0.76	55.18 ± 0.79	54.10 ± 0.93	55.80 ± 0.69	-	54.30 ± 0.66	55.91 ± 0.82	87.74 ± 0.71	-	90.05 ± 0.86	**99.99± 0.00**
	US Legis.	54.93 ± 2.29	57.28 ± 0.71	51.00 ± 3.11	58.63 ± 0.37	53.17 ± 1.20	-	52.59 ± 0.97	50.71 ± 0.76	54.28 ± 2.87	-	59.52 ± 0.54	**78.78 ± 0.06**
	UN Trade	59.65 ± 0.77	57.02 ± 0.69	61.03 ± 0.18	58.31 ± 3.15	65.24 ± 0.21	-	62.21 ± 0.12	62.17 ± 0.31	64.55 ± 0.62	-	65.87±0.40	**97.39 ± 0.02**

In addition to prediction performance, we also highlight DyGSSM's advantages in model efficiency. Figures 4 and 5 (see Appendix D) compare model parameter sizes across different datasets on a logarithmic scale. DyGSSM consistently has the smallest parameter sizes compared to other SOTA models.

We also performed complexity analysis of proposed $\mathcal{RW}$ for both discrete and continuous datasets (see Table 11,12 in Appendix D). The proposed caching mechanism yields consistent speedups between 1.8× and 2.4×, with a maximum of 3.9× improvement on large-scale graphs, corresponding to an average 50–60% reduction in random-walk time. The gains become more substantial on larger graphs. For example on StackOverflow dataset (≈2.6M nodes, ≈63M edges) proposed $\mathcal{RW}$ was 2× faster. Since only structurally updated nodes trigger recomputation, runtime scales linearly with

active nodes—not total graph size—enabling real-time feasibility even on million-edge networks. Note that DyGSSM retains only $\mathcal{RW}$ information related to the most recent snapshot that indicates constant memory footprint over time. In the very high-churn snapshot, worst-case scenario—the framework smoothly degrades to the standard (non-caching) $\mathcal{RW}$ complexity, ensuring no negative impact on correctness or embedding quality, only a temporary reduction in computational gain.

To compare the extent to which DyGSSM relies on local and global structural components, we also analyzed the temporal evolution of the learned attention weights on the UCI dataset (see Fig. 6 in Appendix E). The average attention weights remain stable over time, with slightly higher values assigned to the local component (0.514) compared to the global component (0.486). This indicates that the model maintains a balanced contribution from both components, with a mild preference for local neighborhood information during the fusion process.

5.2 Ablation study

To evaluate the contribution of each component in DyGSSM, we performed an ablation study utilizing the DBLP and UCI datasets. Specifically, we computed MRR and Recall@10 after disabling attention, local and global embeddings, and SSM. We selected 50 and 1000 negative samples per positive edge in DBLP and UCI, respectively. Table 4 and Table 13 (see Appendix E) show that all components contribute to DyGSSM's overall performance. In DBLP, removing global information led to substantial drop in performance, whereas in UCI, local information was most critical (as the attention scores also show (see Fig. 6 in Appendix E). We observed performance drop when we disabled the SSM and the attention mechanism, too. Both ablation studies confirm the necessity of each component.

To explicitly evaluate the contribution of the HiPPO algorithm in generating the SSM projection matrix, we conducted an experiment on DBLP, UCI and AS733 datasets where the SSM matrix was randomly initialized using a Gaussian distribution (see Table 14 in Appendix E). The proposed HiPPO initialization consistently improved the model performance across three datasets of varying graph sizes under two different settings (i.e., WinGNN and HawkesGNN). These results demonstrate that HiPPO initialization provides a more structured and informative state representation, enabling the model to capture temporal dependencies more effectively and achieve superior overall performance.

Table 4: Ablation results for DyGSSM on DBLP dataset.

Model	MRR	Recall@10
No global information	23.99±0.0050	58.91±0.0205
No local information	56.40±0.0021	92.64±0.0010
No SSM	61.52±0.0196	97.08±0.0120
No attention	69.55±0.0149	99.18±0.0007
DyGSSM	**79.11±0.0442**	**99.85±0.0005**

6 Conclusion

In this study, we propose DyGSSM, a multi-view dynamic graph representation learning approach for link prediction tasks. We trained DyGSSM in a supervised manner, leveraging both the local and global structure of each node in each snapshot to generate two distinct node embeddings. We integrate these embeddings using a lightweight attention mechanism. To mitigate RW cost, we introduced a caching mechanism that reduce the complexity and time of running RW on each time steps. To effectively incorporate past information when updating the model parameters, and to avoid the need for numerous hyperparameters, we utilized an SSM-based approach using the HiPPO algorithm to incorporate a meta-learning strategy into DyGSSM. Experiments on 12 public datasets with two training settings show that DyGSSM outperforms SOTA models in 32 out of 36 evaluation metrics. As future work, we plan to extend DyGSSM to downstream tasks such as node classification to further validate its representational power on real-world dynamic graphs. In addition, we aim to optimize the $\mathcal{RW}$ component through parallelized mini-batch sampling, graph partitioning, and subgraph-based $\mathcal{RW}$ to improve scalability on large and dense networks. Finally, we intend to explore replacing the $\mathcal{RW}$ process with learned graph embeddings.

References

[1] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370, 2020. 1

[2] Andrea Cini, Ivan Marisca, Filippo Maria Bianchi, and Cesare Alippi. Scalable spatiotemporal graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 7218–7226, 2023. 1

[3] Kiarash Shamsi, Friedhelm Victor, Murat Kantarcioglu, Yulia Gel, and Cuneyt G Akcora. Chartalist: Labeled graph datasets for utxo and account-based blockchains. *Advances in Neural Information Processing Systems*, 35:34926–34939, 2022. 1

[4] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction. *Advances in Neural Information Processing Systems*, 35:32928–32941, 2022. 1, 8, 16

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[7] Cagri Ozdemir, Mohammad Al Olaimat, Yashu Vashishath, Serdar Bozdag, and Alzheimer's Disease Neuroimaging Initiative. Igcn: Integrative graph convolutional networks for multimodal data. *arXiv preprint arXiv:2401.17612*, 2024.

[8] Ziynet Nesibe Kesimoglu and Serdar Bozdag. Supreme: multiomics data integration using graph convolutional networks. *NAR Genomics and Bioinformatics*, 5(2):lqad063, 2023.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 1

[10] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 2358–2366, 2022. 1, 2, 3, 15, 17

[11] Yifan Zhu, Fangpeng Cong, Dan Zhang, Wenwen Gong, Qika Lin, Wenzheng Feng, Yuxiao Dong, and Jie Tang. Wingnn: Dynamic graph neural networks with random gradient aggregation window. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3650–3662, 2023. 2, 3, 15, 17

[12] Jiapeng Wu, Meng Cao, Jackie Chi Kit Cheung, and William L Hamilton. Temp: Temporal message passing for temporal knowledge graph completion. *arXiv preprint arXiv:2010.03526*, 2020. 3

[13] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858, 2019. 1

[14] Hao Li, Hao Jiang, Fan Jiajun, Dongsheng Ye, and Liang Du. Dynamic neural dowker network: Approximating persistent homology in dynamic directed graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1554–1564, 2024. 1

[15] Stephen Bonner, Amir Atapour-Abarghouei, Philip T Jackson, John Brennan, Ibad Kureshi, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Temporal neighbourhood aggregation: Predicting future links in temporal graphs via recurrent variational graph convolutions. In *2019 IEEE international conference on big data (Big Data)*, pages 5336–5345. IEEE, 2019. 1

[16] Hansheng Xue, Luwei Yang, Wen Jiang, Yi Wei, Yi Hu, and Yu Lin. Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal rnn. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part I*, pages 282–298. Springer, 2021. 1

[17] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370, 2020. 1, 2, 3, 15

[18] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 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*, pages 519–527, 2020. 1, 2, 3, 8, 16

[19] Dingsu Wang, Yuchen Yan, Ruizhong Qiu, Yada Zhu, Kaiyu Guan, Andrew Margenot, and Hanghang Tong. Networked time series imputation via position-aware graph enhanced variational autoencoders. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2256–2268, 2023. 2, 3

[20] Jacob Miller, Vahan Huroyan, and Stephen Kobourov. Balancing between the local and global structures (lgs) in graph embedding. In *International symposium on graph drawing and network visualization*, pages 263–279. Springer, 2023. 2

[21] Zifeng Ding, Yifeng Li, Yuan He, Antonio Norelli, Jingcheng Wu, Volker Tresp, Michael Bronstein, and Yunpu Ma. Dygmamba: Efficiently modeling long-term temporal dependency on continuous-time dynamic graphs with state space models. *arXiv preprint arXiv:2408.04713*, 2024. 2, 3

[22] Haonan Yuan, Qingyun Sun, Zhaonan Wang, Xingcheng Fu, Cheng Ji, Yongjian Wang, Bo Jin, and Jianxin Li. Dg-mamba: Robust and efficient dynamic graph structure learning with selective state space models. *arXiv preprint arXiv:2412.08160*, 2024. 2, 4, 8, 16

[23] Jintang Li, Ruofan Wu, Xinzhou Jin, Boqun Ma, Liang Chen, and Zibin Zheng. State space models on temporal graphs: A first-principles study. *Advances in Neural Information Processing Systems*, 37:127030–127058, 2024. 2

[24] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré. How to train your hippo: State space models with generalized orthogonal basis projections. *arXiv preprint arXiv:2206.12037*, 2022. 2, 5

[25] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. 2, 5

[26] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020. 3, 8, 15

[27] Yanan Jiang, Huiyuan Luo, Qiang Xu, Zhong Lu, Lu Liao, Huajin Li, and Lina Hao. A graph convolutional incorporating gru network for landslide displacement forecasting based on spatiotemporal analysis of gnss observations. *Remote Sensing*, 14(4):1016, 2022. 3

[28] Cheng Yang, Chunchen Wang, Yuanfu Lu, Xumeng Gong, Chuan Shi, Wei Wang, and Xu Zhang. Few-shot link prediction in dynamic networks. In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pages 1245–1255, 2022. 3

[29] Guangyin Jin, Lingbo Liu, Fuxian Li, and Jincai Huang. Spatio-temporal graph neural point process for traffic congestion event prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 14268–14276, 2023. 3

[30] Mahdi Biparva, Raika Karimi, Faezeh Faez, and Yingxue Zhang. Todyformer: Towards holistic dynamic graph transformers with structure-aware tokenization. *arXiv preprint arXiv:2402.05944*, 2024. 3

[31] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. *Advances in Neural Information Processing Systems*, 36: 67686–67700, 2023. 8, 16

[32] Xue Ye, Shen Fang, Fang Sun, Chunxia Zhang, and Shiming Xiang. Meta graph transformer: A novel framework for spatial–temporal traffic prediction. *Neurocomputing*, 491:544–563, 2022. 3

[33] Alan John Varghese, Aniruddha Bora, Mengjia Xu, and George Em Karniadakis. Transformerg2g: Adaptive time-stepping for learning temporal graph embeddings using transformers. *Neural Networks*, 172:106086, 2024. 3, 8, 15

[34] Xi Chen, Yun Xiong, Siwei Zhang, Jiawei Zhang, Yao Zhang, Shiyang Zhou, Xixi Wu, Mingyang Zhang, Tengfei Liu, and Weiqiang Wang. Dtformer: A transformer-based method for discrete-time dynamic graph representation learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 301–311, 2024. 3, 8, 15

[35] Dongyuan Li, Shiyin Tan, Ying Zhang, Ming Jin, Shirui Pan, Manabu Okumura, and Renhe Jiang. Dyg-mamba: Continuous state space modeling on dynamic graphs. *arXiv preprint arXiv:2408.06966*, 2024. 4

[36] QingGuo Qi, Hongyang Chen, Minhao Cheng, and Han Liu. 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*, pages 1138–1149, 2025. 7

[37] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187:104816, 2020. 8, 15

[38] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. *Advances in neural information processing systems*, 32, 2019. 8, 16

[39] Menglin Yang, Min Zhou, Marcus Kalander, Zengfeng Huang, and Irwin King. Discrete-time temporal network embedding via implicit hierarchical learning in hyperbolic space. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 1975–1985, 2021. 8, 16

[40] Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S Yu, and Yanfang Ye. Temporal network embedding with micro-and macro-dynamics. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 469–478, 2019. 8, 16

[41] Jin Shang and Mingxuan Sun. Geometric hawkes processes with graph convolutional recurrent neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4878–4885, 2019. 8, 16

[42] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019. 8, 16

[43] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*, 2019. 8, 16

[44] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020. 8, 16

[45] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020. 8, 16

[46] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. *arXiv preprint arXiv:2101.05974*, 2021. 8, 16

[47] Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. Tcl: Transformer-based dynamic graph modelling via contrastive learning. *arXiv preprint arXiv:2105.07944*, 2021. 8, 16

[48] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks? *arXiv preprint arXiv:2302.11636*, 2023. 8, 16

[49] Yuxing Tian, Yiyan Qi, and Fan Guo. Freedyg: Frequency enhanced continuous-time dynamic graph model for link prediction. In *The twelfth international conference on learning representations*, 2024. 8, 16

[50] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 333–341, 2018. 14

[51] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 221–230. IEEE, 2016. 14

[52] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009. 14

[53] Yichen Hu, Qing Wang, and Peter Christen. Developing a temporal bibliographic data set for entity resolution. *arXiv preprint arXiv:1806.07524*, 2018. 14

[54] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 world wide web conference*, pages 933–943, 2018. 14

A Experimental Details

A.1 Dataset Description

We evaluated DyGSSM on discrete and continuous publicly available benchmarks.

Bitcoin-OTC and ***Bitcoin-Alpha*** are who-trusts-whom network, representing trust relationships among users trading Bitcoin on Bitcoin OTC and Bitcoin Alpha platform [50, 51]. These two datasets have the highest number of snapshots among all five datasets, despite having the lowest number of edges—35,592 and 24,186, respectively.

UCI-Message consists of private message communication exchanged between students at the University of California, Irvine [52]. It has the fewest nodes among all datasets but ranks among the top three in terms of edge density, with 59,835 edges.

DBLP represents a comprehensive list of research papers in computer science. The dataset show research collaborations between two authors, where two authors are connected if they have co-authored at least one paper [53]. Note that we obtained the dynamic DBLP dataset from the WinGNN authors.

Reddit-Title dataset consists of a hyperlink network that captures directed connections between subreddits based on hyperlinks embedded in posts linking from one subreddit to another [54].

Reddit-Body captures networks of hyperlinks between subreddits, where the hyperlinks appear in the body of the posts.

SBM short for Stochastic Block Model, is a widely adopted random graph model designed to simulate the evolution of community structures.

AS short for Autonomous Systems represent a communication network between routers. In this network, the nodes are routers, where each node represents a network or an AS. The edges indicate that two routers exchange traffic or routing information.

StackOverflow is a dataset containing interactions on the Stack Overflow platform. In this dataset, the nodes are users, and an edge between two users appears if one user answered another user's question.

Enron is an email communication network from the Enron Energy Corporation. The dataset was collected over a period of three years. In this network, nodes represent email addresses, and an edge exists from address i to address j if i sent at least one email to j.

Can.Parl. is a political network that shows interactions between Canadian Members of Parliament (MPs). In this dataset, the nodes are MPs from electoral districts, and an edge is formed between two MPs when they both vote "yes" on the same bill.

USLegis dataset is a Senate co-sponsorship network that shows social interactions between legislators in the U.S. The nodes represent senators, and an edge between two nodes indicates how many times those senators co-sponsored a bill together during a given congressional session.

UNTrade is a dataset of food and agriculture trade between 181 countries over the past 30 years. In this dataset, the nodes represent countries, and an edge between two nodes shows the total imports and exports of food and agricultural products exchanged between those countries. A summary of dataset statistics is presented in Tables 5, 6, and 7.

Table 5: Dataset statistics for WinGNN settings.

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^{-3}
DBLP	28,086	162,451	27	9.5423×10^{-3}
Reddit-Title	54,075	571,927	178	1.9592×10^{-5}
UCI	1,899	59,835	28	1.1191×10^{-3}

Table 6: Summary of dataset statistics for HawkesGNN settings.

Dataset	#Nodes	#Edges	Time Steps (Train/Val/Test)	Avg. Degree
UCI	1,899	59,835	35/5/10	0.36
Bitcoin-Alpha	3,777	24,173	95/13/28	0.04
Bitcoin-OTC	5,881	35,588	95/14/28	0.05
Reddit-Title	54,075	571,927	122/35/17	0.06
Reddit-Body	35,776	286,562	122/35/17	0.05
AS733	7,716	1,167,892	70/10/20	2.12
SBM	1,000	4,870,863	35/5/10	97.42
StackOverflow	2,601,997	63,497,050	65/9/18	0.12

Table 7: Statistics of the datasets for continuous dynamic graph.

Dataset	Domain	#Nodes	#Links	Duration	Timestamps
Enron	Social	184	125,235	3 years	22,632
UCI	Social	1,899	59,835	196 days	58,911
Can.Parl.	Politics	734	74,478	14 years	14
USLegis.	Politics	225	60,396	12 congresses	12
UNTrade	Economics	255	507,497	32 years	32

A.2 Description of Baselines

We compare DyGSSM against state-of-the-art models on both discrete-time and continuous-time dynamic graphs.

EvolveGCN [17] introduced a recurrent mechanism to update the network parameters. In other words, it uses GCN to extract the local structure of each snapshot and injects the recurrent neural network (RNN) to capture the dynamism within the parameters of the GCN. In this study, we show the results of EvolveGCN with different temporal encoders (i.e., LSTM vs. GRU) and refer to them as EvolveGCN-O and EvolveGCN-H.

DGNN [26] combined GCN and LSTM to exploit both structured data and temporal information. In their study, they used stack encoder (e.g., LSTM) to capture the dynamics of nodes.

Dyngraph2vec [37] used an encoder-decoder architecture to learn temporal transition in a dynamic graph. They proposed three different settings for their encoder-decoder architectures, composed of dense and recurrent based models.

ROLAND [10] is a meta-learning based approach that update the model parameters of the adjacent snapshots. They introduced a live update based mechanisem on the traditional GNN layer, that makes their model adoptable to convert static graph to dynamic graph learning.

WinGNN [11] is another meta-learning method that introduces an encoder-free architecture to extract the dynamics.

TransformerG2G [33] is a transformer based model that aim to obtain lower-dimensional multivariate Gaussian representations of nodes, that effectively capture long-term temporal dynamics. They trained the transformer encoder from the second timesteps when weights transferred from the pre-trained model for the first graph snapshot embedding.

DTFormer [34] is another transformer based model. They used attention mechanism to capture topological information in each time steps and temporal dynamics of graphs along the timestamps.

DG-Mamba [22] is a SSM based method that design to extract long dependency on dynamic graph. The authors introduced kernelized dynamic message-passing operator. To capture global intrinsic dynamics, we establish the dynamic graph as a self-contained system with SSM.

DySAT [18] learns node representations by jointly applying self-attention across structural neighborhoods and temporal dynamics to capture both relational structure and temporal evolution.

VGRNN [38] is a hierarchical variational model that introduces latent random variables to jointly captures both topology and node attribute changes in dynamic graphs.

HTGN [39] is a model that captures how networks evolve over time by embedding them in hyperbolic space. It uses hyperbolic GNNs, recurrent units, attention, and a stability module to learn evolving patterns effectively and reliably.

M2DNE [40] is a temporal network embedding method that models both micro- and macro-dynamics of evolving networks. It uses a temporal attention to capture fine-grained edge events and a dynamics equation to enforce higher-level structural evolution in node embeddings.

GHP [41] integrates Hawkes processes with a graph convolutional recurrent neural network. It is also computationally efficient, using a constant number of parameters regardless of graph size.

HawkesGNN fused multiple snapshots into a single temporal graph by combining Hawkes process with GNN. They used a Hawkes excitation matrix to model the temporal edges.

JODIE [42] is a coupled recurrent neural network that learns user and item embedding trajectories, predicts future embeddings via a novel projection operator, and accelerates training using the scalable t-Batch algorithm.

DyRep [43] is an inductive deep learning framework that generates low-dimensional node embeddings evolving over time. It models the communication and association dynamics between nodes using a time-scale-dependent multivariate point process.

TGAT [44] is a temporal graph attention layer that aggregates temporal and topological neighborhood features using self-attention and a functional time encoding based on Bochner's theorem

TGN [45] combines memory modules with graph-based operators to achieve superior performance, using a message function, message aggregator, and memory updater.

CAWN [46] captures network dynamics through temporal random walks. CAWs anonymize node identities using hitting counts to maintain inductiveness and motif correlations. These are then encoded by the CAW-N neural network, paired with a constant-time, and constant-memory sampling strategy.

EdgeBank [4] is a memory-based baseline for dynamic link prediction that stores past interactions and predicts edges as positive if observed. It has four variants—unlimited memory, fixed time-window (two versions), and threshold-based, which allow flexible memory management.

TCL [47] TCL is a graph neural network for continuous-time dynamic graphs. It introduces a graph-topology-aware Transformer, a two-stream encoder with co-attentional modeling of interaction dependencies, and a contrastive learning objective that maximizes mutual information between future interaction nodes.

GraphMixer [48] is a simple yet effective architecture composed of three components: an MLP-based link encoder, a neighbor mean-pooling node encoder, and an MLP-based link classifier.

DyGFormer [31] is a Transformer-based model that only relies on nodes' historical first-hop interactions. It encodes neighbor co-occurrences to capture source–destination correlations and uses a patching technique to handle longer histories.

FreeDyG [49] is a continuous-time dynamic graph model for link prediction that enhances learning by encoding node interaction frequency. Unlike prior time-domain methods, it leverages the frequency domain to capture periodic and shifting interaction patterns.

B Evaluation Metrics and Implementation Details

We evaluate the effectiveness of our model using four widely adopted metrics: accuracy, macro-AUC, Mean Reciprocal Rank (MRR), and Recall@10. Among these, MRR and Recall@10 are our primary

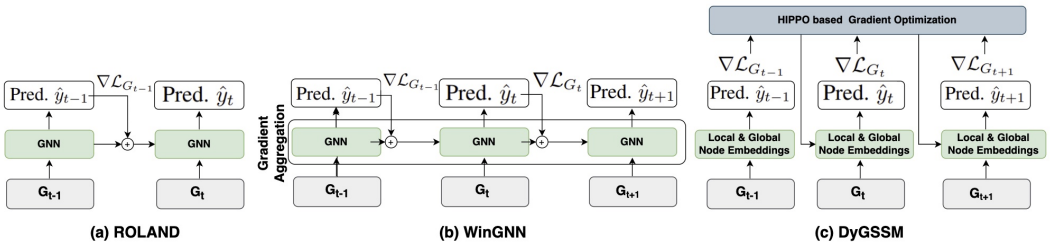

Figure 2: The figure compares how ROLAND, WinGNN, and DyGSSM update their model parameters. (a) ROLAND updates model parameters between consecutive time steps using fixed meta-learning weights. (b) WinGNN updates parameters between time steps with a fixed learning rate. Instead of relying on explicit time encoding, it uses a window-based gradient aggregation mechanism. (c) DyGSSM utilizes the HiPPO-based algorithm to update model parameters without a need to specify a window size.

evaluation metrics, as accuracy and AUC can be overly sensitive to imbalanced class distributions. To ensure a fair comparison with SOTA methods, we follow ROLAND [10] framework for future link prediction task. For each node u with a positive edge (u,v) at time $t+1$, we randomly sample 1,000 negative edges originating from u. The rank of the prediction score for the positive edge (u,v) is then determined relative to the scores of the sampled negative edges. The MRR is computed as the average of the reciprocal ranks across all nodes u. Using the same ranking, Recall@10 is calculated as the proportion of positive edges ranked within the top 10. It is worth noting that, due to memory constraints, we limit the sampling to 50 negative edges for DTFormer and DG-Mamba on the DBLP dataset, as indicated in Table 1 with an asterisk. To have a fair comparison with the HawkesGNN model, we used Average Precision (as they used in their paper) to compare DyGSSM with the SOTA model. We used their source code and integrated DyGSSM into their code. All the results in Table 1 up to WinGNN columns are coming from the WinGNN paper [11]. We followed WinGNN in train test data division, 70% of snapshots for training, and remaining 30% for testing. Training is set for 100 epochs, with patience of 10 epochs for early stopping. We used Adam as our optimizer and repeated the experiment with 10 random seeds to ensure robust error estimation. All the results in Table 2 come from the HawkesGNN paper. All experiments are performed on a single GPU equipped with Nvidia A100 with 80GB of memory.

C Additional Results

Table 8: Average Precision (AP) score comparison on five datasets. The best and second best results are shown in bold and underlined, respectively. We repeated the experiment with 10 random seeds and reported the average metrics with standard deviation.

Dataset	WinGNN	TransformerG2G	DTFormer	DG-Mamba	DyGSSM
DBLP	92.96±0.0019	59.41±0.0077	82.03±0.0119	53.70±0.0385	**98.69±0.0002**
UCI	96.49±0.0119	74.36±0.0660	86.19±0.0208	72.54±0.0063	**98.89±0.0061**
Bitcoin-OTC	92.25±0.0067	63.25±0.0518	**97.83±0.0039**	OOM	95.52±0.0284
Bitcoin-Alpha	93.85±0.0139	OOM	95.97±0.0162	OOM	**97.29±0.0229**
Reddit-Title	99.99±0.0001	OOM	95.80±0.0000	OOM	**99.99±0.0000**

Table 9: Performance comparison of MRR and Recall@10 on four datasets for the models with asterisk in Table 1 using 50 negative samples instead of 1000. The best and second best results are shown in bold and underlined, respectively. We repeated the experiment with 10 random seeds and reported the average metrics with standard deviation. TransformerG2G for DBLP results are not shown as they are available in Table 1. OOM: out-of-memory

Dataset	Metric	DTFormer	DG-Mamba	TransformerG2G	DyGSSM
DBLP	MRR	61.07±0.0102	16.00±0.0042	-	**79.11±0.0442**
	Recall@10	68.14±0.01658	38.71±0.0230	-	**99.85±0.0005**
Bitcoin-OTC	MRR	**77.49±0.0266**	OOM	48.40±0.1327	75.35±0.0802
	Recall@10	**85.95±0.0225**	OOM	60.72±0.1618	78.21±0.0987
Bitcoin-Alpha	MRR	55.22±0.0307	OOM	OOM	**77.88±0.0449**
	Recall@10	71.21±0.0532	OOM	OOM	**79.33±0.0450**
Reddit-Title	MRR	80.12±0.0054	OOM	OOM	**96.43±0.0067**
	Recall@10	85.94±0.0006	OOM	OOM	**99.99±0.0000**

Table 10: AUC-ROC for transductive and inductive dynamic link prediction with random negative sampling strategies. The best and second best results are shown in **bold** and underlined, respectively. The results are taken from the DyGFormer and FreeDyG papers. Since the FreeDyG authors did not evaluate their model on Can. Parl, US Legist, and UN Trade, we used the results reported for DyGFormer on these datasets and marked FreeDyG with "–". Inductive results for EdgeBank were not reported by either DyGFormer or FreeDyG and are marked as "–".

Settings	Datasets	JODIE	DyRep	TGAT	TGN	CAWN	EdgeBank	TCL	GraphMixer	DyGFormer	FreeDyG	DyG-Mamba	DyGSSM
Transductive	Enron	87.96±0.52	84.89±3.00	68.89±1.10	88.32±0.99	90.45±0.14	87.05±0.00	75.74±0.72	84.38±0.21	93.33±0.13	**94.01±0.11**	93.05±0.17	92.60±0.01
	UCI	90.44±0.49	68.77±2.34	78.53±0.74	92.03±1.13	93.87±0.08	77.30±0.00	87.82±1.36	91.81±0.67	94.49±0.26	95.00±0.21	95.32±0.18	**96.95±0.00**
	Can. Parl.	78.21±0.23	73.35±3.67	75.69±0.78	76.99±1.80	75.70±3.27	64.14±0.00	72.46±3.23	83.17±0.53	97.76±0.41	–	98.67±0.29	**99.99±0.00**
	US Legis.	82.85±1.07	82.28±0.32	75.84±1.99	83.34±0.43	77.16±0.39	62.57±0.00	76.27±0.63	76.96±0.79	77.90±0.58	–	78.19±0.64	**92.86±0.03**
	UN Trade	69.62±0.44	67.44±0.83	64.01±0.12	69.10±1.67	68.54±0.18	66.75±0.00	64.72±0.05	65.52±0.51	70.20±1.44	–	72.19±0.09	**99.99±0.00**
Inductive	Enron	81.96±1.34	76.34±4.20	64.63±1.74	78.83±1.11	87.02±0.50	–	72.33±0.99	76.51±0.71	90.69±0.26	89.51±0.20	90.84±0.18	**97.40±0.01**
	UCI	78.80±0.94	58.08±1.81	77.64±0.38	86.68±2.29	90.40±0.11	–	84.49±1.82	89.30±0.57	92.63±0.13	93.01±0.08	91.99±0.03	**96.77±0.00**
	Can. Parl.	53.81±1.14	55.27±0.49	56.51±0.75	55.86±0.75	58.83±1.13	–	55.83±1.07	58.32±1.08	89.33±0.48	–	90.77±0.86	**99.99±0.00**
	US Legis.	58.12±2.35	61.07±0.56	48.27±3.50	62.38±0.48	51.49±1.13	–	50.43±1.48	47.20±0.89	53.21±3.04	–	56.56±1.08	**74.35±0.07**
	UN Trade	62.28±0.50	58.82±0.98	62.72±0.12	59.99±3.50	67.05±0.21	–	63.76±0.07	63.48±0.37	67.25±1.05	–	69.22±0.52	**96.59±0.02**

D Model Scalability

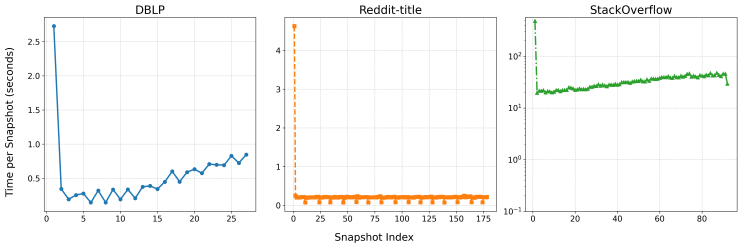

Figure 3: Global neighborhood computation time using RW on DBLP, Reddit-Title, and StackOverflow datasets after applying the caching mechanism. The computation cost per snapshot is initially high, but it significantly decreases when the caching mechanism is used.

Table 11: Runtime comparison of the random-walk (RW) component with and without caching across seven discrete dynamic graph datasets.

Dataset	# Nodes	# Edges	Normal RW (s)	Caching RW (s)	Speedup (× Faster)	% Time Reduction
Bitcoin-OTC	5,881	35,588	5.31	2.67	1.99×	49.8%
Bitcoin-Alpha	3,777	24,173	3.63	1.50	2.42×	58.8%
UCI	1,899	59,835	1.34	0.70	1.91×	47.8%
Reddit-Title	54,075	571,927	61.86	34.26	1.81×	44.6%
Reddit-Body	35,776	286,562	41.61	21.90	1.90×	47.4%
AS733	7,716	1,167,892	6.13	6.01	1.02×	∼2%
Stack Overflow	2,601,997	63,497,050	1795.54	897.01	2.00×	50.0%

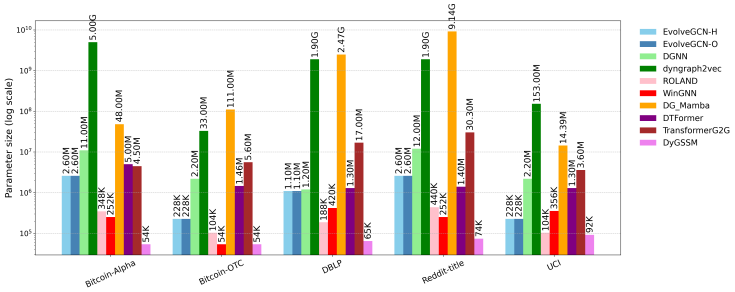

Figure 4: Model parameter size comparison discrete-time dynamic graphs. Each bar represents a model and its number of learnable parameters in millions (M) or thousands (K). DyGSSM consistently has one of the smallest parameter sizes, typically ranging from 50K to 92K. Despite integrating GCN, Conv1D, and light attention, our model remains lightweight and highly scalable.

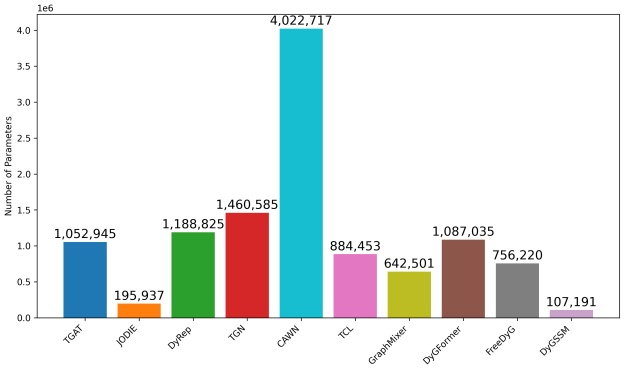

Figure 5: Model parameter size comparison (continuous-time dynamic graph). Each bar represents a model and its number of learnable parameters in millions (M) or thousands (K). DyGSSM has the smallest parameter size.

Table 12: Runtime comparison of the random walk with and without caching across four continuous dynamic graph datasets.

Dataset	# Nodes	# Edges	Without Caching	With Caching	Speedup (×) / % Time Reduction
Reddit	10,984	672,447	5 min 59 s (≈ 359 s)	1 min 32 s (≈ 92 s)	3.9× / 74%
UN Vote	201	1,035,742	14 min 28 s (≈ 868 s)	7 min 17 s (≈ 437 s)	2.0× / 50%
UN Trade	255	507,497	3 min 12 s (≈ 192 s)	0 min 52 s (≈ 52 s)	3.7× / 73%

E Ablation and Attention Results

Table 13: Ablation results for DyGSSM on UCI dataset.

Model	MRR	Recall@10
No global information	30.85±0.0011	55.17±0.0031
No local information	20.11±0.0000	45.40±0.0000
No SSM	27.32±0.0036	55.05±0.0083
No attention	25.57±0.0131	54.76±0.0012
DyGSSM	**42.92±0.0072**	**74.08±0.0018**

Table 14: Comparison of Random and HiPPO-based initialization on the DBLP dataset.

Dataset	Initialization Setting	MRR (↑)	# Nodes	# Edges
DBLP	Gaussian distribution	19.18 ± 0.0026	28,086	162,451
	HiPPO	**27.90 ± 0.0449**		
UCI	Gaussian distribution	33.59 ± 0.0081	1,899	59,835
	HiPPO	**36.08 ± 0.0300**		
AS733	Gaussian distribution	38.14 ± 0.0222	7,716	1,167,892
	HiPPO	**52.64 ± 0.0000**		

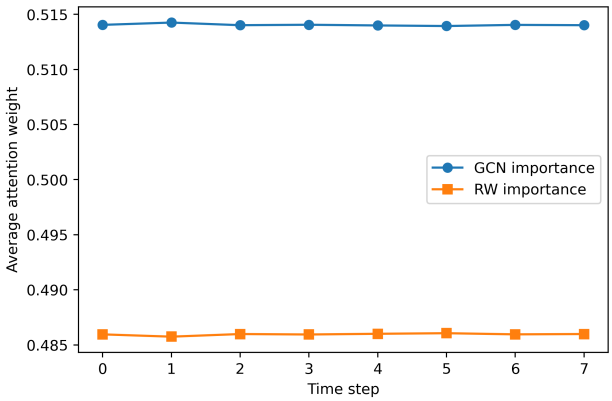

Figure 6: DyGSSM attention to local and global structure on UCI dataset.