On the Feasibility of Simple Transformer for Dynamic Graph Modeling

Yuxia Wu
Singapore Management University
Singapore
yuxiawu@smu.edu.sg

Yuan Fang*
Singapore Management University
Singapore
yfang@smu.edu.sg

Lizi Liao
Singapore Management University
Singapore
lzliao@smu.edu.sg

ABSTRACT

Dynamic graph modeling is crucial for understanding complex structures in web graphs, spanning applications in social networks, recommender systems, and more. Most existing methods primarily emphasize structural dependencies and their temporal changes. However, these approaches often overlook detailed temporal aspects or struggle with long-term dependencies. Furthermore, many solutions overly complicate the process by emphasizing intricate module designs to capture dynamic evolutions. In this work, we harness the strength of the Transformer's self-attention mechanism, known for adeptly handling long-range dependencies in sequence modeling. Our approach offers a simple Transformer model, called SimpleDyG, tailored for dynamic graph modeling without complex modifications. We re-conceptualize dynamic graphs as a sequence modeling challenge and introduce a novel temporal alignment technique. This technique not only captures the inherent temporal evolution patterns within dynamic graphs but also streamlines the modeling process of their evolution. To evaluate the efficacy of SimpleDyG, we conduct extensive experiments on four real-world datasets from various domains. The results demonstrate the competitive performance of SimpleDyG in comparison to a series of state-of-the-art approaches despite its simple design.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; • **Information systems** → **Data mining**; **World Wide Web**.

KEYWORDS

Dynamic graphs, Transformer, graph representation learning

ACM Reference Format:
Yuxia Wu, Yuan Fang*, and Lizi Liao. 2024. On the Feasibility of Simple Transformer for Dynamic Graph Modeling. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3589334.3645622

* Corresponding author.

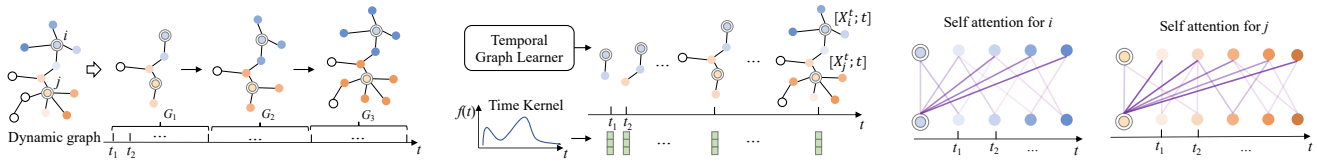
WWW '24, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0171-9/24/05…$15.00
https://doi.org/10.1145/3589334.3645622

1 INTRODUCTION

Graph-structured data are prevalent on the World Wide Web [52], such as social networks [9, 33], recommender systems [45, 47], citation graphs [15, 53, 54], dialogue systems [24, 46], and so on. Thus, graph-based mining and learning have become fundamental tools in many Web applications, ranging from analyzing communication patterns within social friendships, to predicting users' behavior on digital platforms and investigating citation trends in the academic community. Traditionally, many works focus on static graphs characterized by fixed nodes and edges. However, many real-world graphs on the Web are intrinsically dynamic, which continuously evolve over time [37]. That is, the nodes and their edges in such graphs are undergoing constant addition or reorganization based on some underlying patterns of evolution. For example, in a social network like UCI [31], where nodes represent users and edges represent messages exchanged between users, new edges constantly emerge as users frequently exchange messages with their friends. To study this important class of graphs and their applications on the Web, we focus on dynamic graph modeling in this paper, aiming to capture the evolving patterns in a dynamic graph.

Existing works for dynamic graph modeling mainly fall into two categories: discrete-time approaches [32, 37] and continuous-time approaches [6, 40, 44, 49], as shown in Figures 1(a) and 1(b), respectively. The former regards dynamic graphs as a sequence of snapshots over a discrete set of time steps. This kind of approach usually leverages structural modules such as graph neural networks (GNN) [48] to capture the topological information of graphs, and temporal modules such as recurrent neural networks (RNN) [38] to learn the sequential evolution of the snapshots [37]. Meanwhile, the latter focuses on modeling continuous temporal patterns via specific temporal modules such as temporal random walk [30] or temporal kernel [7], illustrated by Figure 1(b). Despite the significant progress made in dynamic graph modeling, there still exist some key limitations. First, the modeling of temporal dynamics on graphs is still coarse-grained or short-termed. On one hand, discrete-time approaches discard the fine-grained temporal information within the snapshot, which inevitably results in partial loss of temporal patterns. On the other hand, while continuous-time approaches retain full temporal details by mapping each interaction to a continuous temporal space, capturing long-term dependency within historical graph data still remains a difficult problem [36, 51]. Second, the majority of the existing works rely extensively on the message-passing GNNs to encode the structural patterns in dynamic graphs. Although powerful in graph modeling, the message-passing mechanism shows inherent limitations such as over-smoothing [5] and over-squashing [1] that become more pronounced as model depth increases, preventing deeper and more expressive architectures.



(a) Discrete-time approaches (b) Continuous-time approaches (c) Self-attention in Transformer

Figure 1: Dynamic graph modeling can be summarized as follows: (a) Discrete-time methods treat the dynamic graph as a series of snapshots, ignoring the timing details within each. (b) Continuous-time methods factor in the timing of interactions, using them along with a graph learning process to update node representations X_i^t at each time t. (c) Transformer-based models handle node sequences continuously, utilizing self-attention to recognize long-term dependencies.

In pursuit of addressing these limitations, we have been intrigued by the successful application of Transformer [41] and its variants in natural language processing (NLP) [3, 16, 22] and computer vision (CV) [8, 25]. The success is underpinned by two distinct advantages inherent to the Transformer architecture: as shown in Figure 1(c), it can naturally support a continuous sequence of data without the need for discrete snapshots, and its self-attention mechanism can capture long-term dependency [41], which are important factors for dynamic graph modeling. Transformer also presents a potentially better alternative to capturing topological information, as it is less affected by the over-smoothing and over-squashing issues associated with message-passing GNNs. Hence, in this work, we explore the feasibility of the Transformer architecture for dynamic graph modeling. In fact, we have observed a growing body of research trying to modify the Transformer for static graphs [17, 34, 50]. Nonetheless, these studies primarily focus on integrating graph structural knowledge into the vanilla Transformer model, which generally still leverage message-passing GNNs as auxiliary modules to refine positional encoding and attention matrices based on graph-derived information [28]. More recently, Ying et al. [50] have demonstrated that the pure Transformer architecture holds promise for graphs. However, all these previous Transformer-based approaches only focus on static graphs, leaving unanswered questions about the feasibility for dynamic graphs, as follows.

The first challenge lies in the need to preserve the historical evolution throughout the entire timeline. However, due to the calculation of pairwise attention scores, existing Transformer-based graph models can only deal with a small receptive field, and would face serious scalability issues on even a moderately large graph. Notably, their primary application is limited to small-size graphs such as molecular graphs [34]. However, many dynamic graphs on the Web such as social networks or citation graphs are generally much larger for the vanilla Transformer to handle. To this end, we adopt a novel strategy wherein we treat the history of each node as a *temporal ego-graph*, serving as the receptive field of the ego-node. The temporal ego-graph is much smaller than the entire graph, yet it retains the full interaction history of the ego-node in the dynamic graph. Thus, we are able to preserve the temporal dynamics of every user across the entire timeline, while simultaneously ensuring scalability. Subsequently, this temporal ego-graph can be tokenized into a sequential input tailored for the Transformer. Remarkably, through this simple tokenization process, no modification to the original Transformer architecture is required.

The second challenge lies in the need to align temporal information across input sequences. Specifically, on dynamic graphs different input sequences converge within a common time domain—whether through absolute points in time (e.g., 10am on 12 October 2023) or relative time intervals (e.g., a one-hour window), with uniformity across all sequences generated from different nodes' history. In contrast, sequences for natural language or static graphs lack such a universal time domain, and can be regarded as largely independent of each other. Thus, vanilla sequences without temporal alignment lack a way to differentiate variable time intervals and frequency information. For example, a bursty stream of interactions, happening over a short one-hour interval, has a distinct evolution pattern from a steady stream containing the same number of interactions, but happening over a period of one day.

Therefore, it becomes imperative to introduce a mechanism that infuses temporal alignment among distinct input sequences generated from the ego-graphs. To address this challenge, we carefully design special *temporal tokens* to align different input sequences in the time domain. The temporal tokens serve as indicators of distinct time steps that are globally recognized across all nodes, thereby unifying different input sequences. While achieving the global alignment, local sequences from each node still retain the chronological order of the interactions in-between the temporal tokens, unlike traditional discrete-time approaches where temporal information within each snapshot is lost.

Based on the above insights, we propose a **Simple** Transformer architecture for **Dy**namic **G**raph modeling, named **SimpleDyG**. The word "simple" is a reference to the use of the original Transformer architecture without any modification, where the capability of dynamic graph modeling is simply and solely derived from constructing and modifying the input sequences. In summary, the contribution of our work is threefold. (1) We explore the potential of the Transformer architecture for modeling dynamic graphs. We propose a simple yet surprisingly effective Transformer-based approach for dynamic graphs, called SimpleDyG, without complex modifications. (2) We introduce a novel strategy to map a dynamic graph into a set of sequences to improve the scalability, by considering the history of each node as a temporal ego-graph. Furthermore, we design special temporal tokens to achieve global temporal alignment across nodes, yet preserving the chronological order of interactions at a local level. (3) We conduct extensive experiments and analyses across four real-world Web graphs, spanning diverse domains on the Web. The empirical results demonstrate not only the feasibility, but also the superiority of SimpleDyG.

2 RELATED WORK

Dynamic Graph Learning. Current dynamic graph learning methods can be categorized into discrete-time and continuous-time approaches. In discrete-time methods, dynamic graphs are treated as a series of static graph snapshots taken at regular time intervals. To model both structural and temporal aspects, these approaches integrate the GNNs with sequence models, such as RNNs or self-attention mechanisms [10, 32, 37, 39]. For instance, DySAT [37] leverages a graph attention network and self-attention as fundamental components for both structural and temporal modules, whereas EvolveGCN [32] employs an RNN to evolve the parameters of graph convolutional networks. Nevertheless, they often fall short of capturing the granular temporal information. In contrast, the continuous-time approaches consider every interaction event at each specific timestamp. Some approaches model dynamic graph evolution as temporal random walks or causal anonymous walks [30, 43]. Another line of research focuses on time encoding techniques, which integrate with graph structure modeling, such as the temporal graph attention used in TGAT [49] and TGN [36], or MLP-Mixer layers applied in GraphMixer [6]. Additionally, researchers also leverage temporal point processes to capture the graph formation process [14, 40, 44]. Despite the promise demonstrated by continuous-time approaches, it is important to note that they come with limitations in capturing long-term dependencies within historical data.

The differences between our work and the previous dynamic graph learning methods lie in two points. First, our method effectively mitigates the long-term dependency challenge, leveraging the inherent advantages of the Transformer architecture. Second, our method preserves the chronological history of each input sequence. In particular, the temporal alignment mechanism synchronizes different input sequences, empowering our model to capture both global and local information within the dynamic graphs.

Transformers for Graphs. Transformer architectures for graphs have emerged as a compelling alternative to conventional GNNs, aiming to mitigate issues like over-smoothing and over-squashing. Prior research has focused on integrating graph information into the vanilla Transformer through diverse strategies. Some methods integrate GNNs as auxiliary components to bolster structural comprehension within the Transformer architecture [18, 35]. Others focus on enriching positional embeddings by spatial information derived from the graph. For instance, Graphormer [50] integrates the centrality, spatial and edge encoding into Transformers, whereas Cai and Lam [4] adopt distance embedding for tree-structured abstract meaning representation graph, and Kreuzer et al. [19] utilize the full Laplacian spectrum to learn the positional encoding. There are also studies on refining the attention mechanism in Transformers for graph modeling. For instance, Min et al. [29] employ a graph masking attention mechanism to seamlessly inject graph-related priors into the Transformer architecture. More recently, Kim et al. [17] have shed light on the effectiveness of pure Transformers in graph learning without complex designs. Their approach treats all nodes and edges as independent tokens, serving as inputs for the Transformer. Besides, Mao et al. [26] propose a Transformer-based model for heterogeneous graphs, integrating local structures and heterogeneous relations into the attention mechanism.

It is worth noting that most of the previous works based on Transformers mainly deal with static graphs. Recently, Yu et al. [51] have introduced a Transformer-based model designed for dynamic graph learning, which is contemporary with our work. The difference lies in that they rely on complex designs for capturing co-occurrence neighbors of different nodes and encoding temporal intervals. In contrast, we explore a simple Transformer for dynamic graphs without the need for complex modifications.

3 PRELIMINARIES

We first define the problem of dynamic graph modeling. Then, we briefly introduce the background of the Transformer architecture.

3.1 Dynamic Graph Modeling

We define a dynamic graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{X})$ with a set of nodes $\mathcal{V}$, edges $\mathcal{E}$, a time domain $\mathcal{T}$ and an input feature matrix $\mathcal{X}$. It can be characterized by a sequence of time-stamped edges $\mathcal{G} = \{(v_i, v_j, \tau)_n : n = 1, 2, \ldots, |\mathcal{E}|\}$. Here, each tuple (v_i, v_j, τ) denotes a distinct interaction between nodes v_i and v_j at time $\tau \in \mathcal{T}$, with $|\mathcal{E}|$ representing the total number of interactions in the dynamic graph. Given the dynamic graph $\mathcal{G}$, we learn a model with parameter θ to capture the temporal evolution of the graph. The temporal representations encoded by the learned model θ can be used for different tasks such as node classification, link prediction and graph classification.

3.2 Transformer Architecture

The standard Transformer architecture comprises two main components: the multi-head self-attention layers (MHA) and the position-wise feed-forward network (FFN). In the following part, we will briefly introduce these blocks.

We represent an input sequence as $\mathbf{H} = \langle \mathbf{h}_1, \ldots, \mathbf{h}_N \rangle \in \mathbb{R}^{N \times d}$, where $\mathbf{h}_i$ is the hidden representation for token i, and d is the dimension of the representations. The MHA module projects $\mathbf{H}$ to a triplet $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, as follows.

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_Q, \mathbf{K} = \mathbf{H}\mathbf{W}_K, \mathbf{V} = \mathbf{H}\mathbf{W}_V, \tag{1}$$

where $\mathbf{W}_Q \in \mathbb{R}^{d \times d_K}$, $\mathbf{W}_K \in \mathbb{R}^{d \times d_K}$, $\mathbf{W}_V \in \mathbb{R}^{d \times d_V}$ are learnable weights, with $d_K = d_V = d/H$. Overall, H such projections are performed, resulting in $(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h)$ for $1 \le h \le H$. The self-attention operation is then applied to each triplet:

$$\text{head}_h = \textsc{Softmax}\left(\frac{\mathbf{Q}_h \mathbf{K}_h^T}{\sqrt{d_K}}\right)\mathbf{V}_h, \tag{2}$$

$$\text{MHA}(\mathbf{H}) = \textsc{Concat}(\text{head}_1, \ldots, \text{head}_H)\mathbf{W}_O, \tag{3}$$

where $\mathbf{W}_O \in \mathbb{R}^{d \times d}$ is a learnable weight matrix.

The output of the MHA module is then passed through a feed-forward network layer followed by residual connection [12] and layer normalization (LN) [2]. Finally, the output of the l^{th} layer $\mathbf{H}^l$ is computed as follows:

$$\widehat{\mathbf{H}}^l = \text{LN}(\mathbf{H}^{l-1} + \text{MHA}(\mathbf{H}^{l-1})), \tag{4}$$

$$\mathbf{H}^l = \text{LN}(\widehat{\mathbf{H}}^l + \text{FFN}(\widehat{\mathbf{H}}^l)). \tag{5}$$

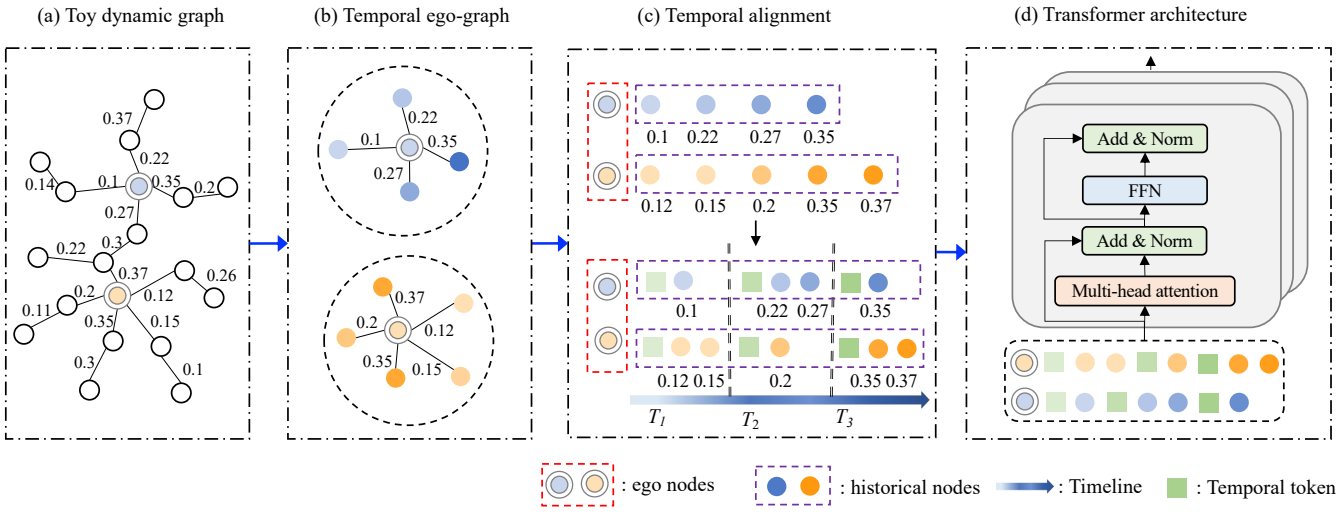


Figure 2: Overall framework of SimpleDyG. Best viewed in color. The numerical values adjacent to the links in (a) and (b), as well as beneath the nodes in (c), represent the time elapsed from the beginning, indicating the moments at which the links emerge (ranging from 0 to 1). The color intensity of nodes in the historical sequence represents the time span, where darker colors signify a longer duration, while lighter colors indicate a shorter duration.

4 PROPOSED APPROACH

The overall framework of SimpleDyG is illustrated in Figure 2. Our framework is applied to a dynamic graph $\mathcal{G}$ (Figure 2(a)), where multiple temporal links emerge at various time points. In order to capture the dynamic evolution, we begin by extracting a *temporal ego-graph* for each ego-node, which contains the entire historical interactions as shown in Figure 2(b). These temporal graphs are subsequently transformed into sequences while preserving their chronological order. To incorporate temporal alignment among different ego-graphs, we segment the timeline into various time spans with the same temporal interval as in Figure 2(c). Then, we add *temporal tokens* into the ego-sequence to identify different time spans. Finally, these sequences are fed into a Transformer architecture to facilitate various downstream tasks.

4.1 Temporal Ego-graph

The strategy of mapping a dynamic graph into a sequence of tokens is crucial for utilizing the Transformer architecture for dynamic graph modeling. In this work, we regard nodes in the dynamic graphs as input tokens, which is a common approach in Transformer models for graphs. Besides, to preserve the historical interactions of all the nodes while ensuring scalability, we extract a temporal ego-graph for each node in the dynamic graph. Each ego-graph serves as the receptive field of its ego-node, which is mapped into a sequence to capture the structural and temporal evolution of the ego-node.

Specifically, we denote $v_i \in \mathcal{V}$ as an ego-node in the dynamic graph $\mathcal{G}$. We extract the historically interacted nodes for v_i and construct an temporal ego-graph centered at v_i. Formally, we denote the *temporal ego-graph* for the ego-node v_i as a chronologically ordered sequence $w_i = \langle v_i^1, v_i^2 \dots v_i^{|w_i|} \rangle$, where $|w_i|$ is the length of the sequence. Note that $\forall 1 \le k < k' \le |w_i|$, v_i^k and $v_i^{k'}$ represent some historical interactions (v_i, v_i^k, τ) and $(v_i, v_i^{k'}, \tau')$, respectively, such that $\tau \le \tau'$.

To better model the patterns within the sequences, we follow similar practices as in natural language processing and include some special tokens designed for our task. During training, the input sequence x_i and the ground-truth output sequence y_i are constructed as follows[1].

$$\begin{aligned} x_i &= \langle |hist| \rangle, v_i, v_i^1, \dots v_i^{|w_i|}, \langle |endofhist| \rangle, \\ y_i &= \langle |pred| \rangle, v_i^{|w_i|+1}, \dots, v_i^{|w_i|+z} \langle |endofpred| \rangle, \end{aligned} \tag{6}$$

where the "$\langle |hist| \rangle$" and "$\langle |endofhist| \rangle$" are special tokens indicating the start and end of the input historical sequence, and "$\langle |pred| \rangle$" and "$\langle |endofpred| \rangle$" are reserved for signalling the prediction of the next nodes following the input sequence. Specifically, the model will halt its predictions once the end special token is generated, enabling automatic decisions on the number of predictions at a future time point. Potentially, by devising appropriate special tokens, our framework can also support other operations such as link deletion, as illustrated in Appendix A.

4.2 Temporal Alignment

In the original Transformer, the input sequence is treated as a sequence of tokens, and the model captures the relationships between these tokens based on their relative positions in the sequence. For dynamic graph modeling, the positions in the sequence represent the temporal order. However, it inherently lacks the capability to account for a universal time domain across sequences, to synchronize the time interval and frequency information.

[1]Special tokens in the beginning and at the end such as "$\langle |endoftext| \rangle$" are omitted for easy illustration.

We introduce a straightforward yet effective strategy to incorporate temporal alignment into the input sequences of the Transformer. First, we segment the time domain $\mathcal{T}$ into coarse-grained time steps, where the intervals between two consecutive time steps are equal, such as one week or one month, determined by dataset characteristics. It is important to note that our approach differs from discrete-time approaches: Within each time step, we consider the precise temporal order of each event. In contrast, discrete-time approaches treat each time step as a static snapshot. Next, we incorporate special *temporal tokens* into the sequences, which explicitly denote different time steps that are globally recognized across all sequences. Suppose we split the time domain $\mathcal{T}$ into T time steps, where the input sequence includes the first $T-1$ time steps and the output sequence covers the last time step. Then, the sequences of the ego-node i can be denoted as follows:

$$x'_i = \langle|hist|\rangle, v_i, \langle|time1|\rangle, S_i^1, \dots \langle|timeT\text{-}1|\rangle, S_i^{T-1}, \langle|endofhist|\rangle, \quad (7)$$

$$y'_i = \langle|pred|\rangle, \langle|timeT|\rangle, S_i^T \langle|endofpred|\rangle, \quad (8)$$

$$S_i^t = \langle v_i^{t,1}, v_i^{t,2} \dots v_i^{t,|S_i^t|}\rangle. \quad (9)$$

Here S_i^t represents the historical sequence of node v_i at time step t with length $|S_i^t|$. In particular, "$\langle|time1|\rangle$", ..., "$\langle|timeT|\rangle$" are temporal tokens that serve as indicators of temporal alignment, enabling the model to recognize and capture temporal patterns in different sequences. The temporal tokens enhances the Transformer's ability to understand the dynamics across the entire graph.

4.3 Training objective

A training instance is formed by concatenating the input x and output y as $[x; y]$. We denote it as $R = \langle r_1, r_2, \cdots, r_{|R|}\rangle$ with $|R|$ tokens. For a given training instance in this format, we follow the typical masking strategy: During the prediction of the i-th token, only the input sequence up to position $i-1$, denoted by $R_{<i}$, is taken into account, while the subsequent tokens are subject to masking. The joint probability of the next token is calculated as follows:

$$p(R) = \prod_{i=1}^{|R|} p(r_i|R_{<i}), \quad (10)$$

where $p(r_i|R_{<i})$ is the probability distribution of the token to be predicted at step i conditioned on the tokens $R_{<i}$. It is computed as

$$p(r_i|R_{<i}) = \mathrm{LN}(\mathbf{R}^l_{<i})\mathbf{W}_{\text{vocab}}, \quad (11)$$

where LN means layer normalization, $\mathbf{R}^L_{<i}$ denotes the hidden representation of the historically generated tokens before step i, which is obtained from the last layer of the Transformer, and $\mathbf{W}_{\text{vocab}}$ is a learnable matrix aiming to compute the probability distribution across the vocabulary of nodes in the graph.

Given a set of training instances $\mathcal{R}$, the loss function for training the model with parameters θ is defined as the negative log-likelihood over $\mathcal{R}$, as follows:

$$\mathcal{L} = -\sum_{R\in\mathcal{R}} \sum_{i=1}^{|R|} \log p_\theta(r_i|R_{<i}). \quad (12)$$

We outline the training procedure of SimpleDyG in Algorithm 1. For each prediction step i of one training instance, the hidden representations of the generated sequence $\mathbf{R}_{<i}$ are used for predicting the next token. The joint probability of the next token is computed using Equations 10 and 11. Our model is trained using the Adam optimizer with a loss function based on the negative log-likelihood in Equation 12.

Algorithm 1: Training Procedure of SimpleDyG

```
Input: Dynamic graph G = (V, E, T, X),
       training instances R
Output: Model with parameters θ
initialize θ
while not converged do
    sample a batch of instances B from R
    for each instance R = ⟨r_1, r_2, ··· , r_|R|⟩ ∈ B do
        while step i < |R| do
            /* prediction steps for one instance */
            calculate the representation R_<i for R_<i
            calculate the joint probability by Eqs. 10 and 11
            calculate the loss by Eq. 12
    update θ via backpropagation
return θ
```

Table 1: Dataset statistics

Dataset	UCI	ML-10M	Hepth	MMConv
Domain	Social	Rating	Citation	Conversation
# Nodes	1,781	15,841	4,737	7,415
# Edges	16,743	48,561	14,831	91,986

5 EXPERIMENTS

In this section, we conduct extensive experiments on four public datasets across different domains, with comparison to the state-of-the-art baselines and detailed analysis of the model performance.

5.1 Experimental Setup

Datasets. To evaluate the performance of our proposed method, we conducted experiments on four datasets from various domains, including the communication network UCI [31], the rating network ML-10M [11], the citation network Hepth [21], and the multi-turn conversation dataset MMConv [23]. The detailed statistics of all datasets after preprocessing are presented in Table 1.

UCI [31]: It represents a social network in which edges represent messages exchanged among users. For temporal alignment, we employ 13 time steps following previous work [37].

ML-10M [11]: We utilize the ML-10M dataset from MovieLens comprising user-tag interactions, where the edges connect users to the tags they have assigned to specific movies. For temporal alignment, we employ 13 time steps following previous work [37].

Hepth [21]: It is a citation network for high-energy physics theory. For temporal alignment, we extract 24 months of data from this dataset and split them into 12 time steps based on the wall-clock timestamps. Note that this dataset contains new emerging nodes as time goes on. We use the word2vec model [27] to extract the input feature for each paper based on the abstract.

MMConv [23]: It contains a multi-turn task-oriented dialogue system that assists users in discovering places of interest across five domains. Leveraging this rich annotation, we represent the dialogue as a dynamic graph, a widely adopted strategy in task-oriented dialogue systems. For temporal alignment, we employ 16 time steps, each corresponding to a distinct turn in the conversation.

Baselines. We compare SimpleDyG with baselines in two categories: (1) discrete-time approaches: DySAT [37] and EvolveGCN [32]; (2) continuous-time approaches: DyRep [40], JODIE [20], TGAT [49], TGN [36], TREND [44] and GraphMixer [6].

- **DySAT** [37] leverages joint structural and temporal self-attention to learn the node representations at each time step.
- **EvolveGCN** [32] employs RNNs to evolve graph convolutional network parameters.
- **DyRep** [40] utilizes a two-time scale deep temporal point process model to capture temporal graph topology and node activities.
- **JODIE** [20] focuses on modeling the binary interaction among users and items by two coupled RNNs. A projection operator is designed to predict the future node representations at any time.
- **TGAT** [49] employs temporal graph attention layers and time encoding techniques to aggregate temporal-topological features.
- **TGN** [36] combines the memory modules and message passing to maintain the dynamic representations. This model also adopts time encoding and temporal graph attention layers.
- **TREND** [44] exploits Hawkes process-based GNNs for dynamic graph modeling. It integrates event and node dynamics to capture the individual and collective characteristics of events.
- **GraphMixer** [6] relies on MLP layers and neighbor mean-pooling to learn the edge and node encoders. An offline time encoding function is adopted to capture the temporal information.

Implementation Details. We evaluate the performance of SimpleDyG on the link prediction task. Given the ego-nodes, the objective of the link prediction task is to predict the possible linked nodes at time step T. For all the datasets, we follow the previous setting [6] by treating the dynamic graphs as undirected graphs. We split each dataset into training/validation/testing based on the predefined time steps. We choose the data at the last time step T as the testing set, while the data at time step $T-1$ serves as the validation set, with the remaining data for training. We tune the hyperparameters for all methods on the validation set. All experiments are repeated ten times, and we report the averaged results with standard deviation. We provide further implementation details and hyperparameter settings in Appendices B and C.

Evaluation Metrics. In our evaluation, we carefully select metrics catered to our specific task. The goal of the link prediction task is to predict a set of nodes linked to each ego-node at a given time step. Notably, our SimpleDyG model predicts a node sequence, with each prediction influenced by the prior ones until the generation of an end token. In contrast, the baseline models make simultaneous predictions of entire ranking sequences for each ego-node. To evaluate the ranking performance and the similarity between predicted and ground-truth node sets, we employ two key metrics: *NDCG@5* and *Jaccard* similarity. *NDCG@5* is a well-established metric commonly used in information retrieval and ranking tasks [42], aligning with our objective of ranking nodes and predicting the top nodes linked to an ego-node. On the other hand, *Jaccard* similarity is valuable for quantifying the degree of overlap between two sets [13], measuring the similarity between predicted nodes and the ground-truth nodes associated with the ego-node. Specifically, for the baseline models, we choose the top k nodes ($k = 1, 5, 10, 20$) as the predicted set, as they are not generative models and cannot determine the end of the prediction. We then select the maximum Jaccard similarity value across different k's as the final Jaccard similarity score. This evaluation strategy ensures a thorough and fair assessment of the baselines in comparison to SimpleDyG.

5.2 Performance Comparison to Baselines

We report the results of all methods under *NDCG@5* and *Jaccard* across four diverse datasets in Table 2. Generally speaking, our method outperforms all the baselines on all datasets, and we make the following key observations.

First, we find that continuous-time approaches generally perform better than discrete ones across a wide range of scenarios, indicating the important role of time-related information in dynamic graph analysis. Notably, continuous-time baselines such as GraphMixer exhibit superior performance. This superiority can be mainly attributed to the simple MLP-Mixer architecture, which makes it easier to capture long-term historical sequences with lower complexity. In contrast, other models like DyRep, TGAT, and TGN, which rely on complex designs such as GNNs and GATs, display subpar performance. This phenomenon stems from the inherent limitations of GNNs and GATs in capturing distant relationships or broader historical contexts within predefined time windows.

Second, in inductive scenarios like the Hepth dataset, where the ego-nodes in the testing data are newly emerged nodes, continuous-time models that employ a GNN-based backbone exhibit superior performance compared to GraphMixer. To be able to handle new nodes, the initial node features are constructed using word2vec, which might be relatively coarse. Since GraphMixer predominantly relies on an MLP-based architecture, it may encounter challenges given the coarse-grained initial features. Conversely, GNN-based methods integrate structural information with these features, thereby empowering them to excel in the inductive scenario. Nevertheless, in our Transformer-based model, there is the added advantage of modeling long-range dependencies, resulting in consistently better performance of SimpleDyG.

5.3 Effect of Extra Tokens

We design extra tokens to make the vanilla Transformer architecture more suitable for dynamic graph modeling. To assess their effectiveness, we conduct an in-depth analysis of these token designs, including *special tokens* that signal the input and output, and the *temporal tokens* that align different sequences.

Impact of *special tokens*. The *special tokens* include the start and end of the historical sequence ("$\langle|hist|\rangle$" and "$\langle|endofhist|\rangle$"), as well as the predicted sequence ("$\langle|pred|\rangle$"and "$\langle|endofpred|\rangle$"). To comprehensively evaluate their effect across diverse scenarios, we examine two variants of SimpleDyG: (1) *same special*, where we use the same *special tokens* for input and output. (2) *no special*, where we entirely remove all special tokens from each instance. We show the results in Table 3 and make the following observations.

Table 2: Performance of dynamic link prediction by SimpleDyG and the baselines on four datasets. (In each column, the best result is bolded and the runner-up is underlined.)

	UCI		ML-10M		Hepth		MMConv	
	NDCG@5	*Jaccard*	*NDCG@5*	*Jaccard*	*NDCG@5*	*Jaccard*	*NDCG@5*	*Jaccard*
DySAT [37]	0.010±0.003	0.010±0.001	0.058±0.073	0.050±0.068	0.007±0.002	0.005±0.001	0.102±0.085	0.095±0.080
EvolveGCN [32]	0.064±0.045	0.032±0.026	0.097±0.071	0.092±0.067	0.009±0.004	0.007±0.002	0.051±0.021	0.032±0.017
DyRep [40]	0.011±0.018	0.010±0.005	0.064±0.036	0.038±0.001	0.031±0.024	0.010±0.006	0.140±0.057	0.067±0.025
JODIE [20]	0.022±0.023	0.012±0.009	0.059±0.016	0.020±0.004	0.031±0.021	0.011±0.008	0.041±0.016	0.032±0.022
TGAT [49]	0.061±0.007	0.020±0.002	0.066±0.035	0.021±0.007	0.034±0.023	0.011±0.006	0.089±0.033	0.058±0.021
TGN [36]	0.041±0.017	0.011±0.003	0.071±0.029	0.023±0.001	0.030±0.012	0.008±0.001	0.096 ±0.068	0.066±0.038
TREND [44]	0.067±0.010	0.039±0.020	0.079±0.028	0.024±0.003	0.031±0.003	0.010±0.002	0.116±0.020	0.060±0.018
GraphMixer [6]	**0.104±0.013**	0.042±0.005	0.081±0.033	0.043±0.022	0.011±0.008	0.010±0.003	0.172±0.029	0.085±0.016
SimpleDyG	**0.104±0.010**	**0.092±0.014**	**0.138±0.009**	**0.131±0.008**	**0.035±0.014**	**0.013±0.006**	**0.184±0.012**	**0.169±0.010**

Table 3: Impact of special tokens in SimpleDyG across four datasets.

	UCI		ML-10M		Hepth		MMConv	
	NDCG@5	*Jaccard*	*NDCG@5*	*Jaccard*	*NDCG@5*	*Jaccard*	*NDCG@5*	*Jaccard*
SimpleDyG	0.104±0.010	0.092±0.014	**0.138±0.009**	**0.131±0.008**	0.035±0.014	0.013±0.006	**0.184±0.012**	0.169±0.010
same special	**0.113±0.007**	**0.095±0.010**	0.085±0.046	0.079±0.043	0.027±0.014	0.009±0.005	0.179±0.013	**0.170±0.010**
no special	0.041±0.025	0.020±0.011	0.006±0.009	0.006±0.009	**0.096±0.016**	**0.025±0.006**	0.01±0.008	0.008±0.007

Table 4: Impact of different temporal alignment designs on the four datasets.

	UCI		ML-10M		Hepth		MMConv	
	NDCG@5	*Jaccard*	*NDCG@5*	*Jaccard*	*NDCG@5*	*Jaccard*	*NDCG@5*	*Jaccard*
SimpleDyG	0.104±0.010	**0.092±0.014**	0.138±0.009	0.131±0.008	0.035±0.014	0.013±0.006	0.184±0.012	0.169±0.010
same time	0.090±0.013	0.083±0.012	**0.147±0.001**	**0.139±0.001**	**0.046±0.009**	0.017±0.004	0.240±0.031	0.212±0.025
no time	**0.111±0.015**	0.091±0.014	0.117±0.062	0.111±0.059	0.045±0.007	**0.018±0.003**	**0.260±0.019**	**0.237±0.016**

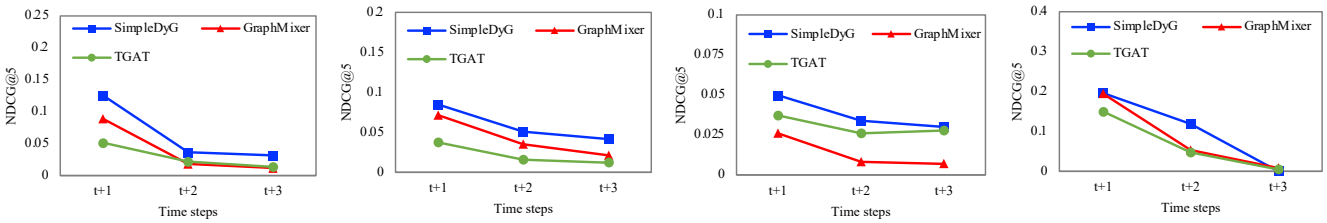
In general, *special tokens* enhance the link prediction performance across different datasets. Furthermore, the differences between the *same special* and original SimpleDyG tend to be small. However, an interesting finding emerges in the case of the Hepth dataset, where the *no special* variant yields the best performance. It can be explained by the specific characteristic of the citation graph. In the testing data of Hepth, the ego-nodes are newly emerged nodes, representing the newly published papers. Consequently, the input samples lack any historical information, leaving the distinction between the history and the predicted irrelevant.

Impact of *temporal tokens*. To evaluate the impact of *temporal tokens*, we compare the performance of SimpleDyG with two ablated variants: (1) *same time*, where we do not distinguish specific time steps and employ the same *temporal tokens* for each time step, and (2) *no time*, in which we entirely remove all temporal tokens from all sequences.

The results are presented in Table 4. It is surprising and interesting to observe performance improvement with less or no temporal alignment, particularly on the MMConv and Hepth datasets. We hypothesize that the citation relationship and the conversation among different ego-nodes do not strictly follow a universal temporal framework. Using the same temporal tokens (*same time*) or none at all (*no time*) allows the model to adapt more naturally to the temporal order. The temporal alignment plays a more important role for the UCI and ML-10M datasets. However, they show different trends with the *same time* version. A potential reason is that, in UCI, the communication patterns between different users are sensitive to the segmentation into different time steps. Hence, *same time* performs the worst, as it divides a sequence into time steps yet without time differentiation to align across the sequences, where the additional same tokens may confuse the model. On the other hand, *no time* still retains the full temporal order, and thus perform better than *same time*.

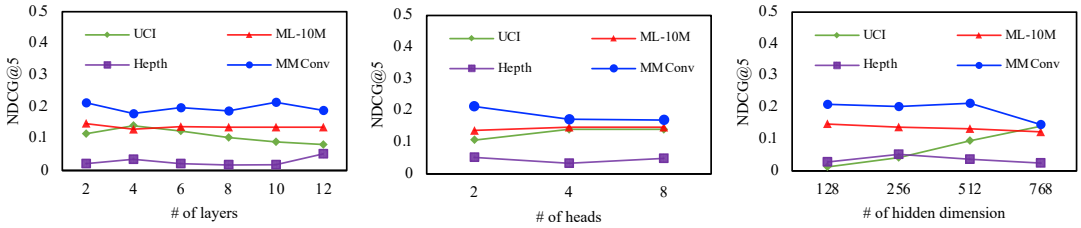
5.4 Performance of Multi-step Prediction

We evaluate the ability of SimpleDyG for multi-step prediction with the time steps ranging from t to $t + \triangle t$, utilizing a model that has been trained on data up to time t. In our experiment, we set $\triangle t = 3$. For SimpleDyG, we perform step-by-step prediction, conditioned on the results of previous steps. We compare to two competitive baselines, TGAT and GraphMixer. As both methods incorporate



(a) UCI (b) ML-10M (c) Hepth (d) MMConv

Figure 3: Performance trends of multi-step prediction.



(a) Number of layers (b) Number of heads (c) Hidden dimension

Figure 4: Impact of hyperparameters.

time information via time encoding, we employ the model trained at time t to directly predict the links at time $t + \triangle t$. Their performance trends are plotted in Figure 3.

We observe a natural decay in performance over time for all methods, as anticipated. However, SimpleDyG consistently outperforms the baselines as time progresses. This trend underscores the effectiveness of our Transformer architecture in modeling long-term dependencies in a dynamic graph.

5.5 Hyper-parameter Analysis

We undertake an examination of the critical hyperparameter choices, taking into account the variations observed across different datasets. Specifically, we systematically explore the impact of several key hyperparameters, namely, the number of layers, the number of heads, and the hidden dimension size. These hyperparameters play a pivotal role in shaping the model's capacity and its ability to capture intricate patterns within dynamic graphs. We vary the value of each hyperparameters while keeping all other parameters constant. From Figure 4, we draw some key observations as follows.

- **Number of layers**: The variance of performance under different numbers of layers is relatively small. This suggests that the choice of the number of layers in SimpleDyG has a more consistent impact across different datasets and scenarios. Generally speaking, two layers are typically sufficient for most cases. For inductive scenarios such as the Hepth dataset, it is advisable to use more layers to effectively capture the evolving graph structure.
- **Number of heads**: For the number of attention heads, we find that using either 2 or 4 heads is generally suitable for a wide range of scenarios. These settings provide a good balance between performance and computational efficiency.
- **Hidden dimension size**: The choice of hidden dimension size depends on the complexity of the dataset. For datasets like movie ratings (e.g., ML-10M), a hidden dimension size of 128 is often adequate. However, for datasets involving more intricate interactions, such as communication networks or conversation datasets, it becomes necessary to use larger hidden dimension sizes like 256 or 512. Notably, the UCI dataset requires a hidden dimension of 768, which can be explained by the complexity and richness of the interactions among users within the dataset.

6 CONCLUSION

In this work, we explored the problem of dynamic graph modeling, recognizing its significance across a wide range of applications. Drawing from the strengths of the Transformer's self-attention mechanism, we tailored a solution that supersedes the often convoluted designs in many existing methods. Our novel approach, named SimpleDyG, reformulates dynamic graphs from a sequence modeling perspective. It is superior to not only the discrete-time approaches by considering the full temporal order, but also the continuous-time approaches by capturing long-term dependencies using a Transformer. SimpleDyG is exceptionally simple as it does not modify the Transformer architecture; instead, it maps a dynamic graph into a set of sequences via temporal ego-graphs, and modifies the input sequences to achieve temporal alignment. Nevertheless, SimpleDyG achieves surprisingly strong performance in diverse dynamic graphs. As future work, we will investigate the nuances of the temporal alignment technique for further optimizations.

ACKNOWLEDGMENTS

This research / project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Proposal ID: T2EP20122-0041). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

REFERENCES

[1] Uri Alon and Eran Yahav. 2020. On the Bottleneck of Graph Neural Networks and its Practical Implications. In *International Conference on Learning Representations*.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 7464–7471.

[5] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 3438–3445.

[6] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. 2022. Do We Really Need Complicated Model Architectures For Temporal Networks?. In *The Eleventh International Conference on Learning Representations*.

[7] Bert De Vries and Jose C Principe. 1992. The gamma model—A new neural model for temporal processing. *Neural networks* 5, 4 (1992), 565–576.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

[9] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The world wide web conference*. 417–426.

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

[11] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Paul Jaccard. 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat* 37 (1901), 547–579.

[14] Yugang Ji, Tianrui Jia, Yuan Fang, and Chuan Shi. 2021. Dynamic heterogeneous graph embedding via heterogeneous hawkes process. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*. Springer, 388–403.

[15] Anshul Kanakia, Zhihong Shen, Darrin Eide, and Kuansan Wang. 2019. A scalable hybrid research paper recommender system for microsoft academic. In *The world wide web conference*. 2893–2899.

[16] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[17] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. 2022. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems* 35 (2022), 14582–14595.

[18] Jinwoo Kim, Saeyoon Oh, and Seunghoon Hong. 2021. Transformers generalize deepsets and can be extended to graphs & hypergraphs. *Advances in Neural Information Processing Systems* 34 (2021), 28016–28028.

[19] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. 2021. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems* 34 (2021), 21618–21629.

[20] 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.

[21] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 177–187.

[22] Lizi Liao, Le Hong Long, Yunshan Ma, Wenqiang Lei, and Tat-Seng Chua. 2021. Dialogue state tracking with incremental reasoning. *Transactions of the Association for Computational Linguistics* 9 (2021), 557–569.

[23] Lizi Liao, Le Hong Long, Zheng Zhang, Minlie Huang, and Tat-Seng Chua. 2021. MMConv: an environment for multimodal conversational search across multiple domains. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 675–684.

[24] Lizi Liao, Grace Hui Yang, and Chirag Shah. 2023. Proactive conversational agents. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*. 1244–1247.

[25] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*. 10012–10022.

[26] Qiheng Mao, Zemin Liu, Chenghao Liu, and Jianling Sun. 2023. Hinormer: Representation learning on heterogeneous information networks with graph transformer. In *Proceedings of the ACM Web Conference 2023*. 599–610.

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[28] Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. 2022. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455* (2022).

[29] Erxue Min, Yu Rong, Tingyang Xu, Yatao Bian, Da Luo, Kangyi Lin, Junzhou Huang, Sophia Ananiadou, and Peilin Zhao. 2022. Neighbour interaction based click-through rate prediction via graph-masked transformer. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 353–362.

[30] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion proceedings of the the web conference 2018*. 969–976.

[31] 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.

[32] 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.

[33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[34] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. 2022. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems* 35 (2022), 14501–14515.

[35] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. 2020. Self-supervised graph transformer on large-scale molecular data. *Advances in Neural Information Processing Systems* 33 (2020), 12559–12571.

[36] 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).

[37] 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.

[38] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[39] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2019. Learning to represent the evolution of dynamic graphs with recurrent models. In *Companion proceedings of the 2019 world wide web conference*. 301–307.

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

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

[42] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[43] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *International Conference on Learning Representations (ICLR)*.

[44] 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.

[45] Yuxia Wu, Ke Li, Guoshuai Zhao, and Xueming Qian. 2020. Personalized long-and short-term preference learning for next POI recommendation. *IEEE Transactions on Knowledge and Data Engineering* 34, 4 (2020), 1944–1957.

[46] Yuxia Wu, Lizi Liao, Xueming Qian, and Tat-Seng Chua. 2022. Semi-supervised New Slot Discovery with Incremental Clustering. In *Findings of the Association for Computational Linguistics*. 6207–6218.

[47] Yuxia Wu, Lizi Liao, Gangyi Zhang, Wenqiang Lei, Guoshuai Zhao, Xueming Qian, and Tat-Seng Chua. 2022. State graph reasoning for multimodal conversational recommendation. *IEEE Transactions on Multimedia* (2022).

[48] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.
[49] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. (2020).
[50] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems* 34 (2021), 28877–28888.
[51] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. 2023. Towards Better Dynamic Graph Learning: New Architecture and Unified Library. In *NeurIPs*.
[52] Xingtong Yu, Yuan Fang, Zemin Liu, Yuxia Wu, Zhihao Wen, Jianyuan Bo, Xinming Zhang, and Steven CH Hoi. 2024. Few-Shot Learning on Graphs: from Meta-learning to Pre-training and Prompting. *arXiv preprint arXiv:2402.01440* (2024).
[53] Xingtong Yu, Zemin Liu, Yuan Fang, and Xinming Zhang. 2023. HGPROMPT: Bridging Homogeneous and Heterogeneous Graphs for Few-shot Prompt Learning. *arXiv preprint arXiv:2312.01878* (2023).
[54] Xingtong Yu, Chang Zhou, Yuan Fang, and Xinming Zhang. 2023. MultiGPrompt for Multi-Task Pre-Training and Prompting on Graphs. *arXiv preprint arXiv:2312.03731* (2023).

APPENDICES

A LINK DELETION OPERATION

Our framework can support the link deletion operation within the evolution of dynamic graphs. For example in the ML-10M dataset with links representing users' ratings for the movies, assuming that users can delete the historical ratings. Suppose the historically rated movies of one user is "$[a, b, c]$", he/she may delete the rating of b, then we can add a special token "[del]" to identify the deletion operation: "$[a, [del], b, c]$". It is worth noting that most works for dynamic graph modeling are associated with link addition with benchmark datasets mainly involving this type of operation.

B ADDITIONAL IMPLEMENT DETAILS

Note that the implementation details of baseline approaches in their publicly released code are quite different. For instance, most of them regard the link prediction task as binary classification, where the objective is to determine the presence or absence of links between the positive pairs of nodes and randomly selected negative pairs. They either employ a binary cross-entropy loss to facilitate classifier learning or utilize logistic regression to train an additional classifier. To tailor these baselines to our specific task for a fair comparison, we adapt them into a ranking task and substitute the classifier loss with a pair-wise Bayesian personalized ranking (BPR) loss for all baselines.

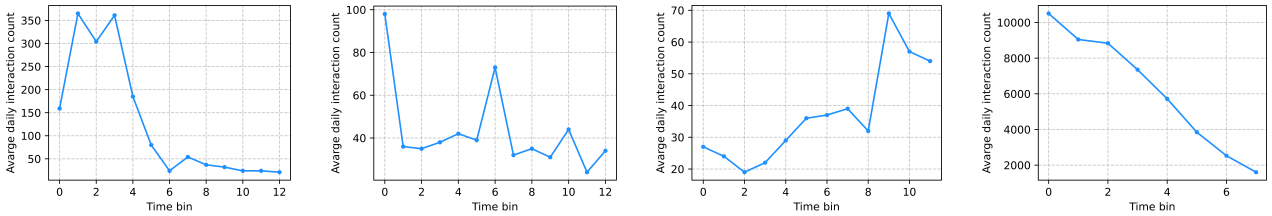
C HYPERPARAMETERS SETTINGS OF BASELINES

Considering that we refine the loss of the baselines as BPR loss, we tune the important parameters of all baselines for all the datasets. For all baselines, we tune the parameter of hidden dimension with $\{16, 32, 64, 128, 256, 512\}$ for each dataset. For a fair comparison with our model, we don't set a historical window for discrete-time approaches and use all the historical data.

Some important parameters for each baseline are listed as follows: For DySAT [37], We set the self-attention layers and head to be 2 and 8, respectively. For EvolveGCN [32], the number of GCN layers is 1. For DyRep [40], the message aggregation layer is 2, and the number of neighbor nodes is 20. For TGAT [49] and TGN [36], the number of graph attention heads is 2 and the attention layers are 1 and 2, respectively. For GraphMixer [6], the number of MLP layers for UCI is 1 and 2 for other datasets. For a fair comparison, we set the historical length of each node to 1024, which is the same as our model.

D STATISTICAL TEMPORAL PATTERN ANALYSIS IN EACH DATASET

We incorporate statistical analyses to understand the temporal patterns within each dataset. Specifically, we counted the number of interactions per day for the UCI, ML-10M, and Hepth datasets. For the conversation dataset (MMConv), we counted the number of interactions per turn. We split the dataset into several bins with each bin covering a range of time (10 days, 90 days, 60 days and two turns for UCI, ML-10M, Hepth and MMConv, respectively), and then calculate the average daily interaction counts in each bin. We show the results in Figure A.1.



(a) UCI (b) ML-10M (c) Hepth (d) MMConv

Figure A.1: The temporal pattern of each dataset.

Table 5: Time Efficiency of Different Methods

Method	DySAT	EvolveGCN	DyRep	JODIE	TGAT	TGN	TREND	GraphMixer	SimpleDyG
Time (s)	12.24	11.89	6.4	6.25	18.54	8.05	7.45	6.89	6.21

We observe sudden changes in interactions in the UCI and ML-10M datasets. While the Hepth and MMConv datasets generally show gradual change across the entire timeline. This indicates that in the UCI and ML-10M datasets, the interactions among different time slots show different patterns. Thus it is more important to conduct temporal alignment using temporal tokens which is consistent with our analysis for Table 4. For the Hepth and MMConv datasets, a simpler design for temporal alignment (e.g., same token or no token) is enough to achieve good performance.

E TIME COMPLEXITY ANALYSIS

The time complexity of our SimpleDyG model is the same as the vanilla Transformer, which is $O(n^2)$ where n is the sequence length. We conduct time efficiency experiments about the training time per epoch of different methods on UCI dataset using a machine with a NVIDIA L40 GPU with 64 CPU cores. The results show that our method trains faster than or comparable to all baselines. The methods integrating temporal modeling (e.g., RNN, self-attention) with structural modeling (e.g., GNN, GAT) suffer from the complexity of these modules.