# ScaDyG: A New Paradigm for Large-scale Dynamic Graph Learning [Scalable Data Science]

Xiang Wu, Xunkai Li, Rong-Hua Li, Kangfei Zhao Guoren Wang
Beijing Institute of Technology, China
xiangwubit@163.com, cs.xunkai.li@gmail.com, lironghuabit@126.com, zkf1105@gmail.com, wanggrbit@gmail.com

## ABSTRACT

Dynamic graphs (DGs), which capture time-evolving relationships between graph entities, have widespread real-world applications. To efficiently encode DGs for downstream tasks, most dynamic graph neural networks follow the traditional message-passing mechanism and extend it with time-based techniques. Despite their effectiveness, the growth of historical interactions introduces significant scalability issues, particularly in industry scenarios. To address this limitation, we propose ScaDyG, with the core idea of designing a time-aware scalable learning paradigm as follows: 1) Time-aware Topology Reformulation: ScaDyG first segments historical interactions into time steps (intra and inter) based on dynamic modeling, enabling weight-free and time-aware graph propagation within pre-processing. 2) Dynamic Temporal Encoding: To further achieve fine-grained graph propagation within time steps, ScaDyG integrates temporal encoding through a combination of exponential functions in a scalable manner. 3) Hypernetwork-driven Message Aggregation: After obtaining the propagated features (i.e., messages), ScaDyG utilizes hypernetwork to analyze historical dependencies, implementing node-wise representation by an adaptive temporal fusion. Extensive experiments on 12 datasets demonstrate that ScaDyG performs comparably well or even outperforms other SOTA methods in both node and link-level downstream tasks, with fewer learnable parameters and higher efficiency.

**PVLDB Reference Format:**
Xiang Wu, Xunkai Li, Rong-Hua Li, Kangfei Zhao Guoren Wang
Beijing Institute of Technology, China
xiangwubit@163.com, cs.xunkai.li@gmail.com, lironghuabit@126.com, zkf1105@gmail.com, wanggrbit@gmail.com . ScaDyG: A New Paradigm for Large-scale Dynamic Graph Learning [Scalable Data Science]. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/BITNEO/ScaDyG.

## 1 INTRODUCTION

Recently, dynamic graphs (DGs) are widely used in social analysis [29, 32], recommendation [38, 53], and financial management [31, 46]. As a type of high-order relational data, DGs capture the time-evolving relationships between graph entities, providing insights into the dynamic nature of complex systems. To encode DGs, dynamic graph neural networks (DGNNs) are designed to model temporal topology and integrate time-evolving node profiles.

To provide a clear presentation, we first conduct a systematic review of the most prevalent DGNNs and propose the following taxonomy: (1) Discrete-based methods [31, 35, 50, 59] divide a dynamic graph into a sequence of snapshots, with each snapshot treated as an individual static graph. Based on this, these approaches model each snapshot by time-independent GNN (e.g., GAT [41]), while the temporal dependencies between snapshots are captured with sequence models such as RNNs [31, 50]. Despite their simplicity and intuitiveness, they overlook the fine-grained dynamic information within each snapshot. (2) Continuous-based methods [27, 28, 43, 47, 51, 61] emphasize time-based dynamics and focus on interaction granularity, enabling nuanced modeling of temporal dependencies. Specifically, they sample informative historical neighbors and aggregate nodes with learnable aggregators, achieving SOTA performance due to their fine temporal resolution [51]. However, these methods struggle with large-scale DGs because the algorithm complexity of sampling and aggregation scales linearly with the number of historical neighbors [24, 25]. Moreover, they frequently compute node embeddings at each interaction timestamp, leading to redundant calculations. Consequently, most DGNNs are constrained in their scalability to handle large-scale DGs.

During our investigation, we found that recent advancements have introduced techniques to improve the scalability of DGNNs. Specifically, they improve sampling efficiency [25, 56] or avoid redundant calculations [24]. Despite their effectiveness, they still rely on the sampling-aggregation framework, which is constrained by sampling quality, especially when sample sizes are much smaller than the historical neighbors in large-scale scenarios. In such cases, preserving all interactions is necessary to achieve satisfactory performance. However, the potentially thousands of historical neighbors in million-level DGs lead to unaffordable computational costs.

To break the above limitations, we draw inspiration from decoupled scalable GNNs in time-independent graphs [3, 6, 7, 44, 55, 58], which reduce algorithm complexity by separating feature propagation from learnable transformations. Specifically, by designing advanced propagation operators and learnable message aggregators, these methods have achieved SOTA performance across various tasks [22, 23, 49]. In this decoupled framework, weight-free feature propagation is efficiently precomputed through the sparse matrix, eliminating time-based neighborhood sampling and gradient updates. This framework offers a potentially scalable solution for


Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

DGNNs but presents the following two significant challenges: C1: Due to the static topology (i.e., one timestamp) in simple graphs, traditional decoupled methods directly achieve graph propagation. However, the dynamic topology of DGs makes it impractical to preprocess node features at each individual timestamp due to expensive computational overhead. C2: Although traditional decoupled methods apply various propagation rules to different nodes [8, 14, 23], they fail to dynamically capture the evolving temporal dependencies between nodes and their historical neighbors.

To address C1, we propose Time-Aware Topology Reformulation (TTR), which reorganizes historical interactions using a two-step time partitioning principle to achieve fine-grained and weight-free dynamic graph propagation. Notably, previous studies segment the time range of historical interactions into discrete steps with equal intervals, similar to snapshot-based methods. However, they oversimplify each step into a single timestamp, leading to inevitable information loss. Therefore, the key motivation of TTR is to incorporate finer-grained historical interactions within each timestamp. Specifically, we reformulate temporal message passing from historical neighbors to the current time into intra-step propagation and inter-step propagation. In intra-step propagation, messages are propagated to the last timestamp of the step to form intermediate messages. In inter-step propagation, the intermediate messages from all steps are transferred to the current time message. This reformulation enables efficient intra- and inter-step propagation within a single time in preprocessing.

To address C2, inspired by the widely used exponential functions in dynamic modeling, we introduce Dynamic Temporal Encoding (DTE) to further enhance TTR. [1, 43, 61]. Notably, existing fixed exponential functions inadequately capture dynamic temporal dependencies. Therefore, to better model complex temporal patterns, we propose DTE, which combines exponential functions with varying parameters to effectively model the influence decay of historical interactions. The key motivation of our approach is that, beyond the effectiveness of DTE in modeling temporal relationships, its multiplicative property enables seamless integration of intra- and inter-step interactions. Specifically, since nodes exhibit different dynamics across temporal states, determining the weights for these composite functions is critical. We demonstrate that DTE, by applying a learnable transformation to the current message, is equivalent to performing an adaptive weight fusion of composite functions. However, simple transformations of the propagated features obtained by DTE-enhanced TTR result in all nodes sharing the same weights across different temporal states. To address this issue, we introduce Hypernetwork-driven Message Aggregation. The key motivation behind our method is to leverage hypernetwork [9, 39] to generate tailored transformation networks for each node, ensuring effective dynamic modeling of node-wise temporal dependencies.

**Our contributions**. (1) *New Perspective.* In this paper, we address the scalability challenges of existing DGNNs by proposing a novel time-oriented decoupled learning paradigm. (2) *New Method.* We propose ScaDyG, which first employs TTR to achieve weight-free dynamic graph propagation by two-step time partitioning. This process can be efficiently computed using sparse matrix multiplication and is executed only once during pre-processing. To further highlight temporal dependencies between nodes, we introduce DTE, which seamlessly integrates time influence in both inter and intra-step interactions. Subsequently, through Hypernetwork-driven Message Aggregation, we achieve node-wise adaptive representation. (3) *SOTA Performance.* We conduct experiments on 12 datasets, including million-level dynamic graphs. Results on link prediction and node affinity prediction tasks demonstrate that our approach achieves superior or comparable performance to SOTA baselines. Meanwhile, ScaDyG enjoys higher efficiency, with training times up to 60x faster and requiring up to 50x fewer parameters.

## 2 RELATED WORKS

**Dynamic Grpah Neural Networks.** The existing literature on dynamic graph neural networks can be categorized into discrete-based and continuous-based approaches. Discrete-based approach partitions a temporal graph into a series of snapshots with fixed time intervals. Typically, they are equipped with mechanisms dedicated to encoding structural patterns, such as GCN, within the snapshot as well as temporal dynamics, such as RNN, across snapshots [10, 31, 35–37, 48, 50, 59]. Despite their simplicity, these methods overlook the temporal information within snapshots. SimpleDyG [45] is the only method that considers the order of interactions within a snapshot, focusing on fine-tuning a Transformer model for interaction sequences modeling. Continuous-based methods model temporal graphs directly from the finer-grained temporal interactions. They first sample a set of historical interactions from the neighborhood of a given node and aggregate the interactions with carefully designed encoders to compute the temporal embeddings. For instance, temporal random walk-based methods [17, 42] model the propagation of temporal information through the process of random walks. Temporal encoding-based methods develop message massing-based methods temporal directly into message passing schemes [5, 47]. Moreover, temporal point process based methods [27, 40, 43, 61] consider excitation effects of historical interactions to the occurrence of current interaction. However, the number of historical neighbors often increases rapidly with time, leading to a temporal explosion of interactions and scalability challenges. To enhance the efficiency of dynamic GNNs on large-scale DGs, several approaches [25, 56] propose efficient neighborhood sampling algorithms to accelerate the aggregation of neighborhood messages. Orca [24] suggests reusing previously computed embeddings to reduce redundant computations. However, these methods still operate within the sampling-aggregating framework.

**Scalable Graph Neural Networks.** Scalable Graph Neural Networks aim to process large-scale graph data by reducing the computational overhead during model training and inference. Existing methods can be categorized into sampling-based approaches [2, 4, 11, 16, 52, 60] and decouple-based approaches [7, 14, 44, 54, 58], while we primarily focus on the latter line of work. As the seminal decoupled GNN, Simplified Graph Convolution (SGC) [44] first computes the node feature matrix results with feature propagation for multiple hops. The prediction results on downstream tasks, e.g., node classification are achieved by a logistic regression classifier based on node feature. However, it employs a fixed receptive field for each node, posing a limitation to adaptively leverage multi-hop neighborhood information. Therefore, subsequent works enhanced

the node propagation rules by incorporating layer-wise propagation [7, 58] and node-wise propagation [14, 26, 54, 55]. Other methods designed decoupled GNNs on sophisticated types of graphs, such as heterogeneous graphs [49], directed graphs [22] and user-item interaction graphs [13]. Notably, TDLG [1] is the only decoupled-based temporal graph embedding method in the literature. It models temporal associations between interactions using a line graph and exclusively focuses on edge-level tasks. However, its fixed temporal modeling function struggles to capture complex dynamics effectively. Overall, designing a scalable and effective framework for large dynamic graphs remains an open challenge.

# 3 PRELIMINARIES

## 3.1 Dynamic graph representation learning

A dynamic graph can be characterized by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{X})$, where $\mathcal{V}$ represents the set of nodes, $\mathcal{E}$ denotes the set of edges, $\mathcal{T}$ is set of the the timestamps of all edges, and $\mathcal{X} = \{\mathbf{X}_v, \mathbf{X}_e\}$ is the feature matrix, including $d_v$ dimension node features and $d_e$ dimension edge features. Alternatively, it can be conceptualized as a chronologically ordered series of interactions $\mathcal{I} = \{(u_i, v_i, t_i)\}_{i=1}^{k}$, in which each triplet $(u_i, v_i, t_i)$ signifies the establishment of edge $(u_i, v_i)$ at $t_i$, with $k$ representing the total count of interactions. Dynamic graph representation learning aims to derive a mapping function that, for any given timestamp $t$, leverages the accumulated information up to $t$ to project nodes into their respective time-aware embeddings. These embeddings can be utilized for downstream link-level and node-level tasks.

**Temporal message passing.** Dynamic graph neural networks have become the SOTA methods for dynamic graph representation learning. Most existing dynamic GNNs adhere to the temporal message-passing mechanism [25]. Despite their diversity, most 1-hop temporal message passing algorithm is formulated as follows:

$$\mathbf{x}_{m,v}^{t} = \text{Message}(\mathbf{x}_v, t - t'), \quad \mathbf{h}_u^t = \text{Aggregate}\{\mathbf{x}_{m,v}^{t} \mid v \in \mathcal{N}_u^{(t)}\}, \tag{1}$$

where $\mathcal{N}_u^{(t)} = \{(u, v, t') | t' < t\}$ is the set of historical neighbors at current time, $\mathbf{x}_{m,v}^{t}$ is the temporal message computed by the temporal edge $(u, v, t')$, $\mathbf{x}_v$ is the edge feature of the temporal edge $(u, v, t')$, and $\mathbf{h}_u^t$ is the time aware embedding of $u$ at $t$.

# 4 THE PROPOSED METHOD

In this section, we first introduce Time-aware Topology Reformulation, which computes temporal message passing in a decoupled framework. We decompose message computation into intra- and inter-step propagation, utilizing exponential time encoding as a unified time modeling in two phases. In dynamic time encoding, we demonstrate that ScaDyG can dynamically model the time dependency with exponential time encoding. Finally, we present Hypernetwork-driven Message Aggregation. By utilizing a hypernetwork to scale the transformation matrix, we achieve node-wise message passing across different temporal states. The overview of ScaDyG is shown in figure 1.

## 4.1 Time-aware Topology Reformulation

**Decoupling of temporal messages.** The core idea of Time-aware Topology Reformulation is to separate the message-passing process with steps, decoupling the fine-grained intra-step interactions modeling from the coarse-grained inter-step modeling. Specifically, the steps represent a series of equal time intervals covering the time span of historical interactions. For simplicity, all nodes are divided using the same interval. We denote the steps as $s_1, s_2, ...s_L$, which are divided by the series of timestamps $[t_{s_1}, t_{s_2}, \ldots, t_{s_L}]$. The $i$-th step is defined as $(t_{s_{i-1}}, t_{s_i}]$, where $t_{s_i}$ is the largest timestamp of the $i$-th step. Using the steps, the time range for computing a temporal message, i.e., $[t', t)$ can be split into two parts. Suppose $t'$ is in the $i$-th step, the two parts are $[t', t_{s_i})$ and $[t_{s_i}, t)$, separately. Our idea is to first compute the message in $[t', t_{s_i})$ with preprocessing, obtaining an intermediate message, and then compute the message passing in $[t_{s_i}, t)$ with the intermediate messages. For example, in Figure 1, if we need to compute the central node embedding at both 0.5 and 1 timestamps, continuous-time methods would compute the messages of nodes 1, 2, and 3 twice, since they are the historic neighbors at both timestamps. Our method computes the message for nodes 1, 2, and 3 only once, storing it as an intermediate message at time 0.5. At time 1, we simply reference this intermediate result and combine it with the message of nodes 4, 5, and 6, reducing the unnecessary re-computation. We denote the first phase as intra-step propagation and the second as inter-step propagation.

**Intra-step propagation.** Intra-step propagation computes message passing within each step, preserving the temporal information of each interaction, which is oversimplified in discrete-based methods. Specifically, let $t_j$ denote the timestamp of the $j$-th temporal edge in the $i$-th step. The time interval between $t_j$ and the last timestamp of the step is $\Delta t_j = t_{s_i} - t_j$. Therefore, the time intervals of all temporal edges in the step are denoted as $\Delta t_1, \ldots, \Delta t_{k_i}$, where $k_i$ is the number of edges in the step. To encode these time intervals, we introduce an exponential time encoding in Definition 1.

Definition 1. *(Exponential time encoding). Given a set of predefined parameters $\gamma_1, \gamma_2 ..., \gamma_{d_e}$ and a time interval $\Delta t$ , the exponential time encoding of $\Delta t$ is defined as $T_e(\Delta t) = [e^{\gamma_1 \Delta t}, e^{\gamma_2 \Delta t}, ..., e^{\gamma_{d_e} \Delta t}]$.*

We will discuss the effectiveness of this encoding in section 4.2. The temporal features encoded by $T_e$ are represented by $\mathbf{X}_t \in \mathbb{R}^{k \times d_e}$. To incorporate the edges with time information, these time features are element-wisely multiplied with edge features: $\mathbf{X}_e^{t_{s_i}} = \mathbf{X}_t \odot \mathbf{X}_e^i$. Next, we design two operators, $\mathbf{A}_{e,v}$ and $\mathbf{A}_{v,e}$, to propagate edge features to nodes and node features to edges, respectively. Specifically, $\mathbf{A}_{e \to v} \in \mathbb{R}^{|\mathcal{V}| \times k_i}$ is the edge-node adjacent matrix where each element $A_{v_i, e_j}$ is defined as

$$A[v, e] = \begin{cases} 1 & \text{if edge } e \text{ is connected to node } v \\ 0 & \text{if edge } e \text{ is not connected to node } v. \end{cases} \tag{2}$$

$\mathbf{A}_{v \to e} \in \mathbb{R}^{k_i \times |\mathcal{V}|}$ is the node-edge adjacent matrix, obtained by transposing $\mathbf{A}_{e,v}$. We've omitted the superscripts $(i)$ for simplicity, which denotes that the letters are specific to the step $s_i$. Therefore, The 1-hop propagation of edge features to their connected nodes is constructed as $\mathbf{X}_{m,e} = \mathbf{A}_{e \to v} \mathbf{X}_e^{t_{s_i}}$, where the subscript $m$ denotes the message. Here, each node's intermediate message is the sum of the features of the adjacent edges. The reason for using sum instead of mean is to preserve the effect of repeated interactions. To incorporate node features, the initial node features are first propagated to connected edges and then to adjacent nodes, which

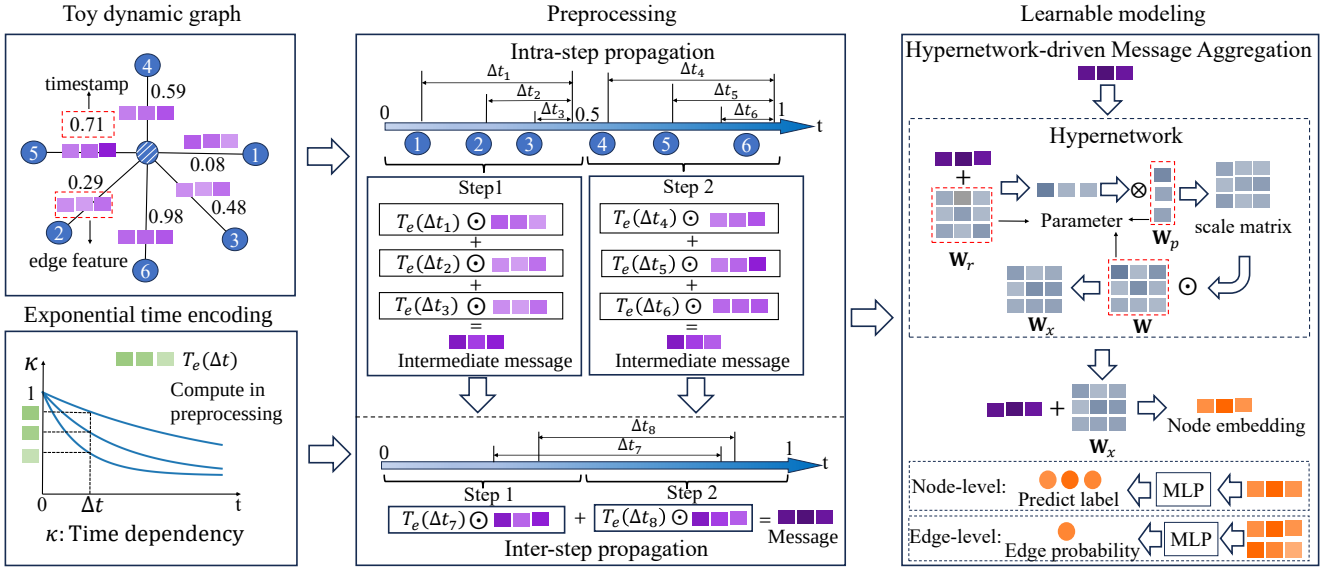



**Figure 1: Overview of ScaDyG. $\odot$ denotes the element-wise multiplication, and $\otimes$ denotes the outer product. We display the propagation from edge-to-node features in two steps as a toy example.**

is computed as $\mathbf{X}_{e,v} = \mathbf{A}_{v\to e}\mathbf{X}_v$. Time is not involved in this process because we just want to transform edge features to node features. Next, we element-wisely multiply $\mathbf{X}_{e,v}$ with the time feature and then propagate it to the nodes using the edge-node matrix $A_{e\to v}$ as previously described, forming $\mathbf{X}_{m,v}$. Finally, we concatenate $X_{m,e}$ with $X_{m,v}$ to derive the intermediate messages $\mathbf{X}_m = [\mathbf{X}_{m,e}||\mathbf{X}_{m,v}]$.

The design of these two operators is based on two insights. Firstly, in dynamic graphs, temporal information is associated with edges, making edge-to-node propagation essential for integrating temporal information into nodes. Secondly, since a node could form edges with another node multiple times, node features should be propagated through each edge to preserve each individual interaction. Therefore, for node-to-node propagation, node features should first propagate to their associated edges and then to the adjacent nodes.

**Multi-hop propagation.** Our framework can conveniently support multi-hop features. To this end, we follow [44] by first performing multi-hop propagation, and then concatenating the results of each hop's propagation. Specifically, the $l$-hop node and edge features are iteratively computed as $\mathbf{X}^{(l)}_{m,v} = \mathbf{A}_{e\to v}\mathbf{A}_{v\to e}\mathbf{X}^{(l-1)}_v$ and $\mathbf{X}^{(l)}_e = \mathbf{A}_{v\to e}\mathbf{A}_{e\to v}\mathbf{X}^{(l-1)}_{m,e}$. We provide results and analysis of multi-hop propagation in section 5.5.

**Inter-step propagation.** Given the intermediate messages at each step, the inter-step propagation first models the time interval from each step to the current time using $T_e$. Then, it aggregates the intermediate messages from all steps to obtain a current message. We denote intermediate messages obtained at all steps, i.e., $s_1, ..., s_L$, as $\mathbf{X}^1_m, ..., \mathbf{X}^L_m$, where $L$ is the largest step so that $t_L \le t$. The time interval between the timestamp of step $s_i$ and current timestamp is $\Delta t_i = t - t_{s_i}$, and then the time feature encoded is $T_e(\Delta t_i)$. To integrate the inter-step time intervals, the intermediate messages at each historical step $i$ are element-wisely multiplied by $T_e(\Delta t_i)$, transferring them to the current time. To obtain the current messages, we sum the transferred message across all steps with $\mathbf{X}^t_m = \sum_{i=1}^{L}\mathbf{X}^i_m$.

## 4.2 Dynamic time encoding

In Time-aware Topology Reformulation, we introduced exponential time encoding as a general time modeling in intra and inter-step propagation. Here, we further demonstrate the time encoding can be as expressive as a dynamic fusion of diverse parameterized exponential functions with a simple learnable transformation. We begin by formulating the concept of composite exponential dependency.

Definition 2. *(Composite exponential dependency). Given an exponential time encoding as* $T_e(\Delta t) = [e^{\gamma_1\Delta t}, e^{\gamma_2\Delta t}, ..., e^{\gamma_{d_e}\Delta t}]$ *and a set of learnable parameters* $[a_1, a_2, ..., a_{d_e}]$. *The composite exponential dependency is defined as* $\kappa(\Delta t) = a_1 e_1^{\gamma_1\Delta t} + a_2 e_2^{\gamma_2\Delta t} + ... + a_{d_e} e^{\gamma_{d_e}\Delta t}$.

Exponential functions have shown their effectiveness in modeling time dependency decay [12, 43, 61]. However, existing methods typically employ a fixed exponential function to model such effects of historical neighbors [1, 43, 61]. A fixed $\gamma$ value fails to characterize the varying temporal patterns exhibited by different nodes and time states. In exponential time encoding, each element of the feature represents the time interval is modeled by a unique exponential function, and the dynamic fusion of the elements captures a diverse range of temporal patterns. Therefore, by applying a weighted sum of these exponential functions, composite exponential (CE) dependency achieves good representational capacities for modeling temporal dependencies [57]. Building on CE dependency, we introduce composite exponential (CE) message passing, a robust implementation of learnable message passing that utilizes CE dependency as the temporal modeling function.

Definition 3. *(Composite exponential message passing). Given an composite exponential dependency* $\kappa(t-t')$ *and a learnable matrix* $\mathbf{W}$. *Assuming a historical neighbor* $v$ *interacts with* $u$ *at* $t'$, $t$ *is the current timestamp, and the time interval* $\Delta t = t - t'$. *Its composite exponential (CE) message passing is defined as* $\mathrm{CE}(\mathbf{x}_v, t-t') = \mathbf{x}_v\kappa(t-t')\mathbf{W}$.

Next, we demonstrate that $\mathrm{CE}(\mathbf{x}_v, t-t')$ can be computed sequentially in preprocessing and learnable modeling.

Proposition 1. *If $\Delta t = t - t'$ can be split by $t_s$ as $\Delta t_1 = t_s - t'$, $\Delta t_2 = t - t_s$, and $\mathbf{W}_1$ is a learnable parameter matrix, $\mathbf{x}_v \odot T_e(\Delta t_1) \odot T_e(\Delta t_2)\mathbf{W}_1$ is equivalent to $\mathbf{x}_v \kappa(\Delta t)\mathbf{W}$.*

The proof of Proposition 1 is given in Appendix A . From Proposition 1, we can observe $\mathbf{x}_v \odot T_e(\Delta t_1) \odot T_e(\Delta t_2)$ involves no learnable parameter. When setting $t_s = t_{s_i}$, i.e., the largest timestamp of the $i$-th step, $\mathbf{x}_v^{t_i} = \mathbf{x}_v \odot T_e(\Delta t_1)$ corresponds to the intra-step propagation. Conversely, $\mathbf{x}_v^t = \mathbf{x}_v^{t_{s_i}} \odot T_e(\Delta t_2)$ corresponds to inter-step propagation. In Time-aware Topology Reformulation, we sum all messages across nodes to obtain $\mathbf{X}_m^t$. By multiplying $\mathbf{X}_m^t$ with a learnable matrix $\mathbf{W}$, we effectively implement CE message passing from all historical neighbors. Although no learnable parameters is involved during preprocessing, our analysis shows that ScaDyG achieves dynamic modeling of temporal dependencies.

### 4.3 Hypernetwork-driven Message Aggregation

Although we have established a decoupled framework dynamically modeling temporal dependencies, simple message aggregation with $\mathbf{W}$ has the following limitations. 1) It applies the same transformation for all nodes, ignoring the dynamic differences between individual nodes. 2) In the inference stage, $\mathbf{W}$ is frozen. This hinders the model to adapt and update with the evolving of new data. To address the limitations, we introduce a Hypernetwork-driven Message Aggregation, enabling node-specific and dynamic updates to the transformation matrix as time evolves.

A hypernetwork [9] is a meta-model designed to generate weights for a main network. In our context, the main network refers to the transformation matrix for aggregating messages. The idea for generating node-specific weights is to select node-related identifiers as inputs to the hypernetwork. The messages of nodes, i.e., $\mathbf{X}_s^t$ are natural choices for these identifiers, as it captures the historical neighborhood interactions that characterize the node's current state. To generate the weights, we apply gird-wise projections [39] for two reasons. 1) Fewer extra parameters. Only an additional matrix and a vector are introduced. 2) Good Interpretability. Different node patterns and states are reflected in specific regions of projected weights, as we will show in section 5.4.

Specifically, we first introduce a primary weight matrix $\mathbf{W}$, and use a projection weight generated by hypernetwork to scale it to a node-specific matrix. To this end, we introduce two additional learnable parameters $\mathbf{W}_r \in \mathbb{R}^{d\times d}$ and $\mathbf{W}_p \in \mathbb{R}^d$ apart from $\mathbf{W}$. The aim of $\mathbf{W}_r$ is to generate row vector specific node features, while $\mathbf{W}_p$ is a column vector extending it into a scale matrix with the outer product. Finally, we perform element-wise multiplication with the scale matrix and $\mathbf{W}$, projecting it into a node-specific transformation matrix $\mathbf{W}_x$. The projection is formulated as:

$$\mathbf{W}_x = (\sigma((\mathbf{W}_r\mathbf{X}^t) \otimes \mathbf{W}_p)) \odot \mathbf{W}, \tag{3}$$

where $\sigma$ is the sigmoid function, and $\otimes$,$\odot$ are the element-wise product and the outer product, respectively. To obtain the final node representation, we aggregate the messages in inter-step propagation with a feature transformation using $\mathbf{W}_x$ :

$$\mathbf{X}^t = \mathbf{X}_s^t\mathbf{W}_x. \tag{4}$$

**Training objective.** For node-level tasks, the node representations are fed into an MLP for generating prediction labels. We use the cross entropy based objective function as follows:

$$\mathcal{L}_{node} = \sum_i \sum_{u\in\mathcal{V}} -\mathrm{Y}_{vi} \ln\left(\mathrm{Softmax}\left(\mathrm{MLP}(\mathbf{X}^t)\right)_{ui}\right), \tag{5}$$

where the subscript $vi$ denotes the $i$-th label of node $u$. For link prediction, we replace $\mathbf{X}^t$ with the concatenation of source and target node representation, and sample a negative edge for each existing edge following [51] to construct training pairs.

## 5 EXPERIMENTS

In this section, we conduct extensive experiments on our approach. To begin with, we introduce 12 benchmark datasets along with prevalent DGNN baselines. Subsequently, we present the temporal-based evaluation methodology. Details about these experimental setups can be found in Appendix. After that, we aim to address the following questions: **Q1**: Compared to existing DGNNs, can ScaDyG achieve SOTA predictive performance? **Q2**: What is the running efficiency of ScaDyG, especially in large-scale scenarios? **Q3**: If ScaDyG is effective, what contributes to its performance? **Q4**: How robust is ScaDyG when dealing with hyperparameters?

### 5.1 Experimental settings

**Datasets and Baselines.** We conduct link- and node-level evaluations on 12 datasets from 5 real-world application domains. The statistics of these datasets are presented in Table 1. For baselines, we compare ScaDyG with the following baselines: (1) Discrete-based methods: EvolveGCN [31] and ROLAND [50]); (2) Continuous-based methods: DyGFormer [51], GraphMixer [5], TGAT [47], TGN [34], DyRep [40], JODIE [21]. Detailed descriptions of the above datasets and baselines are provided in Appendix B.2.

**Evaluation.** Regarding link-level prediction, we select transductive link prediction, a widely used task in existing studies [5, 34, 47, 50, 51]. In link prediction, we follow [50] to use the ranking setting and MRR as the evaluation metric. We also report AP and AUC in binary classification setting in Appendix B.4. As for node-level experiments, we choose the recently introduced node affinity prediction [15], where the model predicts a multi-dimensional label representing the affinity between a node and a set of nodes. We use NDCG as the evaluation metric. We provide detailed background and motivation for the tasks, metrics and datasets in Appendix B.1.

### 5.2 Overall performance

**Link-level performance.** To answer **Q1**, we first report the performance of ScaDyG in predicting temporal relationships between nodes on MRR, as shown in Table 2. Our findings indicate that ScaDyG consistently achieves the highest or second-highest performance on nine datasets, with an average improvement of 30.65%, validating its effectiveness. Furthermore, we observe that discrete-based methods underperform compared to continuous-based methods because they neglect fine-grained temporal interactions within snapshots. Meanwhile, discrete-based methods encounter OOM error on the StackOverflow dataset, while several continuous-based methods face OOT errors, indicating scalability struggles with complex models like RNNs or memory networks. Notably, Graphmixer

**Table 1: The statistics of experimental dynamic graph datasets.**

| Datasets | # Nodes | # Edges | # Node/Edge features | # Unique Timestamps | # Time Steps | Description | Task | #Avg.Deg |
|---|---|---|---|---|---|---|---|---|
| UCI | 1,899 | 59,835 | -/- | 58,911 | 273 | Social | Link prediction | 31.51 |
| MOOC | 7,144 | 411,749 | –/4 | 345,600 | 100 | Interaction | Link prediction | 57.63 |
| BitcoinAlpha | 3,783 | 24,186 | -/2 | 1,647 | 226 | Transaction | Link prediction | 6.93 |
| LastFM | 1,980 | 1,293,103 | -/- | 1,283,614 | 500 | Interaction | Link prediction | 653.08 |
| UNvote | 201 | 1,035,742 | -/1 | 72 | 72 | Politics | Link prediction | 5152.94 |
| Reddit-title | 54,075 | 571,927 | 300/88 | 354,507 | 178 | Interaction | Link prediction | 10.57 |
| Enron | 184 | 125,235 | -/- | 22,632 | 100 | Social | Link prediction | 680.63 |
| Stackoverflow | 2,601,977 | 63,497,050 | -/- | 1,846,553 | 91 | Interaction | Link prediction | 24.20 |
| tgbl-trade | 255 | 468,245 | -/1 | 30 | 30 | Economics | Node affinity prediction | 1836.25 |
| tgbl-genre | 1,505 | 17,858,395 | -/1 | 4,187,046 | 1580 | Interaction | Node affinity prediction | 11,866.04 |
| tgbl-reddit | 11,766 | 27,174,118 | -/1 | 21,889,537 | 1090 | Social | Node affinity prediction | 2309.55 |
| tgbl-token | 61,756 | 72,936,998 | -/1 | 2,036,524 | 785 | Transaction | Node affinity prediction | 1181.05 |

**Table 2: Performance on dynamic link prediction. The best result is bold, the second best result is underlined.** OOM denotes that the method encountered an out-of-CUDA-memory issue, while OOT indicates that the training did not finish within 48 hours.

| Datasets | JODIE | EvolveGCN | ROLAND | DyRep | TGAT | TGN | GraphMixer | DyGFormer | ScaDyG |
|---|---|---|---|---|---|---|---|---|---|
| UCI | 0.402 ± 0.003 | 0.213 ± 0.009 | 0.419 ± 0.014 | 0.434 ±0.007 | 0.515 ± 0.016 | 0.463 ± 0.011 | 0.522 ± 0.008 | **0.601** ±0.031 | 0.534 ± 0.007 |
| MOOC | 0.356 ± 0.014 | 0.373 ± 0.016 | 0.756 ± 0.017 | 0.653 ±0.020 | 0.780 ± 0.007 | 0.855 ± 0.010 | 0.799 ± 0.016 | 0.730 ± 0.012 | **0.931** ± 0.009 |
| BitcoinAlpha | 0.332 ± 0.011 | 0.157 ± 0.022 | 0.232 ± 0.023 | 0.174 ± 0.005 | 0.319 ± 0.023 | 0.218 ± 0.006 | 0.224 ± 0.026 | 0.503 ± 0.030 | **0.597** ± 0.019 |
| LastFM | 0.098 ± 0.007 | 0.039 ± 0.006 | 0.053 ± 0.002 | 0.159 ±0.006 | 0.193 ± 0.012 | 0.042 ± 0.001 | 0.177 ± 0.009 | 0.236 ± 0.025 | **0.248** ± 0.012 |
| UNvote | 0.056 ± 0.019 | 0.405 ± 0.018 | 0.474 ± 0.010 | 0.446 ± 0.031 | 0.564 ± 0.018 | 0.596 ± 0.027 | 0.519 ± 0.016 | 0.657 ± 0.018 | **0.725** ± 0.009 |
| Reddit-title | 0.414 ± 0.019 | 0.354 ± 0.020 | 0.440 ± 0.036 | 0.537 ± 0.014 | 0.597 ± 0.011 | 0.628 ± 0.006 | 0.658 ± 0.015 | **0.706** ± 0.010 | 0.679 ± 0.013 |
| Enron | 0.407 ± 0.006 | 0.061 ± 0.009 | 0.065 ± 0.005 | 0.118 ± 0.007 | 0.147 ± 0.010 | 0.235 ± 0.002 | 0.224 ±0.008 | 0.433 ±0.017 | **0.481** ± 0.009 |
| Stackoverflow | 0.181 ± 0.016 | OOM | OOM | OOT | OOT | OOT | 0.231 ± 0.009 | 0.217 ± 0.018 | **0.559** ± 0.012 |

**Table 3: Performance on dynamic node affinity prediction. The best result is bold, the second best result is underlined.**

| Datasets | JODIE | EvolveGCN | ROLAND | DyRep | TGAT | TGN | GraphMixer | DyGFormer | ScaDyG |
|---|---|---|---|---|---|---|---|---|---|
| tgbl-trade | 0.373 ± 0.002 | 0.315 ± 0.011 | 0.439 ± 0.018 | 0.376 ± 0.005 | 0.448 ± 0.07 | 0.381 ± 0.006 | 0.366 ± 0.012 | 0.386 ± 0.008 | **0.631** ± 0.007 |
| tgbl-genre | 0.331 ± 0.007 | 0.305 ± 0.019 | 0.357 ± 0.013 | 0.337 ± 0.011 | 0.243± 0.009 | 0.369 ± 0.021 | 0.330 ± 0.015 | 0.259 ± 0.018 | **0.403** ± 0.005 |
| tgbl-reddit | 0.327 ± 0.012 | 0.257 ± 0.017 | 0.334 ± 0.016 | 0.309 ± 0.011 | 0.356 ± 0.014 | 0.329 ± 0.018 | 0.218 ± 0.015 | 0.305 ± 0.027 | **0.402** ± 0.012 |
| tgbl-token | 0.308 ± 0.010 | 0.234 ± 0.026 | 0.303 ± 0.018 | 0.159 ± 0.024 | 0.334 ± 0.021 | 0.185 ± 0.023 | 0.164 ± 0.022 | 0.251 ± 0.014 | **0.686** ± 0.019 |

and DyGformer perform poorly on the StackOverflow dataset, likely due to the large number of nodes introducing significant variability, which impairs their generalization ability and robustness.

**Node-level performance.** Subsequently, we use node affinity prediction to thoroughly evaluate the DGNNs' ability to capture dynamic relationships within node descriptions. Based on this, we report the overall performance in Table 3. According to the experimental results, ScaDyG achieves the best performance across all datasets, with an average improvement of 42.05%, highlighting its advantage in node-level inference. Unlike several prevalent baselines that prioritize link-level optimizations, ScaDyG avoids task-specific optimizations and instead focuses on modeling dynamic dependencies of temporal interactions. This makes ScaDyG a more general framework for various DG-based downstream tasks.

## 5.3 Scalability performance

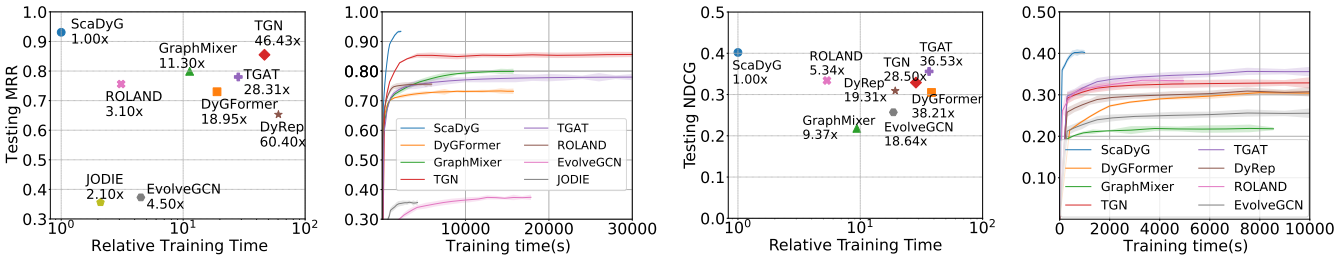
To answer **Q2**, we first provide a theoretical analysis of algorithm complexity in Table 4. Subsequently, Fig. 2 compares the relative training times and convergence speeds of ScaDyG with other competitive baselines in link- and node-level evaluations. Additionally, Table 5-6 report the pre-processing time (Pre), average training time per epoch (E-train), average testing time per epoch (E-test), GPU memory usage for the same batch size (G-Mem), and the number of parameters (Param), offering key insights for practical deployment.

**Complexity analysis.** We first analyze the time and space complexity of ScaDyG. In the pre-processing phase, ScaDyG propagates all edge features to nodes for $l$ hops, resulting in a time complexity of $O(lmf)$. During training, ScaDyG performs linear aggregation of historical steps of length $N$, followed by weight generation and feature transformation, both involving only linear transformations, leading to a time complexity of $O(nNLf^2)$. Regarding space complexity, ScaDyG needs to store model parameters and node features for a batch in GPU memory. Since we generate a parameter matrix for each node in a batch, the space complexity for storing these parameters is $O(bf^2)$. Additionally, the space complexity for storing intermediate messages for $L$ steps is $O(bLf)$. In summary, the total space complexity is $O(bLf + bf^2)$. Discrete-based methods encounter OOM (Out-Of-Memory) issues on the StackOverflow dataset due to the spatial complexity being assoicated to the number of nodes. Conversely, continuous-based methods, face OOT (Out-Of-Time) issues on this dataset because their link prediction complexity is related to the number of edges.

**Running efficiency.** Subsequently, we aim to provide a more comprehensive evaluation from the perspective of practical running efficiency. Specifically, we observe that ScaDyG is the most efficient method and give the following key insights. (1) Time cost: Although ScaDyG introduces pre-processing compared to baselines, this is negligible relative to the total training duration and is executed only once. Furthermore, ScaDyG shows superior performance in both total training time and training time per epoch, thanks to two key designs: (i) Feature propagation is handled during pre-processing, reducing training overhead; (ii) A computation-friendly

**Table 4: Time and space complexity analysis of ScaDyG and existing dynamic graph learning models.** $m$:edge numbers, $n$: node numbers, $f$: feature dim (including node & edge feature), $b$: the batch size, $N$: number of sampled neighbors for continuous-time based methods, and the number of historical steps in ScaDyG. $L$: number of total steps in ScaDyG and snapshots in EvolveGCN, $l$: layers of neural networks, and number of hops in ScaDyG. Link and node represent the complexity of the link and node-level tasks respectively.

| Type | Model | Pre-processing | Training/Inference (link) | Training/Inference (node) | Space |
|---|---|---|---|---|---|
| Discrete-time based | EvolveGCN | - | $O(Ll(n^2f+nf^2))$ | $O(Ll(n^2f+nf^2))$ | $O(nf+lf^2)$ |
| | ROLAND | - | $O(l(m+n)f^2)$ | $O(l(m+n)f^2)$ | $O(nlf+lf^2)$ |
| Continuous-time based | JODIE | - | $O(mlf^2)$ | $O(mlf^2)$ | $O((n+m)f+lf^2)$ |
| | DyRep | - | $O((n+mN)f^2)$ | $O(nf^2+mf^2+mN)$ | $O(nf+bNf+f^2)$ |
| | TGN | - | $O(mklNf^2)$ | $O(nklNf^2)$ | $O(nf+bNf+lf^2)$ |
| | TGAT | - | $O(mkN^lf^2)$ | $O(nkN^lf^2)$ | $O(bN^lf+lf^2)$ |
| | GraphMixer | - | $O(mNlf^2)$ | $O(nNlf^2)$ | $O(bNf+f^2)$ |
| | DyGFormer | - | $O(mkNlf^2)$ | $O(nkNlf^2)$ | $O(bNf+lf^2)$ |
| Decoupled based | ScaDyG | $O(lmf)$ | $O(nNLf^2)$ | $O(nNLf^2)$ | $O(bLf+bf^2)$ |




**(a) Training time on MOOC** **(b) Training time on MOOC** **(c) Training time on tgbl-reddit** **(d) Training time on tgbl-reddit**

**Figure 2: Efficiency comparison of ScaDyG and baselines.**

**Table 5: Epoch and batch efficiency on MOOC dataset.**

| Method | Pre | E-train | E-eval | G-Mem | Param |
|---|---|---|---|---|---|
| EvolveGCN | - | 363.78s | 97.44s | 4573M | 2578K |
| ROLAND | - | 191.21s | 43.53s | 3372M | 383K |
| JODIE | - | 156s | 60.13s | 1326M | 311K |
| DyRep | - | 2676.31s | 528.45s | 4600M | 1188K |
| TGAT | - | 2413.27s | 575.53s | 3526M | 1052K |
| TGN | - | 4479.01s | 993.32s | 4188M | 1460K |
| GraphMixer | - | 1128.64s | 124.41s | 2958M | 643K |
| DyGFormer | - | 2152.91s | 337.34s | 1942M | 976K |
| ScaDyG | 21.13s | 126.43s | 53.78s | 1136M | 55K |

**Table 6: Epoch and batch efficiency on tgbl-reddit dataset.**

| Method | Pre | E-train | E-eval | G-Mem | Param |
|---|---|---|---|---|---|
| EvolveGCN | - | 991.96s | 1008.24s | 22450M | 1018K |
| ROLAND | - | 256.29s | 247.64s | 18970M | 527K |
| JODIE | - | 1057s | 798.5s | 14942M | 132K |
| DyRep | - | 974.51s | 1492.29s | 15556M | 244K |
| TGAT | - | 1350.25s | 1441.36s | 15212M | 712K |
| TGN | - | 1272s | 1466.58s | 15784M | 947K |
| GraphMixer | - | 413.72s | 1215.76s | 14484M | 381K |
| DyGFormer | - | 1966.34s | 939.55s | 15200M | 1067K |
| ScaDyG | 14.36s | 50.78s | 49.68s | 6178M | 182K |

neural architecture is simple yet effective with fewer parameters and faster convergence. As shown in the complexity analysis in Table 4, ScaDyG's efficient training time complexity stems from its minimal dependence on the number of edges, with edge-related operations limited to straightforward feature matrix transformations.

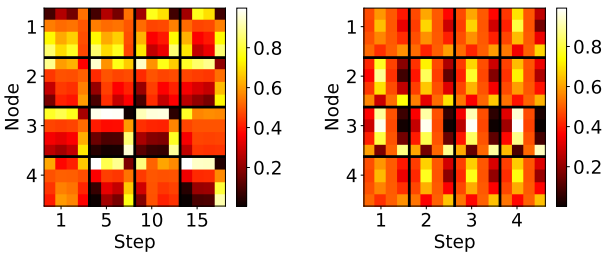
**Table 7: Results of ablation experiments.**

| Datasets | ScaDyG | w/o T | w/o HN | w/o TE |
|---|---|---|---|---|
| UCI | 0.534 ± 0.007 | 0.485 ± 0.009 | 0.521 ± 0.013 | 0.470 ± 0.007 |
| MOOC | 0.931 ± 0.009 | 0.473 ± 0.009 | 0.785 ± 0.011 | 0.794 ± 0.016 |
| BitcoinAlpha | 0.597 ± 0.019 | 0.561 ± 0.016 | 0.567 ± 0.013 | 0.575 ± 0.010 |
| LastFM | 0.248 ± 0.012 | 0.225 ± 0.008 | 0.229 ± 0.012 | 0.231 ± 0.011 |
| UNvote | 0.725 ± 0.009 | 0.675 ± 0.012 | 0.697 ± 0.006 | 0.652 ± 0.011 |
| Reddit-title | 0.679 ± 0.013 | 0.462 ± 0.021 | 0.579 ± 0.009 | 0.584 ± 0.006 |
| Enron | 0.481 ± 0.009 | 0.381 ± 0.010 | 0.413 ± 0.009 | 0.401 ± 0.005 |
| Stackoverflow | 0.559 ± 0.012 | 0.471 ± 0.014 | 0.501 ± 0.009 | 0.446 ± 0.008 |
| tgbl-trade | 0.631 ± 0.007 | 0.375 ± 0.006 | 0.390 ± 0.007 | 0.383 ± 0.019 |
| tgbl-genre | 0.403 ± 0.005 | 0.294 ± 0.009 | 0.357 ± 0.015 | 0.335 ± 0.012 |
| tgbl-reddit | 0.402 ± 0.012 | 0.303 ± 0.011 | 0.356 ± 0.012 | 0.327 ± 0.010 |
| tgbl-token | 0.686 ± 0.019 | 0.578 ± 0.021 | 0.639 ± 0.011 | 0.623 ± 0.014 |

(2) Memory cost: ScaDyG requires fewer trainable parameters, reducing the storage needed for parameter gradients and model states, thereby lowering GPU memory usage. Notably, ScaDyG needs additional storage for propagated features within pre-process, which are stored in main memory, avoiding extra GPU memory consumption. Further analysis of baseline results is provided in Appendix B.5.

## 5.4 Interpretability Analysis

To answer **Q3**, we first conduct an ablation study to verify three key components of our proposed ScaDyG, followed by the visualization analysis of the weights of the transformation matrix generated by the hypernetwork in inference stage. Specifically, in the ablation study, "w/o T" represents ScaDyG without temporal encoding, where the sum of the historical step node features through the hypernetwork is used as the final node embedding. "w/o HN" represents ScaDyG without Hypernetwork-driven message aggregation,




(a) On MOOC dataset. (b) On tgbl-trade dataset.

**Figure 3: Visualization of transformation matrix.**

using $\mathbf{X}_t$ directly for downstream prediction. "w/o TE" involves replacing the combination of exponential functions with a traditional exponential time modeling [43, 61]. We repeat the single exponential function $d_e$ times as the temporal encoding vector. For visualization analysis, we display the top-left 4x4 submatrix of the transformation matrix of different nodes at different steps.

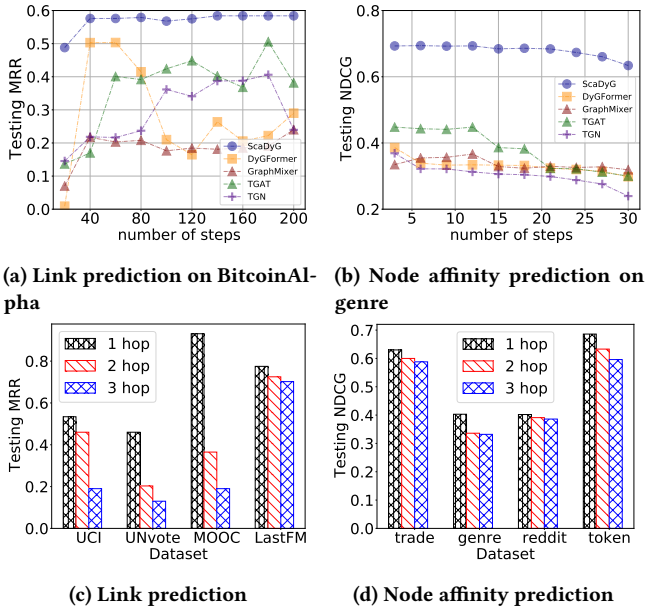
**Result of ablation experiment.** From Table 7, we have the following observations: (1) The most significant performance degradation occurs in the "w/o TE" model, underscoring the importance of dynamic dependencies in historical interactions; (2) The removal of either Hypernetwork-driven Message Aggregation (w/o HN) or the combination of exponential functions (w/o TE) leads to substantial performance drops, indicating that the hypernetwork is crucial for dynamic encoding, and the proposed time encoding outperforms traditional exponential time modeling. The absence of either compromises the node-wise dynamic temporal modeling capability, resulting in insufficient representation of temporal dependencies.

**Visualization Analysis.** Based on Fig. 3, we observe that different nodes exhibit distinct weight patterns, and the weights of the same node gradually evolve over different steps. Specifically, in the MOOC dataset, the weights vary more significantly between nodes and even within the same node (e.g., nodes 3 and 4) across various steps. This variation indicates diverse patterns between nodes and significant changes in the graph structure between steps. In contrast, in the tgbl-trade dataset, the patterns are more similar between different nodes. Within the same step, the weights show a clearer gradual change, reflecting the temporal evolution trend in the dynamic graph. These results further validate the effectiveness of Hypernetwork-driven Message Aggregation in dynamically modeling node-wise temporal patterns.

## 5.5 Hyperparameter Sensitivity

To answer **Q4**, we explore two key performance parameters: historical steps and neighborhood hops. Specifically, historical steps determine the length of the information window, with more steps capturing long-range dependencies and fewer steps focusing on short-range ones. Neighborhood hops define the range of neighborhood information used, with single hops focusing on direct interactions and multiple hops integrating broader propagation. The experimental results are shown in Figure. 4(a)-(b) and (c)-(d).

**Historical steps.** Experimental results show that ScaDyG is less sensitive to the number of steps compared to baselines. Specifically, in the link prediction task on the BitcoinAlpha, ScaDyG's optimal performance is observed with parameters in the 160-200




(a) Link prediction on BitcoinAlpha (b) Node affinity prediction on genre

(c) Link prediction (d) Node affinity prediction

**Figure 4: Results of hyperparameter study.**

range, emphasizing the importance of long-range dependencies in this dataset. In contrast, the optimal parameters for baselines vary inconsistently, suggesting that their performances are affected by the randomness of neighborhood sampling. For the node affinity prediction task on the tgbl-genre, most methods perform best with a time step of 3, with MRR decreasing as the number of time steps increases. This decline may be attributed to the presence of noise in long-range temporal information within the tgbl-trade.

**Neighborhood hops.** According to the experimental results shown in Fig. 4(c)-(d), we observe that for both link prediction and node affinity prediction, performance is optimal at 1-hop and significantly declines with multiple hops. This suggests that incorporating multi-hop information can be detrimental to TG tasks, contrasting with results on simple static graphs [7, 23]. One possible explanation is that multi-hop interactions on DGs often introduce much noise, potentially degrading the model's predictive performance.

## 6 CONCLUSION

In this study, we propose ScaDyG, an efficient and effective framework for learning on large-scale dynamic graphs. By reformulating the message passing process of dynamic GNNs within a decoupled framework, ScaDyG achieves superior time and space efficiency compared to existing methods, making it particularly well-suited for modeling million-level dynamic graphs. Unlike previous approaches that rely on sequence-based models for temporal modeling, ScaDyG employs a simple yet powerful time encoding scheme based on a dynamic fusion of exponential functions. To extend temporal modeling to node-level granularity, the preprocessed temporal messages are aggregated through a hypernetwork-driven transformation. Interestingly, we observe diverse temporal patterns in the generated weights. The designs not only reduce the number of parameters and training time but also enable ScaDyG to deliver competitive results across a diverse range of dynamic graphs. For future work, we will explore broader applications of scalable frameworks on a large scale, e.g., billion-level dynamic graph learning.

# REFERENCES


[1] Sudhanshu Chanpuriya, Ryan A Rossi, Sungchul Kim, Tong Yu, Jane Hoffswell, Nedim Lipka, Shunan Guo, and Cameron N Musco. 2022. Direct embedding of temporal network edges via time-decayed line graphs. In *The Eleventh International Conference on Learning Representations*.

[2] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[3] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems* 33 (2020), 14556–14566.

[4] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 257–266.

[5] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. 2023. Do We Really Need Complicated Model Architectures For Temporal Networks? *arXiv preprint arXiv:2302.11636* (2023).

[6] Wenzheng Feng, Yuxiao Dong, Tinglin Huang, Ziqi Yin, Xu Cheng, Evgeny Kharlamov, and Jie Tang. 2022. Grand+: Scalable graph random neural networks. In *Proceedings of the ACM Web Conference 2022*. 3248–3258.

[7] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[8] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. *Advances in neural information processing systems, NeurIPS* (2019).

[9] David Ha, Andrew M Dai, and Quoc V Le. 2016. HyperNetworks. In *International Conference on Learning Representations*.

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

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

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

[13] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*.

[14] Keke Huang, Jing Tang, Juncheng Liu, Renchi Yang, and Xiaokui Xiao. 2023. Node-wise diffusion for scalable graph learning. In *Proceedings of the ACM Web Conference 2023*. 1723–1733.

[15] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2024. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems* 36 (2024).

[16] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems* 31 (2018).

[17] Ming Jin, Yuan-Fang Li, and Shirui Pan. 2022. Neural Temporal Walks: Motif-Aware Representation Learning on Continuous-Time Dynamic Graphs. In *Advances in Neural Information Processing Systems*.

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

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

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

[21] 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*. 1269–1278.

[22] Xunkai Li, Meihao Liao, Zhengyu Wu, Daohan Su, Wentao Zhang, Rong-Hua Li, and Guoren Wang. 2024. LightDiC: A Simple yet Effective Approach for Large-scale Digraph Representation Learning. *arXiv preprint arXiv:2401.11772* (2024).

[23] Xunkai Li, Jingyuan Ma, Zhengyu Wu, Daohan Su, Wentao Zhang, Rong-Hua Li, and Guoren Wang. 2024. Rethinking Node-wise Propagation for Large-scale Graph Learning. In *Proceedings of the ACM Web Conference, WWW*.

[24] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Orca: Scalable Temporal Graph Neural Network Training with Theoretical Guarantees. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[25] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1332–1345.

[26] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. 2022. SCARA: scalable graph neural networks with feature-oriented optimization. *arXiv preprint arXiv:2207.09179* (2022).

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

[28] Yuhong Luo and Pan Li. 2022. Neighborhood-aware scalable temporal network representation learning. In *Learning on Graphs Conference*. PMLR, 1–1.

[29] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. 2009. 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 (2009), 911–932.

[30] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. 601–610.

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

[32] Hao Peng, Hongfei Wang, Bowen Du, Md Zakirul Alam Bhuiyan, Hongyuan Ma, Jianwei Liu, Lihong Wang, Zeyu Yang, Linfeng Du, Senzhang Wang, et al. 2020. Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting. *Information Sciences* 521 (2020), 277–290.

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

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

[35] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*. 519–527.

[36] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *International conference on neural information processing*. Springer, 362–373.

[37] Li Sun, Zhongbao Zhang, Jiawei Zhang, Feiyang Wang, Hao Peng, Sen Su, and S Yu Philip. 2021. Hyperbolic variational graph neural network for modeling dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4375–4383.

[38] Haoran Tang, Shiqing Wu, Guandong Xu, and Qing Li. 2023. Dynamic graph evolution learning for recommendation. In *Proceedings of the 46th international acm sigir conference on research and development in information retrieval*. 1589–1598.

[39] Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. 2020. Hypergrid transformers: Towards a single model for multiple tasks. In *International conference on learning representations*.

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

[41] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

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

[43] Zhihao Wen and Yuan Fang. 2022. TREND: TempoRal Event and Node Dynamics for Graph Representation Learning. In *Proceedings of the ACM Web Conference 2022*. 1159–1169.

[44] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[45] Yuxia Wu, Yuan Fang, and Lizi Liao. 2024. On the Feasibility of Simple Transformer for Dynamic Graph Modeling. In *Proceedings of the ACM on Web Conference 2024*. 870–880.

[46] Sheng Xiang, Dawei Cheng, Chencheng Shang, Ying Zhang, and Yuqi Liang. 2022. Temporal and heterogeneous graph neural network for financial time series prediction. In *Proceedings of the 31st ACM international conference on information & knowledge management*. 3584–3593.

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

[48] Menglin Yang, Min Zhou, Marcus Kalander, Zengfeng Huang, and Irwin King. 2021. 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*. 1975–1985.

[49] Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. 2023. Simple and efficient heterogeneous graph neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 10816–10824.

[50] Jiaxuan You, Tianyu Du, and Jure Leskovec. 2022. ROLAND: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2358–2366.

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

[52] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[53] Chun-Yang Zhang, Zhi-Liang Yao, Hong-Yu Yao, Feng Huang, and CL Philip Chen. 2022. Dynamic Representation Learning via Recurrent Graph Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2022).

[54] Wentao Zhang, Mingyu Yang, Zeang Sheng, Yang Li, Wen Ouyang, Yangyu Tao, Zhi Yang, and Bin Cui. 2021. Node dependent local smoothing for scalable graph learning. *Advances in Neural Information Processing Systems* 34 (2021), 20321–20332.

[55] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2022. Graph attention multi-layer perceptron. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4560–4570.

[56] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. TGL: a general framework for temporal GNN training on billion-scale graphs. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1572–1580.

[57] Ke Zhou, Hongyuan Zha, and Le Song. 2013. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Artificial Intelligence and Statistics*. PMLR, 641–649.

[58] Hao Zhu and Piotr Koniusz. 2020. Simple spectral graph convolution. In *International conference on learning representations*.

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

[60] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems* 32 (2019).

[61] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2857–2866.

# A PROOF OF PROPOSITION 1

Assuming the initial edge feature of $(u, v, t')$ is $\mathbf{x}_{e_v} = [x_1, x_2, ..., x_d]$, the time feature is $[e^{\gamma_1 t}, e^{\gamma_2 t}, ..., e^{\gamma_d t}]$. let $\Delta t = \Delta t_1 + \Delta t_2$, $\mathbf{x}_{e_v} \odot T_e(\Delta t_1) \odot T_e(\Delta t_2)\mathbf{W}_1$ is ,

$$\begin{aligned}
&\mathbf{x}_{e_v} \odot T_e(\Delta t_1) \odot T_e(\Delta t_2)\mathbf{W}_1 \\
&= [x_1 e^{\gamma_1 \Delta t_1} e^{\gamma_1 \Delta t_2}, x_2 e^{\gamma_2 \Delta t_1} e^{\gamma_2 \Delta t_2}, \ldots, x_d e^{\gamma_d \Delta t_1} e^{\gamma_d \Delta t_2}]\mathbf{W}_1 \\
&= [x_1 e^{\gamma_1 (\Delta t)}, x_2 e^{\gamma_2 (\Delta t)}, \ldots, x_d e^{\gamma_d (\Delta t)}]\mathbf{W}_1.
\end{aligned} \tag{6}$$

Let $\mathbf{W}_1 \in \mathbb{R}^{d \times d}$ be expressed as:

$$\mathbf{W_1} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dd} \end{bmatrix} \tag{7}$$

thus,

$$\begin{aligned}
&\mathbf{x}_{e_v} \odot T_e(\Delta t_1) \odot T_e(\Delta t_2)\mathbf{W}_1 \\
&= \begin{bmatrix} w_{11}x_1 e^{\gamma_1 \Delta t} + w_{12}x_2 e^{\gamma_2 \Delta t} + \cdots + w_{1d}x_d e^{\gamma_d \Delta t} \\ w_{21}x_1 e^{\gamma_1 \Delta t} + w_{22}x_2 e^{\gamma_2 \Delta t} + \cdots + w_{2d}x_d e^{\gamma_d \Delta t} \\ \vdots \\ w_{d1}x_1 e^{\gamma_1 \Delta t} + w_{d2}x_2 e^{\gamma_2 \Delta t} + \cdots + w_{dd}x_d e^{\gamma_d \Delta t} \end{bmatrix}^T
\end{aligned} \tag{8}$$

On the other hand,

Let $\mathbf{W} \in \mathbb{R}^{d \times d}$ be expressed as:

$$\mathbf{W} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1d} \\ z_{21} & z_{22} & \cdots & z_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ z_{d1} & z_{d2} & \cdots & z_{dd} \end{bmatrix}, \tag{9}$$

then,

$$\begin{aligned}
&\mathbf{x}_{e_v}\kappa(\Delta t_1 + \Delta t_2))\mathbf{W} \\
&= \mathbf{x}_{e_v}\kappa(\Delta t))\mathbf{W} \\
&= \mathbf{x}_{e_v}(a_1 e^{\gamma_1 \Delta t} + a_2 e^{\gamma_2 \Delta t} + \cdots + a_d e^{\gamma_\Delta t})\mathbf{W} \\
&= \sum_{i=1}^{d} a_i e^{\gamma_i (\Delta t)} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}^T \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1d} \\ z_{21} & z_{22} & \cdots & z_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ z_{d1} & z_{d2} & \cdots & z_{dd} \end{bmatrix} \\
&= \begin{bmatrix} z_{11}x_1 + z_{12}x_2 + \cdots + z_{1d}x_d \\ z_{21}x_1 + z_{22}x_2 + \cdots + z_{2d}x_d \\ \vdots \\ z_{d1}x_1 + z_{d2}x_2 + \cdots + z_{dd}x_d \end{bmatrix}^T \sum_{i=1}^{d} a_i e^{\gamma_i (\Delta t)}
\end{aligned} \tag{10}$$

By assigning $w_{ij} = z_{ij}(a_1 e^{(\gamma_1 - \gamma_j)\Delta t} + a_2 e^{(\gamma_2 - \gamma_j)\Delta t} + \cdots + a_n e^{(\gamma_d - \gamma_j)\Delta t})$, $i, j = 1, 2, ..., d$ the proposition is proved.

# B ADDITIONAL EXPERIMENTS

## B.1 Additional experimental settings

**Implementation settings.** The hyperparameters of ScaDyG mainly include the learning rate, the parameter $\{\gamma_i\}_{i=1}^{d_e}$ for the combination of exponentials and the number of historical steps. The learning rate is searched from $\{10^{-5}, 10^{-1}\}$, and $n$ is searched from 1 to the total time steps in the specific dataset. The parameters of the exponential functions in time encoding,i.e. $\{\gamma_i\}_{i=1}^{d_e}$, can be any combination of different values. We set them as a uniformly decreasing sequence for convenience, i.e., $\gamma_i = \gamma_0 - id$, $d = (\gamma_{d_0} - \gamma_{d_e})/d_e$ and $\gamma_{d_e} = 10^{-1}\gamma_0$. Since different datasets have varying time ranges, we found that setting $\gamma_0$ to be around the inverse of the dataset's time range works best. For datasets that do not contain edge features, we follow [50] to generate a one-dimensional feature based on the edge timestamps. For datasets with only one-dimensional edge features, we replicate the edge features into 8-dimensional vectors to facilitate dynamic feature encoding. For dynamic link prediction tasks, the methods are evaluated under the ranking setting, as it can avoid the bias of easy negatives compared to binary classification [15]. However, we also provide the results under binary classification in Appendix B.4 for reference. For baselines, we use the library proposed by [51] for the implementation of the five continuous-based methods, i.e, DyGFormer, GraphMixer, TGAT, TGN, JODIE, and DyRep. ROLAND and EvolveGCN are adopted from their official implementation. The same node and edge feature dimensions are set equal to ScaDyG. All experiments are conducted on a Linux server with an Intel Xeon Gold 5218R CPU and an NVIDIA RTX 3090 with 24GB memory.

**Evaluation metrics.** For dynamic link prediction task, we employ a multi-layer perceptron to predict the presence of an edge between two nodes, using the concatenated embeddings of these nodes as input. The methods are evaluated under the ranking setting, where each positive sample is compared against 100 negative samples obtained through random sampling (with collision check). Compared to the binary classification, the ranking setting can avoid the bias of easy negatives, thus making the results more reliable [15]. MRR (Mean Reciprocal Rank) is selected as the metric to evaluate the ranking quality. For node affinity prediction, we use a multi-layer perceptron to predict the affinity vector based on the node representation, and we follow [15] use NDCG (Normalized Discounted Cumulative Gain) as the evaluation metric.

**Additional background for tasks.** Most existing methods focus on link prediction tasks [5, 34, 43, 47, 50], with only a few considering node classification [27, 31, 51]. This is mainly due to the lack of labeled data for nodes in most temporal graph datasets. The few datasets with node labels also suffer from two significant limitations: 1) Their labels are binary (0 or 1), which does not allow for a comprehensive evaluation of node-level performance. 2) Their labels do not change over time, failing to assess the model's ability to capture temporal dynamics. Therefore, we did not use node classification tasks but instead adopted a newly introduced node affinity prediction task. This task's datasets avoid the mentioned limitations and are of a larger scale. We provide a detailed introduction to these datasets in Appendix B.2.

## B.2 Description of datasets

**Dataset for dynamic link prediction.** In this section, we give a brief introduction to the datasets. For dynamic link prediction, we conducted experiments on 8 public datasets, including UCI, MOOC, UNvote, BitcoinAlpha, Stackoverflow, Reddit-title, and Enron. These datasets are uniformly segmented into time steps based

on predefined intervals. The time steps are split chronologically with a ratio of 70%/15%15% for training, validation, and testing.

- **UCI** [29, 33] is a social network, providing records of interaction among students from the University of California at Irvine. Each interaction $(u, v, t)$ characterizes a message between two users at a timestamp down to the second.
- **MOOC** [21, 33] constitutes a network of student interactions derived from components of online courses, including problem sets and video materials. Each interaction $(u, v, t)$ represents a student engaging with a piece of content, characterized by four distinct attributes.
- **LastFM** [21, 33] is a dataset from the online music platform Last.fm, each interaction $(u, v, t)$ in the dataset represents user $u$ listening to a song $v$ at time $t$. The dataset encompasses the listening activities of 1000 users with respect to the top 1000 songs over a timeframe of one month.
- **Bitcoinalpha** [19, 20] represents a trust-based network of bitcoin users engaged in transactions via the Alpha platform. Each interaction $(u, v, t)$ denotes a rating from one user to another.
- **UNvote** [33] tracks the roll-call votes in the United Nations General Assembly. Each interaction represents a joint voting behavior between two nations. The weight of the link between them corresponds to the number of joint votes in a year.
- **Reddit-title** [18] is constructed based on the hyperlink connections between subreddits on the Reddit platform. Each temporal edge represents a hyperlink in the title of a post from one subreddit to another. The timestamp associated with the edge is the creation time of the post.
- **Enron** [33] encompasses email communications among employees within the Enron company. Each interaction $(u, v, t)$ represents an email sent from $u$ to $v$.
- **Stackoverflow** [30] captures user interactions within the Stack Overflow community, where nodes correspond to individual users and each interaction $(u, v, t)$ indicates the act of one u providing an answer to v's inquiry at $t$.

**Datasets for dynamic node affinity prediction.** For dynamic node affinity prediction task, we adopt 4 datasets provided by [15], encompassing tgbl-genre, tgbl-trade, tgbl-reddit and tgbl-token. We chose these four datasets because the node affinity labels in the datasets are multi-dimensional and associated with time steps, allowing a more comprehensive evaluation than binary or static labels [15]. The node affinity labels in the datasets are associated with time steps. Accordingly, we divide the training, validation, and test set according to the time steps in chronological order at a ratio of 70%/15%15%. We give a brief introduction of the dataset as follows.

- **tgbl-trade** [15] records the international agriculture trade network among United Nations member countries from 1986 to 2016. In this network, each temporal edge $(u, v, t)$ represents a trading relation between country $u$ to another country $v$. The edge value signifies the total value of agricultural products traded in one year. The node affinity represents the percentage distribution of annual trade products from a specific country to other countries.
- **tgbl-genre** [15] represents a user-item bipartite network capturing interactions between users and the music genres they prefer. Each temporal edge $(u, v, t)$ delineates a user $u$ has listened to a song of a specific genre $v$ at a particular time $t$. The edge weight signifies the degree of affiliation of the song to this genre, measured in percentages. For a user node, the property is the frequency of interactions between the user and music across all genres.
- **tgbl-reddit** [15] represents the interaction between users and subreddits in the Reddit platform. Each interaction $(u, v, t)$ represents the user made a post on the subreddit. The property of a user node delineates the frequency of its interactions with various subreddits in a week.
- **tgbl-token** [15] is a network that maps user interactions with cryptocurrency tokens. A temporal edge between a user node and a token node signifies a transaction made by the user to acquire that specific token, and the weight of the edge represents the quantity of the token transferred in that transaction. The node affinity is the frequency of interactions that a user has with various types of cryptocurrency tokens over a period of one week.

## B.3 Description of baselines

In this section, we give a brief introduction to the baselines used in comparison experiments.

- **DyGFormer** [51] introduces a Transformer-based architecture for dynamic graph learning that leverages historical interactions through neighbor co-occurrence encodings and a patching technique for long-sequence processing.
- **GraphMixer** [21, 33] aims to reduce the complexity of temporal graph modeling with a straightforward yet effective architecture. It leverages MLP-based link encoders, mean-pooling for node information, and an MLP-based link classifier. The timestamps are projected by a static cosine-based encoding function.
- **TGAT** [47] synthesizes temporal-topological neighborhood features and time-feature interactions through self-attention and a novel time encoding to generate dynamic node embeddings for temporal graphs.
- **TGN** [34] utilized a memory module to the dynamic state involving a node, which is updated on observing a new interaction. The current embedding is then obtained by aggregating the state and the messages received from neighboring nodes.
- **DyRep** [40] leverages two processes, namely the communication and association.It employs a time-scale adaptive multivariate point process model to capture the evolution of dynamic graphs.
- **ROLAND** [50] aims to repurpose state-of-the-art static GNN architectures. It treats node embeddings as evolving states that are hierarchically updated in a recurrent fashion over time. Furthermore, ROLAND features a live-update setting where predictions are made continuously, and the model is updated incrementally. It propose three variants namely ROLAND-Moving Average, ROLAND-MLP

**Table 8: Performance on binary classification setting (AP).**

| Method | MOOC | Bitcoinalpha | Reddit-title | UCI |
|---|---|---|---|---|
| EvolveGCN | 0.728 ± 0.016 | 0.959 ± 0.003 | 0.944 ± 0.003 | 0.808 ± 0.006 |
| ROLAND | 0.874 ± 0.016 | 0.983 ± 0.002 | 0.980 ± 0.003 | 0.894 ± 0.005 |
| JODIE | 0.805 ± 0.026 | 0.984 ± 0.001 | 0.991 ± 0.001 | 0.894 ±0.010 |
| DyRep | 0.820 ±0.006 | 0.901 ± 0.001 | 0.984 ± 0.002 | 0.889 ± 0.003 |
| TGAT | 0.856 ± 0.002 | 0.991 ± 0.001 | 0.990 ± 0.001 | 0.799 ± 0.009 |
| TGN | 0.890 ± 0.018 | 0.992 ± 0.002 | 0.989 ± 0.001 | 0.929 ± 0.015 |
| GraphMixer | 0.828 ± 0.003 | 0.993 ± 0.001 | 0.991 ± 0.001 | 0.931 ± 0.007 |
| DyGFormer | 0.873 ± 0.005 | 0.993 ± 0.001 | 0.991 ± 0.001 | 0.953 ± 0.002 |
| ScaDyG | **0.901** ± 0.012 | **0.996** ± 0.001 | **0.992** ± 0.002 | **0.956** ± 0.002 |

**Table 9: Performance on binary classification setting (AUC).**

| Method | MOOC | Bitcoinalpha | Reddit-title | UCI |
|---|---|---|---|---|
| EvolveGCN | 0.808 ± 0.005 | 0.955 ± 0.001 | 0.947 ± 0.003 | 0.802 ± 0.008 |
| ROLAND | 0.854 ± 0.013 | 0.986 ± 0.001 | 0.981 ± 0.003 | 0.892 ± 0.005 |
| JODIE | 0.828 ± 0.009 | 0.988 ± 0.002 | 0.991 ± 0.001 | 0.905 ± 0.009 |
| DyRep | 0.811 ± 0.002 | 0.857 ± 0.001 | 0.982 ± 0.001 | 0.891 ± 0.002 |
| TGAT | 0.871 ± 0.002 | 0.992 ± 0.001 | 0.989 ± 0.001 | 0.785 ± 0.008 |
| TGN | 0.913 ± 0.002 | 0.993 ± 0.001 | 0.991 ± 0.001 | 0.921 ± 0.002 |
| GraphMixer | 0.840 ± 0.002 | 0.994 ± 0.001 | 0.990 ± 0.001 | 0.920 ± 0.005 |
| DyGFormer | 0.879 ± 0.006 | 0.993± 0.001 | 0.990 ± 0.001 | 0.946 ± 0.001 |
| ScaDyG | **0.931** ± 0.009 | **0.996** ± 0.001 | **0.993** ± 0.002 | **0.955** ± 0.001 |

and ROLAND-GRU. We select the ROLAND-GRU as our baseline because its best reported performance.

- **EvolveGCN** [31] adapts graph convolutional network (GCN) models to the temporal dimension without depending on static node embeddings. It employs RNNs to evolve the GCN parameters dynamically to future time steps.
- **JODIE** [21] learns both static and dynamic user embeddings for users-items interaction graphs, representing their long-term and time-varying properties, respectively. It leverages coupled recurrent neural networks (RNNs) to update embeddings based on user-item interaction.

## B.4 Additional experimental results

**Performance on binary classification.** Besides the ranking setting, we present comparative results for binary classification, which predicts which of two candidate nodes a given node will form an edge at a specific timestamp. We use AP and AUC as the evaluation metrics. The results are shown in table 8 and 9, which show that ScaDyG still achieves the best performance. Additionally, on Bitcoinalpha and Reddit-title, most methods yield similar outcomes, making MRR a better metric for distinguishing between them.

**Hyperparameter analysis on intervals.** We conduct experiments to evaluate ScaDyG's behavior under different intervals between steps (Note that steps = time range/interval). The experiments are conducted only in link prediction, as the intervals for node affinity prediction are predetermined by the dataset. As shown in figure 5, MRR across all datasets does not significantly decrease with increasing step intervals, indicating ScaDyG effectively preserves information between steps. On the Lastfm and Bitcoinalpha datasets, smaller intervals result in a drop in MRR, mainly because of the limitation of the information window, given the large number of steps. Moreover, extremely small intervals are not preferred practically due to scalability considerations.

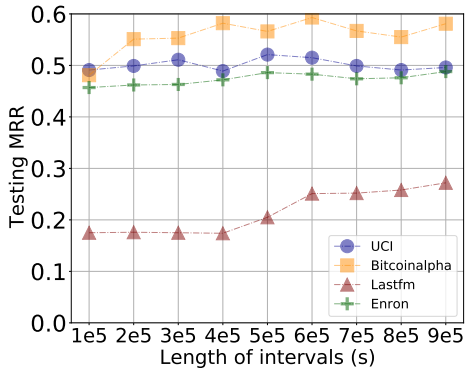



**Figure 5: Hyperparameter analysis on intervals.**

## B.5 Additional analysis on results.

**Baseline performance.** Continuous-time-based methods have the highest complexity related to the number of edges due to their intricate designs for handling the neighborhood sequences of nodes associated with each edge. Since dynamic graphs typically have a significantly larger number of edges than nodes, this results in substantial time overhead. In contrast, discrete-time-based methods exhibit lower complexity as their training time is primarily associated with node-related or simpler edge-related operations, thus exhibit faster training speed compared to continuous-time methods in experiments. Discrete-time-based and memory module-based methods exhibit node-related spatial complexity as they need to store the state of all nodes, leading to higher GPU memory usage compared to other methods in experiments.

## B.6 Relation to existing time encodings.

Existing time modeling techniques mainly fall into two categories: trigonometric-based encoding [5, 47] and exponential-based techniques [27, 43, 61]. The former vectorizes timestamps using functions with different parameters, and models time intervals through the dot product of time vectors. While these techniques are expressive, they require learnable frequencies, and thus cannot be used in preprocessing. Exponential-based techniques, especially point processes-based methods, model the decay of historical interactions over time using a single exponential function. Our approach belongs to exponential-based techniques, but instead of using a single function, we use a combination of exponentials to enable dynamic time modeling. The various exponentials in a time encoding are set with predefined parameters, making it convenient for preprocessing. During the learning phase, the exponential functions are dynamically combined, offering greater representational power than a single exponential function.