Light Dynamic Graph Learning on Temporal Networks

ZHIQIANG PAN, National Key Laboratory of Information Systems Engineering, National University of Defense Technology, Changsha, China
CHEN GAO, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, China
FEI CAI and HONGHUI CHEN, National Key Laboratory of Information Systems Engineering, National University of Defense Technology, Changsha, China
YONG LI, Department of Electronic Engineering, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, China

Dynamic graph learning on temporal networks aims to understand the continuous evolution pattern of networks, with an important application on forecasting the future temporal network. Existing methods mainly focus on modeling the structural and temporal features, with recent research interest shifting toward considering the structural correlations between nodes through their neighbor co-occurrences. Though satisfactory performance has been achieved, there still remain several limitations: (1) the deviation of investigated scenarios from real-world applications, since most previous researches concentrate on special cases of multigraphs with abundant repeat edges; (2) the insufficient computational efficiency of modeling the structural features, since the existing neighbor co-occurrence scheme fails to consider explicit structural correlations between nodes and suffers from a time-consuming pairwise encoding strategy; (3) the unsatisfying prediction accuracy due to inadequate modeling of temporal features, since each neighbor's historical temporal features and the temporal domain shifting with network evolving are both neglected.

To solve these issues, we first focus on the general scenarios of temporal networks without abundant repeat edges for approaching the actual applications and propose an efficient and effective dynamic graph learning method named LightDyG. Specifically, (1) on the one hand, to increase the computational efficiency, LightDyG decouples the structural correlations between nodes and their individual substructures for fast convergence based on the analysis of existing co-occurrence mechanism, and further designs an incremental strategy for efficient structural encoding; (2) on the other hand, to improve the prediction accuracy, the temporal characteristics are considered by including both the interaction and appearance timestamps of neighbors, and a time-invariant temporal encoding strategy is designed to eliminate the temporal bias introduced by the network evolution. Extensive experiments conducted on four public temporal networks demonstrate that LightDyG outperforms the best baselines by 4.54–11.39% and 6.06–16.24% in terms of AP and AUC on the

This work was partially supported by the National Natural Science Foundation of China under Grant Nos. 62372458 and 62272262, and the Postgraduate Scientific Research Innovation Project of Hunan Province under Grant No. CX20230083.
Authors' Contact Information: Zhiqiang Pan, National Key Laboratory of Information Systems Engineering, National University of Defense Technology, Changsha, China; e-mail: panzhiqiang@nudt.edu.cn; Chen Gao, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, China; e-mail: chgao96@gmail.com; Fei Cai (corresponding author), National Key Laboratory of Information Systems Engineering, National University of Defense Technology, Changsha, China; e-mail: caifei08@nudt.edu.cn; Honghui Chen (corresponding author), National Key Laboratory of Information Systems Engineering, National University of Defense Technology, Changsha, China; e-mail: chenhonghui@nudt.edu.cn; Yong Li, Department of Electronic Engineering, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, China; e-mail: liyong07@tsinghua.edu.cn.

ACM 1558-2868/2025/8-ART138
https://doi.org/10.1145/3745024

temporal link prediction tasks, respectively. In addition, LightDyG reduces the time cost for training and test up to 45.91% and 63.94%, respectively, and also achieves a fast convergence speed during training. The implementation of our approach is available in https://github.com/nudtzpan/LightDyG.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Networks** → **Network algorithms**;

Additional Key Words and Phrases: Temporal network, dynamic graph learning, incremental structure encoding, time encoding

ACM Reference format:
Zhiqiang Pan, Chen Gao, Fei Cai, Honghui Chen, and Yong Li. 2025. Light Dynamic Graph Learning on Temporal Networks. *ACM Trans. Inf. Syst.* 43, 5, Article 138 (August 2025), 27 pages.
https://doi.org/10.1145/3745024

1 Introduction

Graph-structured data represent complex systems by regarding the contained elements as nodes and recording their interactions as edges [18, 44, 51, 53], which has been widely investigated in the static environments [14, 19, 42, 43]. Moreover, the formulated graph usually keeps evolving in real-world scenarios [10, 27, 46], forming temporal networks with recording the interactions between nodes with fine-grained timestamps [2, 6, 7, 17], which widely exist in applications such as recommender systems [45, 50, 59], social networks [20, 31, 49], and so on. Thus, dynamic graph learning on temporal networks is investigated to understand the network evolution pattern, which can be applied in various applications such as forecasting the future temporal network.

Existing methods for dynamic graph learning on temporal networks mainly focus on learning the representation of nodes by mapping their individual structural and temporal neighborhood node/edge information [5, 22, 55] into the embedding space [12, 28]. In addition, to consider the **structural correlations (SCs)** between nodes, recent researches propose the neighbor co-occurrence strategy of multi hops [23, 48] or single hop [38, 60], which regards the neighbors of two nodes as their intermediates to model the SCs between them.

However, though previous works have achieved satisfactory performance for dynamic graph learning on temporal networks, there still remain three main problems of two aspects, i.e., (1) the issue *Limited investigation of general temporal networks* in the aspect of investigated temporal network scenarios; and (2) the issues *Inefficient modeling of structural features* and *Inadequate modeling of temporal features* in the modeling process of features on temporal network:

— *Limited Investigation of General Temporal Networks.* Most previous researches investigate the special cases of temporal networks in real-world scenarios, i.e., temporal multigraphs wherein most edges are repeated [48, 55, 60]. These special cases simplify the dynamic graph learning task, leading the existing proposed methods to the deviation from the actual requirements of many other general temporal networks without the above-mentioned characteristic, i.e., without abundant repeat edges. Such existing researches may lead to two main shortcomings: (1) In many real-world scenarios like online question-answering networks and academic citation networks, interactions are usually sporadic yet significant. These temporal networks with few repeat edges are more sparse, differing from the abundant repeat interactions assumed in previous methods; (2) Existing methods heavily rely on the repeat edges with abundant dependencies, which limits their effectiveness in networks with sparse interactions. For example, they generally overlook critical but infrequent interactions, leading to suboptimal performance.

— *Inefficient Modeling of Structural Features.* Existing SC-based methods simply fuse the individual substructures of two nodes and their correlated connections in the structural encoding. Specifically, Wang et al. [48] first propose a **casual anonymous walk (CAW)** strategy to represent the node identities using the hitting counts of nodes in multi-hop neighbors, thus achieving inductive dynamic graph learning. Moreover, Yu et al. [60] and Tian et al. [38] simplify the CAW and design a **neighbor co-occurrence mechanism (NCM)** to merely take the one-hop historical neighbor sequence into consideration. However, the individual substructures of two nodes in a node pair and the correlated substructures of them are simply together considered in the structural encoding of existing methods. That is, they generally fail to explicitly consider the **correlated structural (CS)** connections between nodes, which makes the correlated structures hard to capture during training, thus leading to slow convergence of optimization. More importantly, the neighbor co-occurrence requires high computational cost because of its pairwise encoding strategy, making it unsuitable for real-world applications especially large-scale scenarios.
— *Inadequate Modeling of Temporal Features.* The encoding of time dynamics in current methods merely includes the interval between the future timestamp and the neighbor interaction timestamp to consider the decaying impact of various neighbors. Specifically, Xu et al. [55] first propose a **temporal graph attention (TGAT)** network, which designs a functional time encoding technique based on the classical Bochner's theorem [32], thus achieving continuous-time encoding. After that, most researches on dynamic graph learning generally follow TGAT to take its designed functional time encoding technique as a standard mechanism for time encoding [5, 15, 48, 60]. However, they neglect the interval between the interaction timestamp of each neighbor and its first appearance timestamp in temporal networks, which records the historical temporal characteristics. In addition, the continuously evolving network leads to the temporal domain shifting, decreasing the generalization ability of temporal encoding.

In order to solve the above issues, we focus on the general scenarios of temporal networks without abundant repeat edges and propose an efficient and effective incremental dynamic graph learning method named LightDyG on temporal networks. Specifically, (1) we first conduct an analysis on the existing NCM, illustrating that the neighbor co-occurrence can be decoupled to the individual structural characteristics of source/target nodes and the explicit SCs between them; (2) based on the above conclusion, we accelerate the training convergence by explicitly modeling the SCs between nodes and propose an incremental structure encoding strategy, which saves the computational cost of individual substructure encoding and the SC encoding between nodes; (3) in addition, we consider both the interaction timestamp of each neighbor and its first appearance timestamp in the network and design a time-invariant temporal encoding strategy to provide unbiased modeling of the time dynamics; (4) finally, we adopt a dual-channel Transformer to simultaneously model the individual temporal substructures of source/target nodes and their SCs, generating the temporal node representations for predicting future links. In addition, comprehensive experiments conducted on four real-world temporal networks demonstrate the superiority of LightDyG above the baselines in terms of both effectiveness and efficiency.

The contributions in this article can be summarized as follows:

(1) We point out the existing issue that the investigated scenarios are limited in multigraphs with abundant repeat edges in previous methods for dynamic graph learning on temporal networks. To the best of our knowledge, we are the first to investigate the general temporal networks without abundant repeat edges, which approach most real-world applications.
(2) To increase the model efficiency, we explicitly model the SCs between nodes for achieving fast convergence during training and design an incremental structural encoding mechanism

to save the computational cost of temporal structure encoding. To improve the model effectiveness, we simultaneously consider the interaction timestamp and appearance timestamp of each sampled historical neighbor, and further design a time-invariant temporal encoding strategy to eliminate the bias caused by the continuous network evolution.

(3) Extensive experiments conducted on four real-world temporal networks validate the effectiveness and efficiency of LightDyG. Notably, LightDyG achieves the performance gain up to 12.58% in the transductive dynamic graph learning setting and a higher maximum gain of 16.24% in the inductive setting, respectively. In addition, LightDyG saves the running time in each iteration for training and test up to 45.91% and 63.94%, respectively, with simultaneously achieving fast convergence during the training process.

The remaining part of this article is structured as follows: First, Section 2 reviews the related works in dynamic graph learning, providing a comprehensive overview of existing researches and recent developments. Then, in Section 3, we detail our proposed effective and efficient dynamic graph learning (LightDyG) method. Next, Section 4 focuses on the experimental settings prepared to validate our approach, and the findings and deep analysis are discussed in Section 5, where the experimental results are presented and interpreted. Finally, Section 6 concludes our research with a summary of the key findings and outlines the possible directions for future research.

2 Related Work

We review the related literature to our work from two main categories, i.e., **discrete-time dynamic graph (DTDG)** learning and **continuous-time dynamic graph (CTDG)** learning.

2.1 DTDG Learning

DTDG represents temporal networks as a series of graph snapshots by periodically recording the dynamic systems [13, 34, 52, 56], where the DTDG methods learn the evolution of temporal networks from the changes of structures across various snapshots [26, 30]. For instance, Zhou et al. [62] propose to consider a fundamental mechanism, i.e., the triad closure process, in the evolution of temporal networks for learning the embedding vectors of each node at various timesteps. Moreover, Singer et al. [35] design a semi-supervised approach tNodeEmbed, which is initialized using static node embeddings and learns the network dynamics by aligning the temporal node embeddings at different timesteps. Furthermore, Yang et al. [57] propose to exploit the exponential capacity and hierarchical awareness of hyperbolic geometry for simultaneously capturing the evolving behaviors and preserving the hierarchical information in an implicit way. In addition, several works also pay attention to the computational efficiency of DTDG learning. For example, Sankar et al. [34] propose Dynamic Self-Attention Network that utilizes stacked structural and temporal self-attention layers for efficient dynamic node representation learning compared to the sequential solutions. In addition, Zheng et al. [61] propose Instant Graph Neural Network which computes the node representation matrix of graphs incrementally for saving the repetitive computation time.

However, the DTDGs represent the dynamic systems using the periodical graph snapshots, which cannot record the fine-grained interaction timestamps of each temporal edge. Thus, they fail to provide timely prediction of the temporal network evolution, which deviates from the realistic scenarios.

2.2 CTDG Learning

Different from DTDG, the CTDG records the specific interaction timestamps of the continuous edges in temporal networks [11, 15, 58, 63]. Early methods mainly focus on learning the temporal neighborhood features brought by nodes and edges [5, 33, 40, 55]. For example, Xu et al. [55] introduce the **TGAT** layer, which efficiently aggregates the temporal-topological neighborhood

information as well as the time features of interactions using the **self-attention mechanism (SAT)** [41]. Moreover, Tian et al. [37] propose a self-supervised representation learning framework on temporal networks based on contrastive learning [3, 9]. In addition, Cong et al. [5] design a simple but effective architecture GraphMixer which summarizes the one-hop temporal neighbors using the **multi-layer perception (MLP)**-mixer [39].

Moreover, to take the structure characteristics of temporal graphs into consideration, Wang et al. [48] design a CAW strategy to model the network motifs [24] between source node and target node in a neural way for estimating the correlations between them. Moreover, Jin et al. [15] propose a sptaiotemporal-biased random walk method to sample diverse and expressive walks for extracting the node substructure and adopt the neural ordinary differential equations [4] for modeling the irregularly sampled temporal nodes. In addition, Yu et al. [60] introduce a unified library DyGlib, where a simple one-hop neighbor aware method DyGFormer is also designed using a neighbor co-occurrence scheme for encoding and a patching technique for learning from long historical sequences, respectively.

However, on the one hand, the feature-aware methods fail to take the structure characteristics of source/target nodes into consideration, limiting their ability of accurately predicting future links. On the other hand, the structure-enhanced methods learn both the **individual structure (IS)** characteristics of source/target nodes and their SCs in a fused NCM, which fails to consider the SCs between source and target nodes explicitly. In addition, the structural features of nodes can only be encoded in a pairwise way with large computational cost in the existing neighbor co-occurrence encoding scheme, making it hard for practical applications.

3 Methodology

3.1 Problem Statement and Overview

3.1.1 Problem Statement.

Temporal Network. We first give the formulation of our investigated temporal networks. Specifically, we focus on the continuous-time temporal networks, which can be represented as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ and $\mathcal{E}$ are the constantly increasing nodes and evolving edges in $\mathcal{G}$, respectively. $(v_s, v_t, t_{st}) \in \mathcal{E}$ indicates a temporal edge connecting source nodes v_s and target node v_t at the timestamp t_{st}. In addition, we denote the timestamp of each node v_x's first appearance in the temporal network as t_x^0.

Problem Formalization. Dynamic graph learning aims to generate the temporal representations of nodes in the current timestamp by modeling the evolution pattern of temporal networks. In this article, we evaluate the temporal representation learning ability through the fundamental problem for forecasting the network evolution, i.e., temporal link prediction. In detail, given the observed temporal network $\mathcal{G}_{cur} = \{\mathcal{V}_{cur}, \mathcal{E}_{cur}\}$ before the current timestamp t_{cur}, this task can be formulated as predicting future edges that may appear in the future temporal network, i.e., $\{(v_s, v_t, t_{st}) \in \mathcal{E} \setminus \mathcal{E}_{cur} | t_{st} > t_{cur}\}$. In addition, in the scenarios of temporal multigraphs investigated in previous works [48, 55, 60], edges are highly repeated, where the weight of each edge (i.e., the total number of interactions between each node pair v_s and v_t) can be described as $w(v_s, v_t) \gg 1$. Differently, we focus on the general temporal networks without abundant repeat edges, where the edge weight satisfies $w(v_s, v_t) \approx 1$ by comparing with the temporal multigraphs investigated in previous researches.

The main notations used in this article are listed in Table 1.

3.1.2 Overview. The pipeline of our proposed light dynamic graph learning (LightDyG) method is presented in Figure 1, including the following three main components:

— *Efficient structural encoding*, where we first design an incremental strategy for efficiently encoding the node substructure reflected by the consistently increasing historical neighbor

Table 1. Frequently Used Notations in This Article

Notation	Description
$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$	A temporal network with its contained nodes and edges
(v_s, v_t, t_{st})	A temporal edge connecting source node v_s and target node v_t at timestamp t_{st}
$\mathcal{N}_{v_x}$	The one-hop historical temporal neighbors of node v_x
$\mathcal{F}_{NCM}$	The function for considering the neighbor co-occurrence between two nodes
$g(v_i, \mathcal{N}_{v_x})$	The function for counting v_i's occurrences in v_x's substructure
$h(v_x)$	The neighbor occurrence vector of node v_x
$\mathbf{s}_i$	The IS embedding generated for neighbor v_i
$\mathbf{c}_i$	The correlated structure embedding generated for neighbor v_i
$\bar{\mathbf{t}}_i$	The interaction time encoding vector generated for neighbor v_i
$\hat{\mathbf{t}}_i$	The appearance time encoding vector generated for neighbor v_i
$\mathbf{z}_i$	The individual combined representation of v_i
$\mathbf{h}^x_{\text{ITS}}$	The individual temporal and structural representation of node v_x
$\mathbf{h}^x_{\text{CS}}$	The CS representation of node v_x
$\mathbf{f}_x$	The final representation generated for node v_x
y_{st}	The prediction score between source node and target node
$\mathcal{L}$	The cross-entropy loss for optimizing the trainable parameters

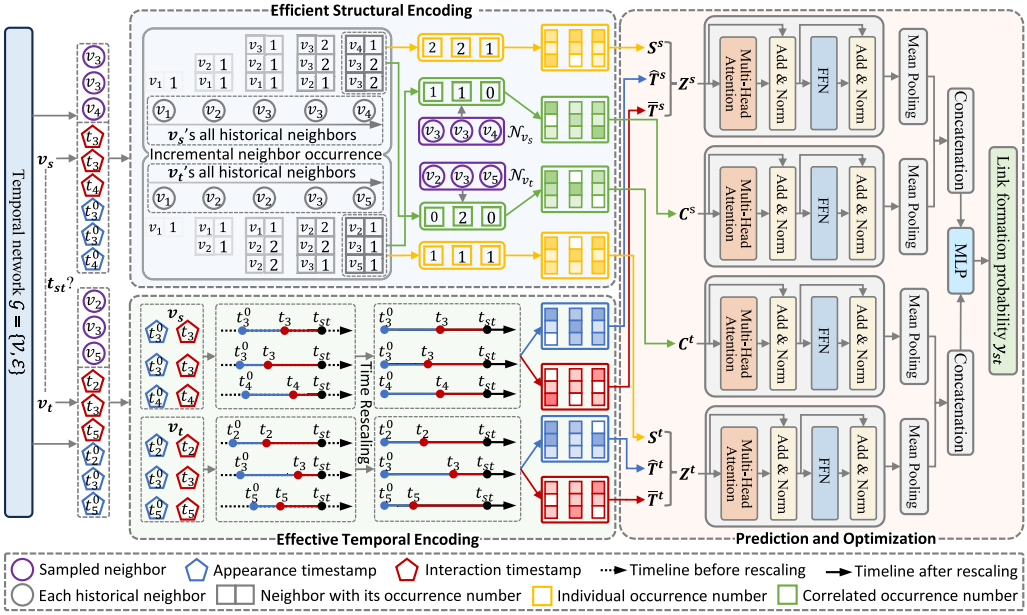


Fig. 1. Framework of our proposed LightDyG. The model inputs include source node v_s and target node v_t with their structural features and temporal features, which are modeled by the components of efficient structural encoding and the effective temporal encoding, respectively. Then, future link predictions are generated by combining the obtained structural and temporal representations of v_s and v_t.

sequences and then model the ISs of source/target nodes and consider the correlated structures between them.

— *Effective temporal encoding*, where we model the temporal characteristics by including both the interaction timestamp and appearance timestamp of the historical neighbors, and further

design a time-invariant temporal encoding method for dealing with the temporal bias caused by the continuous network evolution.
— *Prediction and optimization*, where we adopt a dual-channel Transformer for learning the individual temporal and correlated representations, respectively, and then make future link predictions by combining the representation of source/target nodes, which is then inputted to an MLP for generating the link formation probability.

3.2 Efficient Structural Encoding

3.2.1 NCM. The NCM models the SCs between nodes by modeling the extracted temporal substructures of source node v_s and target node v_t. Specifically, given the anchor node $v_x \in \{v_s, v_t\}$ in the temporal network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, the temporal substructure of v_x can be extracted by backtracking the interaction timestamps, i.e., sampling previously interacted nodes. Specifically, the sampled M-many L-length temporal neighbor walks of $v_x \in \{v_s, v_t\}$ can be denoted as $\mathcal{N}_{v_x} = \{\mathcal{N}^1_{v_x}, \mathcal{N}^2_{v_x}, \ldots, \mathcal{N}^L_{v_x}\}$, where $\mathcal{N}^l_{v_x} = \{v^l_1, v^l_2, \ldots, v^l_M\}$ is the sampled neighbors of the l-hop for the anchor node v_x.

Then, the NCM proposes to estimate the correlations between source node v_s and target node v_t by regarding each neighbor $v_i \in \mathcal{N}_{v_s} \cup \mathcal{N}_{v_t}$ as the intermediate node between v_s and v_t. Specifically, the encoding process of each neighbor $v_i \in \mathcal{N}_{v_s} \cup \mathcal{N}_{v_t}$ in the *NCM* can be formulated as follows:

$$\mathbf{a}_{v_i} = \mathcal{F}_{NCM}(v_i, \mathcal{N}_{v_s}, \mathcal{N}_{v_t}) = [g(v_i, \mathcal{N}_{v_s}), g(v_i, \mathcal{N}_{v_t})], \tag{1}$$

where $\mathbf{a}_{v_i} \in \mathbb{R}^{2L}$ is the encoding vector of neighbor v_i, $\mathcal{F}_{NCM}$ denotes the function of the NCM, consisting of considering v_i's correlations with the substructures $\mathcal{N}_{v_s}$ and $\mathcal{N}_{v_t}$ of v_s and v_t, respectively. In addition, g indicates the function of modeling the correlations between neighbor v_i and the neighbors of node $v_x \in \{v_s, v_t\}$, which can be defined as follows:

$$g(v_i, \mathcal{N}_{v_x})[l] = |\{v_j | v_j \in \mathcal{N}^l_{v_x}, v_i = v_j\}|, \forall l = 1, \ldots, L, \tag{2}$$

where the function g counts the number of neighbor v_i's appearing times at different positions away from v_x in the substructure of v_x. In general, Equation (2) computes v_i's occurrence in the individual substructures $\mathcal{N}_{v_s}$ and $\mathcal{N}_{v_t}$ of v_s and v_t, respectively, which are then combined by Equation (1) to take the co-occurrence of v_i in both $\mathcal{N}_{v_s}$ and $\mathcal{N}_{v_t}$ into consideration.

In addition, recent researches demonstrate that modeling only the first-hop historical neighbors can achieve approximate performance to considering multi-hop neighbors [38, 60]. Then, the temporal neighbors of v_s and v_t degenerate to $\mathcal{N}_{v_s} = \{v_1, v_2, \ldots, v_{M_s}\}$ and $\mathcal{N}_{v_t} = \{v_1, v_2, \ldots, v_{M_t}\}$, respectively. Moreover, Equation (1) remains the same while Equation (2) becomes:

$$g(v_i, \mathcal{N}_{v_x}) = |\{v_j | v_j \in \mathcal{N}_{v_x}, v_i = v_j\}|, \tag{3}$$

Analysis on Neighbor Co-Occurrence: Though the NCM considers the SCs between source node and target node, they simply fuse (1) the individual substructure features of source/target nodes and (2) the SCs driven by the overlapping part of the neighbors (i.e., the common neighbors) of source/target nodes. Thus, though satisfactory performance has been achieved, there still remain two main disadvantages:

— On the one hand, for both source node v_s and target node v_t, the IS and the cross-correlations are simply fused in the NCM, where the SCs between v_s and v_t cannot be emphasized in an explicit way.
— On the other hand, both the occurrence encoding $\{g(v_i, \mathcal{N}_{v_s}) | v_i \in \mathcal{N}_s\}$ and $\{g(v_i, \mathcal{N}_{v_t}) | v_i \in \mathcal{N}_{v_t}\}$ require to be computed for each temporal edge, resulting in large computational cost, especially on large-scale temporal networks.

3.2.2 Incremental Neighbor Occurrence. We first conduct deep analysis on the existing NCM. Specifically, in the neighbor co-occurrence computing, when each neighbor of v_s, i.e., $v_i \in \mathcal{N}_{v_s}$, is taken as the intermediate node between v_s and v_t:

— The former part in Equation (1), i.e., $\{g(v_i, \mathcal{N}_{v_s})|v_i \in \mathcal{N}_{v_s}\}$, models the IS characteristics of v_s brought by its historical neighbors $\mathcal{N}_{v_s}$;
— The latter part in Equation (1), i.e., $\{g(v_i, \mathcal{N}_{v_t})|v_i \in \mathcal{N}_{v_s}\}$, captures the cross-SCs between v_s and v_t reflected by the common neighbors $\mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}$.

Similarly, for the neighbors $\mathcal{N}_{v_t}$ taken as the intermediate nodes, the former part $\{g(v_i, \mathcal{N}_{v_s})|v_i \in \mathcal{N}_{v_t}\}$ captures the cross-SCs between v_s and v_t while the latter part $\{g(v_i, \mathcal{N}_{v_t})|v_i \in \mathcal{N}_{v_t}\}$ models the IS characteristics of v_t. In general, the NCM can be decoupled into (1) $\{g(v_i, \mathcal{N}_{v_s})|v_i \in \mathcal{N}_{v_s}\}$ and $\{g(v_i, \mathcal{N}_{v_t})|v_i \in \mathcal{N}_{v_t}\}$, which model the IS characteristics of v_s and v_t, respectively, and (2) $\{g(v_i, \mathcal{N}_{v_t})|v_i \in \mathcal{N}_{v_s}\}$ and $\{g(v_i, \mathcal{N}_{v_s})|v_i \in \mathcal{N}_{v_t}\}$, which consider the cross-SCs between v_s and v_t.

Based on the above analysis, we design an incremental neighbor occurrence counting method for saving the encoding time of the IS modeling of v_s/v_t and accelerating that of modeling the SCs between v_s and v_t. Specifically, in the NCM, the pointwise IS of a single node $v_x \in \{v_s, v_t\}$ (i.e., $\{g(v_i, \mathcal{N}_{v_s})|v_i \in \mathcal{N}_{v_s}\}$ and $\{g(v_i, \mathcal{N}_{v_t})|v_i \in \mathcal{N}_{v_t}\}$) can be computed in an incremental way by making minor modification on the existing structure encoding vector. In detail, given node v_x with its neighbors denoted as $\mathcal{N}_{v_x}$, the neighbor occurrence vector of v_x can be formulated as follows:

$$h(v_x) = \{g(v_i, \mathcal{N}_{v_x})|\forall v_i \in \mathcal{N}_{v_x}||t_1 < t_2 < \cdots < t_M\}, \tag{4}$$

where $h(v_x) \in \mathbb{R}^M$ is the generated occurrence vector of node v_x, in which the occurrence numbers are arranged according to the ascending order of interaction timestamps, i.e., $t_1 < t_2 < \cdots < t_M$.

Then, when a new interaction happens between node v_x and neighbor v_j, the neighbor occurrence vector of v_x can be simply modified by the following three steps:

Existing Neighbor Modification. First, for the contained neighbor v_i, the occurrence number can be calculated as follows:

$$g(v_i, \mathcal{N}_{v_x}) = \begin{cases} g(v_i, \mathcal{N}_{v_x}) + 1, & \forall v_i = v_j, \\ g(v_i, \mathcal{N}_{v_x}), & \forall v_i \neq v_j, \end{cases} \tag{5}$$

which indicates that the occurrence number of the same historical neighbors to the current neighbor v_j is increased by one, and it remains unchanged for the other historical neighbors.

New Neighbor Padding. Then, we append the occurrence number of the new interacted neighbor to the encoding vector, which can be formulated as follows:

$$g(v_j, \mathcal{N}_{v_x}) = \begin{cases} g(v_i, \mathcal{N}_{v_x}), & \exists v_i = v_j, \\ 1, & \exists v_i \neq v_j, \end{cases} \tag{6}$$

which indicates that the occurrence number of the historically interacted neighbors is directly adopted, and the occurrence number is assigned to 1 for the newly appearing neighbors.

FIFO Strategy. In addition, similar to the recent sampling strategy which emphasizes the recently interacted neighbors, here we adopt a FIFO strategy to abandon the outdated neighbors when the encoding vector length exceeds the set maximum neighbor number. Specifically, setting the maximum number as M, when the neighbor sequence length exceeds M, the first-in neighbor in the neighbor sequence to be removed with index $j - M$ can be denoted as v_{j-M}. That is, $g(v_{j-M}, \mathcal{N}_{v_x})$ is removed from $h(v_x) \in \mathbb{R}^{M+1}$, making the dimension of $h(v_x)$ changing from $h(v_x) \in \mathbb{R}^{M+1}$ to $h(v_x) \in \mathbb{R}^M$. Next, the occurrence numbers for the remained neighbors can be changed as follows:

$$g(v_i, \mathcal{N}_{v_x}) = \begin{cases} g(v_i, \mathcal{N}_{v_x}) - 1, & \forall v_i = v_{j-M}, \\ g(v_i, \mathcal{N}_{v_x}), & \forall v_i \neq v_{j-M}, \end{cases} \tag{7}$$

Algorithm 1: Procedure of the Incremental Neighbor Occurrence

Input: $\mathcal{N}_{v_x} = \{v_1, v_2, \ldots, v_{M_x}\}$: The historical temporal neighbor sequence;
Output: $h(v_x) = \{g(v_i, \mathcal{N}_{v_x}) | \forall v_i \in \mathcal{N}_{v_x}\}$: The generated neighbor occurrence vector for v_x;

1: **for** j in range($|\mathcal{N}_{v_x}|$) **do**
2: $\quad \Phi_j^+ \leftarrow$ search where $h(v_x) \in \mathbb{R}^j$ equals to v_j;
3: $\quad$ **for** i in Φ_j^+ **do**
4: $\quad\quad g(v_i, \mathcal{N}_{v_x}) = g(v_i, \mathcal{N}_{v_x}) + 1$;
5: $\quad$ **end for**
6: $\quad$ **if** $\exists v_i$ equals to v_j **then**
7: $\quad\quad g(v_j, \mathcal{N}_{v_x}) = g(v_i, \mathcal{N}_{v_x})$;
8: $\quad$ **else**
9: $\quad\quad g(v_j, \mathcal{N}_{v_x}) = 1$;
10: $\quad$ **end if**
11: $\quad$ **if** $j + 1 > M$ **then**
12: $\quad\quad h(v_x) = h(v_x)[-M:]$
13: $\quad\quad \Phi_j^- \leftarrow$ search where $h(v_x) \in \mathbb{R}^M$ equals to v_{j-M};
14: $\quad\quad$ **for** i in Φ_j^- **do**
15: $\quad\quad\quad g(v_i, \mathcal{N}_{v_x}) = g(v_i, \mathcal{N}_{v_x}) - 1$;
16: $\quad\quad$ **end for**
17: $\quad$ **end if**
18: **end for**
19: **return** $h(v_x) = \{g(v_i, \mathcal{N}_{v_x}) | \forall v_i \in \mathcal{N}_{v_x}\}$.

which indicates that the occurrence number of the same historical neighbors to the first-in neighbor is decreased by one, and it remains unchanged for the other historical neighbors.

To summarize, our designed incremental neighbor occurrence strategy efficiently generates the pointwise individual neighbor occurrences of nodes in an incremental way by relying on the historically computed neighbor occurrence. Thus, the computational cost is largely saved compared to the existing pairwise neighbor occurrence mechanism, where the neighbor occurrences need to be computed between each node pair. In addition, to clearly illustrate the incremental neighbor occurrence, we present its detailed procedure in Algorithm 1. Specifically, given node v_x with neighbors $\mathcal{N}_{v_x}$, when each neighbor $v_j \in \mathcal{N}_{v_x}$ is interacted, the existing neighbor modification is conducted in lines 2–5, the new neighbor padding is achieved in lines 6–9, and the FIFO strategy is adopted in lines 11–17 if the neighbor sequence length exceeds the maximum number.

In addition, to clearly introduce the incremental neighbor occurrence mechanism, we take the encoding process of source node v_s in Figure 1 as an example for illustration. Specifically, when the fourth neighbor v_3 is interacted with v_s, the fourth neighbor v_3 joins v_s's existing historical neighbors $\{v_1, v_2, v_3\}$, leading v_s's current neighbors to $\{v_1, v_2, v_3, v_3\}$. Then, the neighbor occurrences of v_s is calculated by three steps: (1) First, with the new fourth neighbor v_3 joining, the occurrence number of the existing third neighbor v_3 is increased to 2, leading v_s's neighbor occurrences to $\{1, 1, 2\}$; (2) Then, the occurrence number of the new fourth neighbor v_3 is padded to v_s's occurrence numbers, which is 2 considering that v_3 has appeared in the existing occurrence numbers (i.e., the third number 2 in {1, 2, 2}). Thus, the occurrence numbers of v_s become $\{1, 1, 2, 2\}$; (3) Finally, considering that the length of v_s's occurrence numbers exceeds the maximum number (i.e., 3 in the example presented in Figure 1), the earliest occurrence number (i.e., the occurrence number 1 of the first neighbor v_1) is removed from the occurrence numbers. Moreover, since the removed neighbor v_1 is

not included in the remaining neighbors, the remaining occurrence numbers are unchanged, thus the final occurrence numbers of v_s become $\{1, 2, 2\}$.

3.2.3 IS Embedding. After encoding the individual temporal substructures of source node v_s and target node v_t in an incremental way, we generate the IS embeddings for v_s and v_t. Specifically, for $v_x \in \{v_s, v_t\}$, after encoding the substructure of v_x, an occurrence number $g(v_i, \mathcal{N}_{v_x})$ is assigned to each neighbor $v_i \in \mathcal{N}_{v_x}$. For example, as shown in Figure 1, the occurrence numbers of v_s and v_t are calculated as $\{2, 2, 1\}$ and $\{1, 1, 1\}$, respectively. After that, we first generate an embedding vector for $g(v_i, \mathcal{N}_{v_x})$ by inputting it to an MLP for modeling the *IS* of v_x as:

$$\mathbf{s}_i^x = \text{MLP}_{\text{IS}}(g(v_i, \mathcal{N}_{v_x})) = \mathbf{W}_{\text{IS}}^2(\text{ReLU}(\mathbf{W}_{\text{IS}}^1 g(v_i, \mathcal{N}_{v_x}) + \mathbf{b}_{\text{IS}}^1)) + \mathbf{b}_{\text{IS}}^2, \tag{8}$$

where $\mathbf{s}_i \in \mathbb{R}^d$ is the generated d-dimensional IS vector of v_i, $\mathbf{W}_{\text{IS}}^1 \in \mathbb{R}^{1\times d}$ and $\mathbf{W}_{\text{IS}}^2 \in \mathbb{R}^{d\times d}$ are the trainable parameters, $\mathbf{b}_{\text{IS}}^1, \mathbf{b}_{\text{IS}}^2 \in \mathbb{R}^d$ are the biased vectors, and ReLU is the activation function. Finally, we can obtain the IS embeddings of v_x's neighbors as $\mathbf{S}^s = \{\mathbf{s}_1^x, \mathbf{s}_2^x, \ldots, \mathbf{s}_{M_x}^x\}$, where M_x is the number of node v_x's neighbors.

3.2.4 Correlated Structure Embedding. In addition to the IS embedding, we propose to take the SCs between source node v_s and target node v_t into consideration for generating their respective correlated structure embeddings. Specifically, different from the IS embedding module, here we enhance the representation learning of nodes v_s/v_t using the co-occurrence brought by the common neighbors $\mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}$.

For example, as shown in Figure 1, the occurrence numbers of v_s and v_t are first calculated as $\{2, 2, 1\}$ and $\{1, 1, 1\}$, respectively. Then, given the common neighbor set of v_s and v_t is $\{v_3\}$, the co-occurrence number of v_3 in $\mathcal{N}_{v_s}$ is generated as the occurrence number of v_3 in v_t's occurrence numbers, i.e., 1. Moreover, considering that v_4 does not exist in $\mathcal{N}_{v_t}$, the co-occurrence number of v_4 in $\mathcal{N}_{v_s}$ is assigned as 0. Thus, the co-occurrence numbers of v_s can be generated as $\{1, 1, 0\}$. Similarly, the co-occurrence numbers of v_t can be generated as $\{0, 2, 0\}$.

Then, taking the source node v_s as an example, for the co-occurrence number $g(v_i, \mathcal{N}_{v_t})$ of each neighbor $v_i \in \mathcal{N}_{v_s}$ which belongs to $\mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}$, i.e., $v_i \in \mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}$, we first embed it to a d-dimensional vector as by a two-layer MLP for modeling the SC as follows:

$$\mathbf{c}_i^s = \text{MLP}_{\text{SC}}(g(v_i, \mathcal{N}_{v_t})) = \mathbf{W}_{\text{SC}}^2(\text{ReLU}(\mathbf{W}_{\text{SC}}^1 g(v_i, \mathcal{N}_{v_t}) + \mathbf{b}_{\text{SC}}^1) + \mathbf{b}_{\text{SC}}^2), \tag{9}$$

where $\mathbf{c}_i \in \mathbb{R}^d$ is the generated SC enhanced vector of neighbor v_i, $\mathbf{W}_{\text{SC}}^1 \in \mathbb{R}^{1\times d}$ and $\mathbf{W}_{\text{SC}}^2 \in \mathbb{R}^{d\times d}$ are the trainable matrices, $\mathbf{b}_{\text{SC}}^1, \mathbf{b}_{\text{SC}}^2 \in \mathbb{R}^d$ are the biased vectors, and ReLU is the activation function. Finally, we can obtain the correlated structure embeddings of source node v_s's neighbors as $\mathbf{C}^s = \{\mathbf{c}_1^s, \mathbf{c}_2^s, \ldots, \mathbf{c}_{M_s'}^s\}$, where M_s' is the number of neighbors belonging to the common neighbors $\mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}$ of v_s and v_t. Similarly, the co-occurrence embeddings of neighbors of the target node v_t can be obtained as $\mathbf{C}^t = \{\mathbf{c}_1^t, \mathbf{c}_2^t, \ldots, \mathbf{c}_{M_t'}^t\}$.

3.3 Effective Temporal Encoding

3.3.1 Comprehensive Time Encoding. In addition to the structure encoding, we further consider the temporal characteristics brought by the interaction timestamp of each temporal edge. In detail, given a neighbor $v_i \in \mathcal{N}_{v_x}$ for $v_x \in \{v_s, v_t\}$, we propose to comprehensively take the interaction timestamp t_i with neighbor v_i and its appearing timestamp t_i^0 in the temporal network into consideration, which we denote as interaction time and appearance time, respectively.

Specifically, as shown in Figure 1, the appearance timestamps are marked as blue pentagons and blue points, while the interaction timestamps are marked in red, and the future timestamps are plotted as black points. Then, two aspects of temporal features, i.e., (1) the blue lines which

represent the intervals between the neighbor interaction timestamps and the neighbor appearance timestamps and (2) the red lines which represent the intervals between the future timestamps and the neighbor interaction timestamps, are utilized for generating the time encoding vectors.

Interaction Time Encoding. On the one hand, to take the decaying contribution of each interacted neighbor into consideration, we model the interaction time by encoding the time interval $\Delta\bar{t} = t_{st} - t_i$ between the future timestamp t_{st} and the interaction timestamp t_i, which can be formulated as:

$$\bar{\mathbf{t}}_i = \mathcal{F}_{time}(t_{st} - t_i) = \mathcal{F}_{time}(\Delta\bar{t}_i), \tag{10}$$

where $\bar{\mathbf{t}}_i \in \mathbb{R}^d$ is the encoding vector of the interaction time, d is the vector dimension. Finally, we can obtain the interaction time embeddings of v_x's neighbors as $\bar{\mathbf{T}}^s = \{\bar{\mathbf{t}}_1^x, \bar{\mathbf{t}}_2^x, \ldots, \bar{\mathbf{t}}_{M_x}^x\}$, where M_x is the number of node v_x's neighbors. In addition, $\mathcal{F}_{time}$ is defined by utilizing the random Fourier features following [48, 55, 60], which can be formulated as follows:

$$\begin{aligned} \mathbf{t}_i &= \mathcal{F}_{time}(\Delta t_i) \\ &= [cos(\omega_1 \Delta t_i), sin(\omega_1 \Delta t_i), \ldots, cos(\omega_d \Delta t_i), sin(\omega_d \Delta t_i)], \end{aligned} \tag{11}$$

where $[\omega_1, \omega_2, \ldots, \omega_d]$ are the trainable parameters in the time encoding function $\mathcal{F}_{time}$.

Appearance Time Encoding. On the one hand, to take the historically accumulated temporal characteristics of each neighbor into encoding, we further consider the appearance time by taking the time interval $\Delta\hat{t} = t_i - t_i^0$ between the interaction timestamp t_i and the appearance time t_i^0 of v_i into consideration. This can be formulated as follows:

$$\hat{\mathbf{t}}_i = \mathcal{F}_{time}(t_i - t_i^0) = \mathcal{F}_{time}(\Delta\hat{t}_i), \tag{12}$$

where $\hat{\mathbf{t}}_i \in \mathbb{R}^d$ is the encoding vector of the appearance time, and d is the vector dimension. Finally, the appearance time embeddings of v_x's neighbors can be generated as $\hat{\mathbf{T}}^s = \{\hat{\mathbf{t}}_1^x, \hat{\mathbf{t}}_2^x, \ldots, \hat{\mathbf{t}}_{M_x}^x\}$, where M_x is the number of node v_x's neighbors.

3.3.2 Time Rescaling. In addition, considering that the temporal network keeps evolving with continuously appearing nodes, the time scale of different nodes existing in temporal network may introduce bias to the temporal encoding. Specifically, the temporal features of neighbor v_i can be considered from two aspects in our proposed LightDyG: (1) the time interval $\Delta\bar{t}_i = t_{st} - t_i$ denotes the interval between the future timestamp t_{st} and the neighbor interaction timestamp t_i; (2) the time interval $\Delta\hat{t}_i = t_i - t_i^0$ indicates the interval between the neighbor interaction timestamp t_i and the neighbor appearance timestamp t_i^0. Then, the temporal bias can be described as that with the temporal network evolving, the average intervals $\Delta\bar{t}_i$ and $\Delta\hat{t}_i$ for modeling the temporal features of nodes significantly change. This leads to a problem that the learned time encoding function in the historical time stages cannot well adapt to the future time stages for making accurate predictions.

Thus, to eliminate the bias caused by the various time scales of nodes existing in temporal network, we propose a simple but effective time-invariant temporal encoding strategy by rescaling the corresponding time scales of different nodes. Specifically, for both the interaction and appearance time encodings, we rescale $\Delta\bar{t}$ and $\Delta\hat{t}$ through dividing them by the time interval between the neighbor appearance and the future timestamp, i.e., $t_{st} - t_i^0$. For example, as shown in the timelines before time rescaling in Figure 1, the appearance timestamps of different neighbors are various. After the time rescaling operation, the time intervals between the future timestamps and the appearance timestamps of different neighbors are normalized as identical.

That is, Equations (10) and (12) are modified as follows:

$$\bar{\mathbf{t}}_i = \mathcal{F}_{time}(\lambda(t_{st} - t_i)/(t_{st} - t_i^0)) = \mathcal{F}_{time}(\Delta\bar{t}_i'), \tag{13}$$

$$\hat{\mathbf{t}}_i = \mathcal{F}_{time}(\lambda(t_i - t_i^0)/(t_{st} - t_i^0)) = \mathcal{F}_{time}(\Delta\hat{t}_i'), \tag{14}$$

where $\bar{\mathbf{t}}_i, \hat{\mathbf{t}}_i \in \mathbb{R}^d$ are the improved interaction and appearance time embeddings, respectively, and λ is a hyper-parameter for controlling the rescaled time scale of different nodes.

3.4 Prediction and Optimization

3.4.1 Dual-Channel Transformer. After generating the structural and temporal embeddings of source node v_s and target node v_t, we propose to input them into the Transformer for learning better representations of v_s and v_t.

Specifically, on the one hand, for node $v_x \in \{v_s, v_t\}$, as shown in Figure 1, we concatenate the IS embedding $\mathbf{s}_i$ as well as the interaction time embedding $\bar{\mathbf{t}}_i$ and the appearance time embedding $\hat{\mathbf{t}}_i$ of each neighbor $v_i \in \mathcal{N}_{v_x}$, which can be formulated as follows:

$$\mathbf{z}_i = [\mathbf{s}_i, \bar{\mathbf{t}}_i, \hat{\mathbf{t}}_i], \tag{15}$$

where $\mathbf{z}_i \in \mathbb{R}^{3d}$ is the individual combined representation of neighbor v_i, and $[,]$ indicates the concatenation operation. Thus, the individual combined representations of neighbors of node $v_x \in \{v_s, v_t\}$ can be formulated as $\mathbf{Z}^x = \{\mathbf{z}_1^x, \mathbf{z}_2^x, \ldots, \mathbf{z}_{M_x}^x\}$.

On the other hand, for node $v_x \in \{v_s, v_t\}$, the embeddings generated by Equation (9), i.e., $\mathbf{C}^x = \{\mathbf{c}_1^x, \mathbf{c}_2^x, \ldots, \mathbf{c}_{M'_x}^x\}$, are deemed as the CS representations of v_x.

Moreover, considering that (1) the individual neighbor representation dimension (i.e., $3d$) and the correlated neighbor representation dimension (i.e., d) are different and (2) the individual neighbor numbers (i.e., $M_s = |\mathcal{N}_{v_s}|$ and $M_t = |\mathcal{N}_{v_t}|$) and the correlated neighbor numbers (i.e., $M'_s = M'_t = |\mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}|$) are also different, for neighbor v_i, the individual combined representation (i.e., $\mathbf{z}_i$) and the CS representation (i.e., $\mathbf{c}_i$) cannot be directly concatenated. Thus, here we adopt a dual-channel Transformer, which assigns two different Transformers for modeling the individual neighbors of source node/target node and the correlated neighbors of them, respectively.

Specifically, for learning better representations of $v_x \in \{v_s, v_t\}$, we input (1) the individual combined representations $\mathbf{Z}^x = \{\mathbf{z}_1^x, \mathbf{z}_2^x, \ldots, \mathbf{z}_{M_x}^x\}$ and (2) the SC enhanced representations $\mathbf{C}^x = \{\mathbf{c}_1^x, \mathbf{c}_2^x, \ldots, \mathbf{c}_{M'_x}^x\}$ into two parallel Transformers, respectively, followed by a mean pooling for generating v_x's final *individual temporal and structural (ITS)* representation $\mathbf{h}_{\text{ITS}} \in \mathbb{R}^{3d}$ and *CS* representation $\mathbf{h}_{\text{SC}} \in \mathbb{R}^d$. This can be formulated as:

$$\begin{aligned} \mathbf{H}_{\text{ITS}}^x &= \mathbf{W}_{\text{ITS}}(\mathcal{F}_{mean}(\text{Transformer}_{ITS}(\mathbf{Z}^x)) + \mathbf{b}_{\text{ITS}}, \\ \mathbf{H}_{\text{CS}}^x &= \mathbf{W}_{\text{CS}}(\mathcal{F}_{mean}(\text{Transformer}_{CS}(\mathbf{C}^x)) + \mathbf{b}_{\text{CS}}, \end{aligned} \tag{16}$$

where $\mathbf{W}_{\text{ITS}} \in \mathbb{R}^{3d\times 3d}$, $\mathbf{W}_{\text{SC}} \in \mathbb{R}^{d\times d}$, and $\mathbf{b}_{\text{ITS}} \in \mathbb{R}^{3d}$, $\mathbf{b}_{\text{SC}} \in \mathbb{R}^d$ are the trainable parameters. Moreover, as shown in Figure 1, the Transformer includes the components of **multi-head self-attention (MSA)**, **feed-forward network (FFN)**, Layer Normalization, and Dropout, which can be formulated as follows by taking Transformer_{ITS} as an example:

$$\begin{aligned} \mathbf{Q} &= \mathbf{W}_q\mathbf{Z}^x, \mathbf{K} = \mathbf{W}_k\mathbf{Z}^x, \mathbf{V} = \mathbf{W}_v\mathbf{Z}^x, \\ \mathbf{X} &= \text{MSA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}, \\ \mathbf{O} &= \text{LayerNorm}(\mathbf{S}_{st} + \text{Dropout}(\mathbf{X})), \\ \mathbf{O}' &= \text{FFN}(\mathbf{X}) = \mathbf{W}_2(\text{GELU}(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1)) + \mathbf{b}_2, \\ \mathbf{H}_{ITS}^s &= \text{LayerNorm}(\mathbf{O} + \text{Dropout}(\mathbf{O}')) \end{aligned} \tag{17}$$

where $\mathbf{H}_{ITS}^x \in \mathbb{R}^{M_x\times 3d}$ is the final outputted representations of the neighbors of $v_x \in \{v_s, v_t\}$, $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_{\text{FFN}}^1, \mathbf{W}_{\text{FFN}}^2 \in \mathbb{R}^{3d\times 3d}$ and $\mathbf{b}_{\text{FFN}}^1, \mathbf{b}_{\text{FFN}}^2 \in \mathbb{R}^{3d}$ are the trainable parameters, and GELU is the activation function.

3.4.2 Prediction Generation. Finally, for node $v_x \in \{v_s, v_t\}$, we can obtain the final representation combining both the individual structural and temporal representation as well as the CS representation by concatenating $\mathbf{h}^x_{\text{ITS}}$ and $\mathbf{h}^x_{\text{CS}}$ together as follows:

$$\mathbf{f}_x = \mathbf{W}_{\text{fuse}}[\mathbf{h}^x_{\text{ITS}}, \mathbf{h}^x_{\text{SC}}], \tag{18}$$

where $\mathbf{f}_x$ is the final representation of $v_x \in \{v_s, v_t\}$, and $\mathbf{W}_{\text{fuse}} \in \mathbb{R}^{d \times 4d}$ is the trainable parameters.

After generating the dynamic representations of source node v_s and target node v_t as $\mathbf{f}_s$ and $\mathbf{f}_t$, respectively, as shown in Figure 1, we concatenate them together, which are then inputted into a two-layer MLP to generate the temporal link prediction score, i.e., the probability of forming a future link between v_s and v_t. This can be formulated as follows:

$$y_{st} = \text{MLP}([\mathbf{f}_s, \mathbf{f}_t]) = \mathbf{W}^2_{\text{Pred}}(\text{ReLU}(\mathbf{W}^1_{\text{Pred}}[\mathbf{f}_s, \mathbf{f}_t] + \mathbf{b}^1_{\text{Pred}})) + \mathbf{b}^2_{\text{Pred}}, \tag{19}$$

where $\mathbf{W}^1_{\text{Pred}} \in \mathbb{R}^{d \times 2d}$ and $\mathbf{W}^2_{\text{Pred}} \in \mathbb{R}^{1 \times d}$ are the learnable parameters, $\mathbf{b}^1_{\text{Pred}}, \mathbf{b}^2_{\text{Pred}} \in \mathbb{R}^d$ are the biased vectors, and ReLU is the activation function.

3.4.3 Model Optimization. Finally, to train the proposed LightDyG model, we utilize the cross-entropy function as the optimization objective, which enables the learning of the contained trainable parameters in LightDyG as follows:

$$\mathcal{L} = \sum_{(v_s, v_t, t_{st}) \in \mathcal{E}_{tra}} -\log(\sigma(y_{st})) - \log(\sigma(1 - y_{sn})), \tag{20}$$

where $(v_s, v_t, t_{st}) \in \mathcal{E}_{tra}$ is the observed temporal edge in the training set, and σ denotes the sigmoid function. Moreover, v_n is the randomly sampled negative sample, and z_{sn} is the corresponding prediction score of the negative edge (v_s, v_n, t_{st}) connecting v_s and v_n at the timestamp t_{st}. Finally, the back-propagation algorithm is applied to optimize the trainable parameters in LightDyG.

The training process of LightDyG is shown in Algorithm 2. Specifically, given the temporal networks until the current timestamp as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, for each temporal edge v_s, v_t, t_{st}, we first calculate the incremental neighbor occurrences of source node v_s and target node v_t in line 1. Then, the dynamic representation learning of v_s and v_t is similar, here we take the source node v_s as an example to introduce the detailed procedures. Specifically, (1) on the one hand, for the structural encoding, the individual and correlated structure embeddings $\mathbf{s}^s_i$ and $\mathbf{c}^s_i$ are generated in lines 6 and 7, respectively; (2) on the other hand, for the temporal encoding, the time-invariant interaction and appearance time embeddings $\bar{\mathbf{t}}^s_i, \hat{\mathbf{t}}^s_i$ are generated in line 8, respectively. Then, the generated embeddings $\mathbf{s}^s_i, \mathbf{c}^s_i, \bar{\mathbf{t}}^s_i, \hat{\mathbf{t}}^s_i$ are inputted into a dual-channel Transformer to output the final dynamic representation of source node as $\mathbf{f}_s$ in line 9. In addition, similar operations are conducted on the target node v_t to output its final representation $\mathbf{f}_t$ in lines 12–15. Next, $\mathbf{f}_s$ and $\mathbf{f}_t$ are combined to make predictions in line 17. Finally, the cross-entropy loss is adopted to compute the optimization loss in line 19, and back-propagation is utilized for training the learnable parameters line 20.

4 Experiments

4.1 Research Questions (RQs)

We validate the effectiveness and efficiency of our proposed LightDyG by answering the following six RQs:

RQ1: Can our proposed LightDyG achieve the state-of-the-art performance on four real-world continuous-time temporal networks?

RQ2: How is the time complexity of our proposed LightDyG? Can the designed efficient structural encoding mechanism in LightDyG save the computational time of structure encoding and accelerate the model convergence during training?

Algorithm 2: The Training Procedure of Our Proposed LightDyG

Input: $\mathcal{G}_{cur} = \{\mathcal{V}_{cur}, \mathcal{E}_{cur}\}$: The training temporal network;
Output: The learnable parameters of LightDyG;

for epoch in range(Epochs) **do**
  for b in range(Batches) **do**
    for (v_s, v_t, t_{st}) in $\mathcal{E}_{cur}^b \subseteq \mathcal{E}_{cur}$ **do**
      $h(v_s), h(v_t) \leftarrow$ use Eqs. (4)-(7) with $\mathcal{N}_{v_s}$, $\mathcal{N}_{v_t}$ as inputs;
      for v_i in $\mathcal{N}_{v_s}$ **do**
        $\mathbf{s}_i^s \leftarrow$ use Eq. (8) with $v_i, h(v_s)$ as inputs;
        $\mathbf{c}_i^s \leftarrow$ use Eq. (9) with $v_i, h(v_t)$ as inputs;
        $\bar{\mathbf{t}}_i^s, \hat{\mathbf{t}}_i^s \leftarrow$ use Eqs. (13), (14) and (11) with t_{st}, t_i, t_i^0 as inputs;
        $\mathbf{f}_s \leftarrow$ use Eqs. (15)-(18) with $\mathbf{s}_i^s, \mathbf{c}_i^s, \bar{\mathbf{t}}_i^s, \hat{\mathbf{t}}_i^s$ as inputs;
      end for
      for v_i in $\mathcal{N}_{v_t}$ **do**
        $\mathbf{s}_i^t \leftarrow$ use Eq. (8) with $v_i, h(v_t)$ as inputs;
        $\mathbf{c}_i^t \leftarrow$ use Eq. (9) with $v_i, h(v_s)$ as inputs;
        $\bar{\mathbf{t}}_i^t, \hat{\mathbf{t}}_i^t \leftarrow$ use Eqs. (13), (14) and (11) with t_{st}, t_i, t_i^0 as inputs;
        $\mathbf{f}_t \leftarrow$ use Eqs. (15)-(18) with $\mathbf{s}_i^t, \mathbf{c}_i^t, \bar{\mathbf{t}}_i^t, \hat{\mathbf{t}}_i^t$ as inputs;
      end for
      $y_{st} \leftarrow$ use Eq. (19) with $\mathbf{f}_s, \mathbf{f}_t$ as inputs;
    end for
    $L \leftarrow$ use Eq. (20) with y_{st}, y_{sn} as inputs;
    use back-propagation to optimize the parameters;
  end for
end for
return The learnable parameters of LightDyG.

RQ3: How is the contribution of different components in LightDyG to the prediction performance?
RQ4: How does LightDyG perform compared with the competitive baseline on the historically appearing repeat edges and newly emerging non-repeat edges?
RQ5: How is the performance of LightDyG comparing to the competitive baselines on the temporal multigraphs with abundant repeat edges?
RQ6: How sensitive is the performance of LightDyG to the main hyper-parameters, i.e., the sampled neighbor number M and the rescaling coefficient λ?

4.2 Datasets and Evaluation Metrics

To prove the effectiveness and efficiency of our proposed LightDyG model, we evaluate the performance and computational cost of LightDyG and the baselines on four real-world temporal networks, i.e., MathOverflow,[1] AskUbuntu,[2] SuperUser,[3] and StackOverflow.[4] Four temporal networks contain the interactions on different stack exchange Web sites, which each edge (v_s, v_t, t_{st}) indicates that

[1] https://snap.stanford.edu/data/sx-mathoverflow.html.
[2] https://snap.stanford.edu/data/sx-askubuntu.html.
[3] https://snap.stanford.edu/data/sx-superuser.html.
[4] https://snap.stanford.edu/data/sx-stackoverflow.html.

Table 2. Statistics of the Datasets without Abundant Repeat Edges Used in Our Experiments

Statistics	MathOverflow	AskUbuntu	SuperUser	StackOverflow
# Nodes	7,613	57,737	69,289	267,361
# Unique edges	35,222	145,485	168,156	765,228
# Edges	66,789	250,990	257,809	1,225,463
Duration	1 year	1 year	1 year	1 month
Density	2.30E-03	1.51E-04	1.07E-04	3.43E-05
Average weight	1.90	1.73	1.53	1.60

The density is calculated as $2|E|/|V|(|V|+1)$, where $|V|$ and $|E|$ indicate the number of nodes and edges in the whole dataset, respectively.

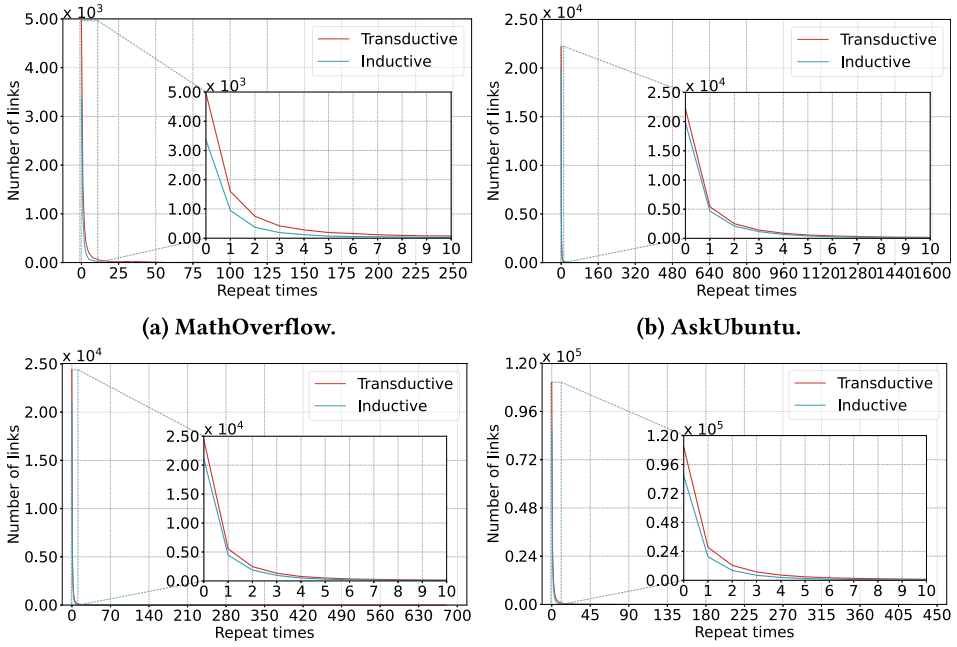
user v_s interacts with user v_t at the timestamp t_{st}. In addition, as for MathOverflow, AskUbuntu, and SuperUser, we adopt the data of the last year for experiments, where the data of the last 1 month for StackOverflow is selected for experiments considering its extremely large size.

Data Split. Moreover, following [48, 60], we separate the temporal edges in each dataset according to the proportions of 75%, 15%, and 15% for training, validation, and test, respectively. Furthermore, for the inductive temporal link prediction, we randomly mask 10% of the nodes in the validation and test sets as the nodes unseen during training. Then, the links associated with the masked nodes in the training set are removed, and those in the validation and test sets are deemed as the future inductive links to predict. We summarize the detailed statistics of four preprocessed datasets in Table 2. Different from the widely adopted datasets which can only represent special cases of temporal networks, i.e., temporal multigraphs [21, 48, 60], our adopted four temporal networks conform to the general real-world scenarios without the above characteristic, i.e., without abundant repeat edges in temporal multigraphs. In addition, we plot the distribution of links' repeat times on four temporal networks in Figure 2, where we can observe that most edges have small repeat times on various temporal networks.

Temporal Multigraph Data. In addition, to further examine the effectiveness of LightDyG on the temporal multigraphs with abundant repeat edges, we further introduce four temporal multigraphs commonly used in previous researches [48, 55, 60], i.e., Wikipedia, Reddit, MOOC, and UCI. The statistics of these four datasets are presented in Table 3. By comparing Tables 2 and 3, we can observe that the average edge weights of Wikipedia, Reddit, MOOC, and UCI are all larger than our mainly adopted four datasets, i.e., MathOverflow, AskUbuntu, SuperUser, and StackOverflow.

Transductive vs. Inductive. Furthermore, nodes in temporal networks with the first appearance timestamp before and after the current timestamp t_{cur} can be divided into nodes seen during training (i.e., the seen nodes $\mathcal{V}_S = \{v_x | t_x^0 \leq t_{cur}\}$) and nodes unseen during training (i.e., the unseen nodes $\mathcal{V}_U = \{v_x | t_x^0 > t_{cur}\}$), respectively. Correspondingly, the temporal link prediction includes the transductive link prediction predicting edges connecting two seen nodes (i.e., $\mathcal{E}_T = \{(v_s, v_t, t_{st}) | v_s, v_t \in \mathcal{V}_S, t_{st} > t_{cur}\}$) and inductive link prediction forecasting edges associated with unseen nodes (i.e., $\mathcal{E}_T = \{(v_s, v_t, v_{st}) | v_s \in \mathcal{V}_U || v_t \in \mathcal{V}_U, t_{st} > t_{cur}\}$). Here, we provide the prediction performance of LightDyG with the baselines on predicting both the transductive and inductive links for proving a comprehensive comparison.

Evaluation Metrics. In addition, following [48, 55, 60], we adopt the **average precision (AP)** and the **area under the ROC curve (AUC)** as the evaluation metrics for examining the prediction performance of our proposed LightDyG and the baselines.



(a) MathOverflow. **(b) AskUbuntu.**

(c) SuperUser. **(d) StackOverflow.**

Fig. 2. Distribution of links' repeat times on four temporal networks.

Table 3. Statistics of Four Temporal Multigraph Datasets with Abundant Repeat Edges

Statistics	Wikipedia	Reddit	MOOC	UCI
# Nodes	9,227	10,984	7,144	1,899
# Unique edges	18,257	78,516	178,443	20,296
# Edges	157,474	672,447	411,749	59,835
Duration	1 month	1 month	17 months	196 days
Density	3.70E-03	1.11E-02	1.61E-02	3.32E-02
Average weight	8.63	8.56	2.31	2.95

4.3 Baselines for Comparison

The following eleven competitive baselines are selected for comparison to validate the effectiveness and efficiency of our proposed LightDyG, which can be mainly divided to three groups: (1) Three traditional heuristic methods, i.e., *preferential attachment (PA)* [29], *common neighbor (CN)* [25], and *Adamic Adar (AA)* [1]; (2) Six feature-aware methods, i.e., *JODIE* [21], *DyRep* [40], *TGN* [33], *TGAT* [55], *TCL* [47], and *GraphMixer* [5]; (3) Two structure-enhanced methods, i.e., *CAWN* [48] and *DyGFormer* [60]. The detailed introduction of the baselines are presented as follows:

Three Traditional Heuristic Methods:

— *PA* [29], which follows the mechanism that links tend to be generated among nodes of large degree and calculates the link formation probability as $d(v_s) * d(v_t)$, where d measures the node degree.

— *CN* [25], which measures the correlations between source and target nodes by the number of their common neighbors, i.e., $|\mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}|$.
— *AA* [1], which further weights the contribution of each common neighbor by the node degree on the basis of CN and can be calculated as $\sum_{v_i \in \mathcal{N}_{v_s} \cap \mathcal{N}_{v_t}} 1/\log(d(v_i))$.

Six Feature-Aware Methods:

— *JODIE* [21] designs a coupled recurrent neural network to learn dynamic embeddings of nodes in temporal interaction networks and further predicts their future embedding trajectory for making interaction predictions.
— *DyRep* [40] proposes a two-time scale deep temporal point process model to capture the observed network evolution, followed by a temporal-attentive representation network to encode temporally evolving structural information.
— *TGN* [33] proposes to efficiently learn from the dynamic graphs represented as sequences of timed events through a combination of memory modules [36] and graph-based operators.
— *TGAT* [55] proposes to aggregate the temporal-topological neighborhood features using the SAT and learns the time features through the continuous-time encoding [54] based on the classical Bochner's theorem [32].
— *TCL* [47] proposes a two-stream encoder for processing source and target nodes using a designed graph-topology-aware Transformer, which are then integrated by a co-attentive Transformer.
— *GraphMixer* [5] designs a simple architecture GraphMixer with fast convergence and well generalization ability, which learns from one-hop historical neighbors using the MLP-Mixer [39].

Two Structure-Enhanced Methods:

— *CAWN* [48] designs a CAW strategy to learn the temporal network motifs between source and target nodes for capturing the network dynamics.
— *DyGFormer* [60] proposes to learn the SCs between source and target nodes by a NCM and processes the long-term temporal dependencies using a patching technique in Transformer.

4.4 Model Implementation

We implement various methods including LightDyG and the baselines based on the open-source toolkit[5] proposed in [60]. Moreover, we tune the hyper-parameters in LightDyG and the baselines on the validation set, where the total training epoch is set to 100 and an early stopping strategy with a patience of 20 is adopted. Then, the hyper-parameters achieving the best validation performance are applied on the test set for evaluation of each model. Specifically, we utilize the Adam optimizer to train the learnable parameters and set the learning rate to $1e^{-4}$. Moreover, we adopt the batch size of 200, and the dimensions of both structure encoding and time encoding are both set to 100. In addition, we tune the sampled neighbor number M of all models in $\{2, 4, \ldots, 128\}$ and search the rescaling coefficient λ of LightDyG in $\{1e^1, 1e^2, \ldots, 1e^8\}$. The impact of the sampled neighbor number M and the rescaling coefficient λ on LightDyG is further analyzed in Section 5.6.

5 Results and Discussion

5.1 Overall Performance

The results of LightDyG and the baselines in terms of AP and AUC on four real-world temporal networks are presented in Table 4, from which we can obtain the following observations:

[5] https://github.com/yule-BUAA/DyGLib.

Table 4. Temporal Link Prediction Performance of LightDyG and the Baselines in Terms of AP and AUC on Temporal Networks without Abundant Repeat Edges

Task	Method	MathOverflow		AskUbuntu		SuperUser		StackOverflow	
		AP	**AUC**	**AP**	**AUC**	**AP**	**AUC**	**AP**	**AUC**
Transductive	PA	$53.54_{\pm 0.07}$	$56.18_{\pm 0.11}$	$55.21_{\pm 0.07}$	$58.11_{\pm 0.10}$	$54.56_{\pm 0.04}$	$57.21_{\pm 0.04}$	$53.88_{\pm 0.02}$	$56.45_{\pm 0.03}$
	CN	$78.85_{\pm 0.14}$	$80.23_{\pm 0.13}$	$74.90_{\pm 0.09}$	$75.94_{\pm 0.08}$	$73.73_{\pm 0.07}$	$74.80_{\pm 0.05}$	$79.01_{\pm 0.01}$	$79.03_{\pm 0.00}$
	AA	$81.54_{\pm 0.09}$	$81.18_{\pm 0.11}$	$77.54_{\pm 0.09}$	$76.75_{\pm 0.07}$	$76.10_{\pm 0.07}$	$75.45_{\pm 0.05}$	$79.09_{\pm 0.01}$	$79.00_{\pm 0.00}$
	JODIE	$78.22_{\pm 0.14}$	$75.56_{\pm 0.26}$	$82.47_{\pm 0.14}$	$79.72_{\pm 0.14}$	$80.77_{\pm 0.12}$	$77.85_{\pm 0.05}$	$89.79_{\pm 0.23}$	$87.04_{\pm 0.29}$
	DyRep	$69.34_{\pm 0.74}$	$68.32_{\pm 0.69}$	$73.63_{\pm 0.81}$	$70.66_{\pm 0.43}$	$71.46_{\pm 0.63}$	$69.12_{\pm 0.51}$	$78.59_{\pm 0.83}$	$74.98_{\pm 0.70}$
	TGAT	$81.03_{\pm 1.49}$	$78.08_{\pm 1.29}$	$84.66_{\pm 0.40}$	$80.65_{\pm 0.34}$	$82.82_{\pm 0.40}$	$78.69_{\pm 0.34}$	$86.08_{\pm 0.44}$	$82.60_{\pm 0.45}$
	TGN	$81.24_{\pm 0.84}$	$79.12_{\pm 0.68}$	$85.34_{\pm 0.30}$	$82.83_{\pm 0.24}$	$82.20_{\pm 0.78}$	$78.84_{\pm 1.46}$	$89.84_{\pm 0.38}$	$87.58_{\pm 0.45}$
	TCL	$84.00_{\pm 0.44}$	$80.28_{\pm 0.40}$	$\underline{88.68_{\pm 0.11}}$	$\underline{84.67_{\pm 0.14}}$	$86.33_{\pm 0.06}$	$81.74_{\pm 0.08}$	$90.47_{\pm 0.06}$	$87.03_{\pm 0.12}$
	GraphMixer	$82.05_{\pm 1.02}$	$79.03_{\pm 0.62}$	$85.25_{\pm 0.64}$	$81.18_{\pm 0.65}$	$83.26_{\pm 0.64}$	$78.71_{\pm 0.72}$	$86.43_{\pm 0.48}$	$82.83_{\pm 0.29}$
	CAWN	$88.13_{\pm 0.22}$	$84.34_{\pm 0.30}$	$88.38_{\pm 0.08}$	$84.43_{\pm 0.12}$	$86.46_{\pm 0.25}$	$82.03_{\pm 0.36}$	$90.65_{\pm 0.08}$	$87.33_{\pm 0.14}$
	DyGFormer	$\underline{89.76_{\pm 0.05}}$	$\underline{86.37_{\pm 0.08}}$	$88.38_{\pm 0.15}$	$84.40_{\pm 0.20}$	$\underline{86.99_{\pm 0.29}}$	$\underline{82.95_{\pm 0.36}}$	$\underline{91.58_{\pm 0.09}}$	$\underline{88.51_{\pm 0.10}}$
	LightDyG	$\mathbf{96.18}_{\pm 0.03}^{\blacktriangle}$	$\mathbf{95.25}_{\pm 0.07}^{\blacktriangle}$	$\mathbf{96.03}_{\pm 0.32}^{\blacktriangle}$	$\mathbf{94.73}_{\pm 0.43}^{\blacktriangle}$	$\mathbf{95.08}_{\pm 0.13}^{\blacktriangle}$	$\mathbf{93.67}_{\pm 0.18}^{\blacktriangle}$	$\mathbf{95.41}_{\pm 0.36}^{\blacktriangle}$	$\mathbf{93.36}_{\pm 0.66}^{\blacktriangle}$
	Improv.	7.15%	10.28%	8.29%	11.88%	9.30%	12.92%	4.18%	5.48%
Inductive	PA	$53.39_{\pm 0.09}$	$55.71_{\pm 0.14}$	$55.13_{\pm 0.13}$	$57.70_{\pm 0.17}$	$54.34_{\pm 0.10}$	$56.56_{\pm 0.11}$	$53.64_{\pm 0.04}$	$55.78_{\pm 0.05}$
	CN	$70.75_{\pm 0.22}$	$74.25_{\pm 0.19}$	$71.97_{\pm 0.05}$	$73.82_{\pm 0.03}$	$70.37_{\pm 0.06}$	$72.22_{\pm 0.04}$	$75.56_{\pm 0.01}$	$75.62_{\pm 0.01}$
	AA	$73.66_{\pm 0.24}$	$75.27_{\pm 0.15}$	$74.74_{\pm 0.07}$	$74.61_{\pm 0.03}$	$72.89_{\pm 0.09}$	$72.86_{\pm 0.04}$	$75.63_{\pm 0.02}$	$75.57_{\pm 0.01}$
	JODIE	$70.07_{\pm 0.11}$	$68.34_{\pm 0.24}$	$81.28_{\pm 0.13}$	$79.11_{\pm 0.12}$	$79.25_{\pm 0.05}$	$76.91_{\pm 0.07}$	$87.80_{\pm 0.22}$	$85.02_{\pm 0.27}$
	DyRep	$60.74_{\pm 0.09}$	$61.06_{\pm 0.46}$	$70.66_{\pm 0.68}$	$68.84_{\pm 0.19}$	$68.77_{\pm 0.35}$	$67.49_{\pm 0.34}$	$74.30_{\pm 0.87}$	$70.86_{\pm 0.68}$
	TGAT	$76.11_{\pm 1.63}$	$72.35_{\pm 1.55}$	$83.16_{\pm 0.51}$	$79.01_{\pm 0.46}$	$81.45_{\pm 0.41}$	$77.27_{\pm 0.35}$	$84.43_{\pm 0.46}$	$80.74_{\pm 0.50}$
	TGN	$78.02_{\pm 1.66}$	$75.77_{\pm 1.33}$	$84.93_{\pm 0.23}$	$82.50_{\pm 0.22}$	$81.63_{\pm 1.18}$	$78.44_{\pm 1.79}$	$88.38_{\pm 0.27}$	$86.12_{\pm 0.37}$
	TCL	$79.00_{\pm 0.47}$	$74.31_{\pm 0.50}$	$\underline{87.65_{\pm 0.14}}$	$\underline{83.57_{\pm 0.17}}$	$85.30_{\pm 0.11}$	$80.62_{\pm 0.11}$	$89.37_{\pm 0.08}$	$85.74_{\pm 0.15}$
	GraphMixer	$77.02_{\pm 1.05}$	$73.01_{\pm 0.80}$	$83.84_{\pm 0.65}$	$79.69_{\pm 0.65}$	$82.00_{\pm 0.63}$	$77.37_{\pm 0.73}$	$84.61_{\pm 0.51}$	$80.84_{\pm 0.35}$
	CAWN	$84.49_{\pm 0.30}$	$79.67_{\pm 0.40}$	$87.51_{\pm 0.08}$	$83.48_{\pm 0.12}$	$85.48_{\pm 0.25}$	$80.94_{\pm 0.41}$	$89.89_{\pm 0.08}$	$86.38_{\pm 0.16}$
	DyGFormer	$\underline{86.10_{\pm 0.08}}$	$\underline{81.83_{\pm 0.15}}$	$87.30_{\pm 0.15}$	$83.20_{\pm 0.17}$	$\underline{85.68_{\pm 0.34}}$	$\underline{81.58_{\pm 0.39}}$	$\underline{90.48_{\pm 0.11}}$	$\underline{87.15_{\pm 0.13}}$
	LightDyG	$\mathbf{95.79}_{\pm 0.05}^{\blacktriangle}$	$\mathbf{94.83}_{\pm 0.09}^{\blacktriangle}$	$\mathbf{96.17}_{\pm 0.35}^{\blacktriangle}$	$\mathbf{94.95}_{\pm 0.46}^{\blacktriangle}$	$\mathbf{95.32}_{\pm 0.14}^{\blacktriangle}$	$\mathbf{94.00}_{\pm 0.18}^{\blacktriangle}$	$\mathbf{95.55}_{\pm 0.41}^{\blacktriangle}$	$\mathbf{93.59}_{\pm 0.74}^{\blacktriangle}$
	Improv.	11.25%	15.89%	9.72%	13.62%	11.25%	15.22%	5.60%	7.39%

The best baseline and the best performer in each column are underlined and bold, respectively. $^{\blacktriangle}$ indicates the improvement of LightDyG over the best baseline is significant with p-value < 0.01.

— *The performance of traditional heuristic baselines is unsatisfying.* First, among the traditional heuristic baselines PA, CN, and AA, we can observe that the pure structure-aware method PA performs worst, followed by the common neighbor mechanism CN which has much better performance than PA. Moreover, the AA method which integrates the node degree information on the basis of CN further improves the performance of CN. However, the heuristic methods merely consider the characteristics reflected on the temporal graph topological structure using different defined rules while neglecting the time dynamics, thus resulting in generally worse performance than the neural methods.

— *SCs are important for future link predictions.* Moreover, for the neural baselines, we can first see that the structure-enhanced methods, i.e., CAWN and DyGFormer, generally outperform the feature-aware methods. This indicates the importance of considering the SCs between source and target nodes in temporal link prediction. In addition, DyGFormer achieves the best performance among the baselines in terms of both AP and AUC in most cases except that TCL performs best in terms of both metrics on AskUbuntu. We analyze the superior performance

Table 5. Time Complexity of LightDyG, w/o IncreSE, and DyGFormer

Methods	CPU	GPU
LightDyG	$O(\|\mathcal{V}\|L)$	$O(\|\mathcal{V}\|L(M+M')d)$
w/o IncreSE	$O(\|\mathcal{V}\|LM)$	$O(\|\mathcal{V}\|L(M+M')d)$
DyGFormer	$O(\|\mathcal{V}\|LM)$	$O(\|\mathcal{V}\|L(N+N(N+Md)))$

of them may be due to that DyGFormer considers the SCs between nodes using the NCM while TCL models the feature correlations between nodes by a co-attentive Transformer. Moreover, the SCs play a pivotal role in predicting future links on temporal networks, which explains the generally better performance of DyGFormer than TCL.

— *LightDyG largely improves the dynamic link prediction performance, where the improvements are especially obvious in the inductive setting.* In addition, as for our proposed LightDyG, we can see that LightDyG achieves the best performance in terms of both AP and AUC in all cases on four temporal networks. Specifically, LightDyG outperforms the best baselines by 4.18% to 9.30% in terms of AP and 5.48% to 12.92% in terms of AUC on the transductive link prediction task, respectively, while the ranges of the corresponding improvement rates are 5.60% to 11.25% and 7.39% to 15.89% on the inductive scenario. It is obvious that LightDyG achieves larger improvements on the more challenging inductive setting predicting future links associated with new nodes than the transductive setting which forecasts links connecting two seen nodes. This indicates the practicability of LightDyG in real-world applications where new nodes continuously integrate into the temporal networks.

5.2 Computational Efficiency

To examine the computational efficiency of LightDyG, we first theoretically analyze the time complexity of LightDyG and its variant w/o IncreSE (short for *without incremental structure encoding*) as well as the best performing baseline DyGFormer. The results are presented in Table 5, where the time complexity on CPU comes from calculating the neighbor occurrences, which cannot be accelerated by GPU. Moreover, the time complexity on GPU comes from the matrix operations including embedding the structural and temporal features, the dual-channel Transformer, and so on.

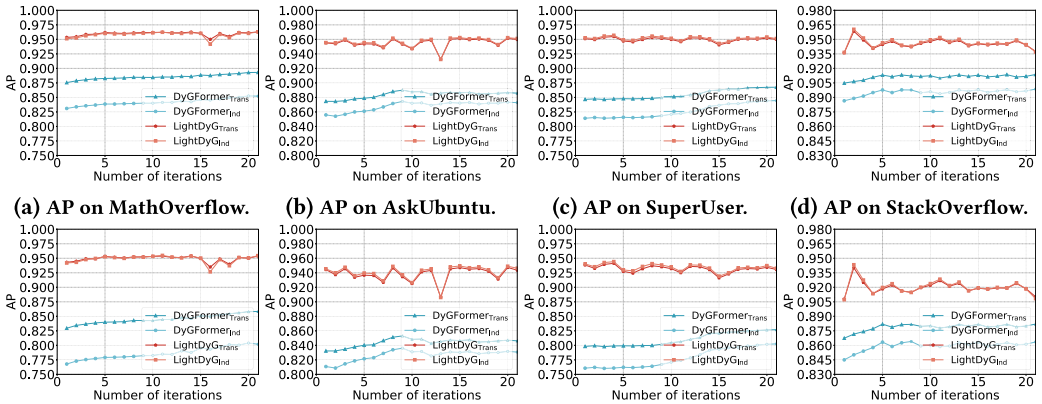
Moreover, we compare the training and test time during each iteration of LightDyG with w/o IncreSE and DyGFormer. The results measured using PyTorch 1.8.1 on a Ubuntu 18.04 server with NVIDIA GeForce RTX 3090 GPU with 24 GB memories are presented in Table 6. In addition, we also investigate the training process of LightDyG by comparing with DyGFormer. The performance curves of LightDyG and DyGFormer in terms of AP on the validation set are plotted in Figure 3.

From the theoretical and empirical results, we can obtain the following observations:

— *LightDyG largely decreases the time complexity of dynamic link prediction on CPU.* In Table 5, L denotes the full neighbor sequence length, M and M' denote the number of the sampled individual neighbors of source node/target node and the number of their common neighbors, respectively. From Table 5, we can see that the complexity of w/o IncreSE on CPU and GPU are equivalent to LightDyG and DyGFormer, respectively. In addition, comparing the complexity of LightDyG and DyGFormer on GPU, the dual-channel Transformer in LightDyG introduces a complexity term $M'd$ in a single Transformer layer, while DyGFormer requires multi-layer Transformers. Thus, the complexity of them on GPU cannot be obviously distinguished. Differently, for the complexity on CPU, LightDyG largely decreases the time complexity on the basis of DyGFormer. Moreover, considering that the computation on CPU cannot

Table 6. Training and Test Time in Seconds during Each Iteration of LightDyG and Its Variant w/o IncreSE as Well as DyGFormer, Where the Training Time Is Further Separated to the Forward Process and the Backward Optimization Stage, and the Test Time Includes Prediction in the Transductive and Inductive Settings

Dataset	Method	Training			Test		
		Both	Forward	Backward	Both	Transductive	Inductive
MathOverflow	LightDyG	13.5503	9.0894	4.4609	3.7103	2.4970	1.2133
	w/o IncreSE	25.0495	20.3723	4.6772	8.3567	5.5809	2.7758
	DyGFormer	29.1450	23.0743	6.0707	9.5359	6.3697	3.1662
AskUbuntu	LightDyG	65.5686	49.5949	15.9737	24.8094	14.6633	10.1471
	w/o IncreSE	87.3911	70.9567	16.4344	35.8399	20.2208	15.6191
	DyGFormer	100.1510	80.5150	19.6360	40.7152	23.1014	17.6138
SuperUser	LightDyG	58.8880	41.2861	17.6019	13.0941	3.6018	9.4923
	w/o IncreSE	96.1152	78.3132	17.8020	36.3108	20.9437	15.3671
	DyGFormer	109.9277	87.7261	22.2016	41.2605	23.7349	17.5256
StackOverflow	LightDyG	244.2792	163.3556	80.9236	75.7900	46.3044	29.4856
	w/o IncreSE	435.8461	352.6108	83.2353	161.6043	98.5081	63.0962
	DyGFormer	509.6913	405.4495	104.2418	184.9016	113.1654	71.7362



(a) AP on MathOverflow. (b) AP on AskUbuntu. (c) AP on SuperUser. (d) AP on StackOverflow.

(e) AUC on MathOverflow. (f) AUC on AskUbuntu. (g) AUC on SuperUser. (h) AUC on StackOverflow.

Fig. 3. Performance curves of LightDyG and DyGFormer on the validation set in training.

be accelerated using the deep learning frameworks like PyTorch, the contribution of the complexity on CPU decreased by LightDyG to the computational time cost is obvious.

— *Our designed incremental structure encoding strategy obviously decreases the computational time of temporal link prediction.* From Table 6, we can first observe that DyGFormer requires slightly more time in all cases than w/o IncreSE due to its introduced patching mechanism for dealing with long sequences. Moreover, comparing LightDyG and w/o IncreSE, we can see that the reduction rates of the running time cost of training and test are 45.91% and 55.60% on MathOverflow, respectively, while the reduction rates are 24.97% and 30.78% on AskUbuntu, 38.73% and 63.94% on SuperUser, and 43.95% and 53.10% on StackOverflow, respectively. We can obviously find that the reduction rates are all higher in the test stage than those during training. This is because that the optimization process during training cannot be accelerated,

Table 7. Performance of LightDyG and Its Variants for Examining the Utility of Different Contained Components in LightDyG

Task	Method	MathOverflow		AskUbuntu		SuperUser		StackOverflow	
		AP	AUC	AP	AUC	AP	AUC	AP	AUC
Transductive	**LightDyG**	**96.18**$_{\pm 0.03}$	**95.25**$_{\pm 0.07}$	**96.03**$_{\pm 0.32}$	**94.73**$_{\pm 0.43}$	**95.08**$_{\pm 0.13}$	**93.67**$_{\pm 0.18}$	**95.41**$_{\pm 0.36}$	**93.36**$_{\pm 0.66}$
	w/o StructuralEncoding	85.83$_{\pm 0.27}$	83.94$_{\pm 0.15}$	91.58$_{\pm 0.75}$	89.03$_{\pm 0.88}$	89.84$_{\pm 0.72}$	87.14$_{\pm 0.51}$	91.29$_{\pm 0.15}$	88.62$_{\pm 0.24}$
	w/o TemporalEncoding	88.21$_{\pm 0.05}$	84.01$_{\pm 0.09}$	86.32$_{\pm 0.34}$	81.52$_{\pm 0.46}$	83.77$_{\pm 0.06}$	78.43$_{\pm 0.12}$	86.95$_{\pm 0.07}$	81.85$_{\pm 0.16}$
	w/o InvariantTime	93.07$_{\pm 0.16}$	91.21$_{\pm 0.23}$	93.92$_{\pm 0.10}$	91.76$_{\pm 0.15}$	92.88$_{\pm 0.14}$	90.50$_{\pm 0.22}$	94.50$_{\pm 0.11}$	92.22$_{\pm 0.18}$
	w/o InteractionTime	91.58$_{\pm 0.06}$	89.13$_{\pm 0.13}$	92.91$_{\pm 0.21}$	90.44$_{\pm 0.35}$	91.76$_{\pm 0.18}$	89.14$_{\pm 0.34}$	93.22$_{\pm 0.08}$	90.90$_{\pm 0.21}$
	w/o AppearanceTime	89.94$_{\pm 0.17}$	86.51$_{\pm 0.20}$	88.91$_{\pm 0.31}$	85.06$_{\pm 0.40}$	86.25$_{\pm 0.30}$	81.78$_{\pm 0.35}$	91.31$_{\pm 0.07}$	88.19$_{\pm 0.13}$
Inductive	**LightDyG**	**95.79**$_{\pm 0.05}$	**94.83**$_{\pm 0.09}$	**96.17**$_{\pm 0.35}$	**94.95**$_{\pm 0.46}$	**95.32**$_{\pm 0.14}$	**94.00**$_{\pm 0.18}$	**95.55**$_{\pm 0.41}$	**93.59**$_{\pm 0.74}$
	w/o StructuralEncoding	83.80$_{\pm 0.36}$	81.63$_{\pm 0.23}$	91.32$_{\pm 0.84}$	88.74$_{\pm 1.01}$	89.82$_{\pm 0.66}$	87.13$_{\pm 0.55}$	90.89$_{\pm 0.12}$	87.98$_{\pm 0.27}$
	w/o TemporalEncoding	84.08$_{\pm 0.09}$	78.79$_{\pm 0.15}$	85.19$_{\pm 0.28}$	80.20$_{\pm 0.44}$	82.58$_{\pm 0.10}$	76.98$_{\pm 0.16}$	85.07$_{\pm 0.09}$	79.49$_{\pm 0.20}$
	w/o InvariantTime	92.02$_{\pm 0.24}$	89.98$_{\pm 0.30}$	93.80$_{\pm 0.11}$	91.63$_{\pm 0.17}$	92.85$_{\pm 0.16}$	90.50$_{\pm 0.24}$	94.32$_{\pm 0.13}$	91.90$_{\pm 0.21}$
	w/o InteractionTime	90.21$_{\pm 0.11}$	87.45$_{\pm 0.26}$	92.64$_{\pm 0.24}$	90.15$_{\pm 0.39}$	91.75$_{\pm 0.20}$	89.21$_{\pm 0.37}$	93.15$_{\pm 0.10}$	90.86$_{\pm 0.25}$
	w/o AppearanceTime	86.45$_{\pm 0.23}$	82.15$_{\pm 0.26}$	88.11$_{\pm 0.31}$	84.13$_{\pm 0.41}$	85.23$_{\pm 0.44}$	80.61$_{\pm 0.56}$	90.28$_{\pm 0.12}$	86.92$_{\pm 0.18}$

The best performer in each column is bold.

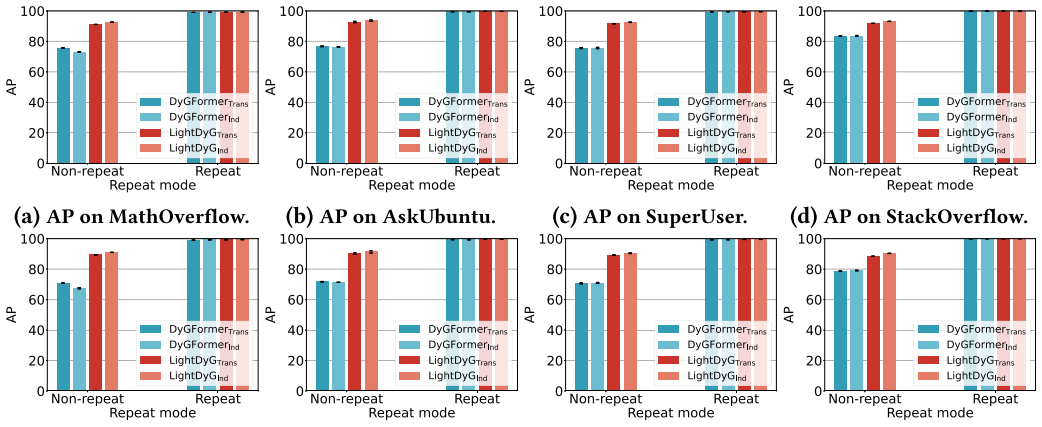
where as shown in Table 6 that LightDyG and w/o IncreSE consume similar optimization time in all cases.

— *The incremental structure encoding strategy effectively promotes the convergence of LightDyG during the model optimization.* From Figure 3, we can clearly observe that DyGFormer requires a large number of iterations to achieve the best performance, especially on the MathOverflow and SuperUser datasets. Differently, our proposed LightDyG generally achieves or approaches the best performance in the first several iterations, i.e., no more than 5. This is due to that the explicit modeling of the SCs between nodes contributes to the fast convergence of LightDyG, which can quickly learns the network evolution from the training set for predicting the future links in the validation set.

5.3 Ablation Study

To investigate the effectiveness of different components in LightDyG, we compare LightDyG with its five variants including: (1) w/o StructuralEncoding, which removes the efficient structural encoding component and generates the node embeddings by modeling the temporal features of neighbors on temporal networks, i.e., the interaction time embedding $\bar{\mathbf{t}}_i$ and the appearance time embedding $\hat{\mathbf{t}}_i$ for each neighbor in Equation (15); (2) w/o TemporalEncoding, which completely removes the effective temporal encoding module, including the time-invariant strategy and the encoding of the neighbor interaction timestamps as well as the neighbor appearance timestamps; (3) w/o InvariantTime, which removes the time-invariant strategy, i.e., Equations (13) and (14), in the effective temporal encoding module; (4) w/o InteractionTime, which removes the encoding of the neighbor interaction timestamp, i.e., Equation (10), on the basis of w/o InvariantTime; (5) w/o AppearanceTime, which removes the encoding of the neighbor appearance timestamp, i.e., Equation (12), on the basis of w/o InvariantTime. In addition, it is worth noting that the time encoding in the variant w/o AppearanceTime, which merely considers the neighbor interaction timestamps, is equivalent to the universal functional time encoding strategy widely adopted in existing researches [55, 58, 60]. We present the results of LightDyG and its variants in terms of AP and AUC on four temporal networks in Table 7, from which we can observe the following observations:

— *Time dynamics are more important than structural features on temporal networks.* From Table 7, by comparing w/o StructuralEncoding and w/o TemporalEncoding with LightDyG, we can



(a) AP on MathOverflow. (b) AP on AskUbuntu. (c) AP on SuperUser. (d) AP on StackOverflow.

(e) AUC on MathOverflow. (f) AUC on AskUbuntu. (g) AUC on SuperUser. (h) AUC on StackOverflow.

Fig. 4. Performance of LightDyG and DyGFormer on non-repeat edges and repeat edges.

find that both the structural encoding and time encoding play important roles in generating accurate predictions. Moreover, w/o TemporalEncoding generally performs worse than w/o StructuralEncoding in most cases and underperforms LightDyG by a large margin in all cases, indicating the necessity of considering the time dynamics in temporal networks.

— *The time scale bias of various nodes brought by the continuous network evolution leads to the performance decrease.* In addition, comparing w/o InvariantTime with LightDyG, the utility of the time-invariant temporal encoding strategy can be obviously validated. This indicates that the bias caused by the continuous network evolution leads to much performance decrease and our designed time-invariant strategy can well eliminate the bias.

— *It is critical to consider the historical temporal characteristics of nodes.* Furthermore, comparing the variants w/o InteractionTime and w/o AppearanceTime, we can observe that the appearance time encoding contributes much more to the performance of LightDyG than the interaction time encoding. This indicates the importance of considering the historical temporal characteristics of the sampled neighbors in temporal link prediction.

5.4 Impact of Edge Repetition

As shown in Figure 2, in our adopted four real-world temporal networks, most edges are of low repeat times. To examine the different performance of our proposed LightDyG on the historically appearing repeat edges and newly emerging non-repeat edges, we divide the edges in each temporal network into non-repeat edges and repeat edges. The performance of LightDyG and the best performing baseline DyGFormer on different edges in four temporal networks are presented in Figure 4, from which we can obtain the following observations:

— *Repeat edges on temporal networks are easy to accurately predict.* From Figure 4, we can first see that for the repeat edges, the performance of LightDyG and DyGFormer in both transductive and inductive settings are similar, where LightDyG achieves slight improvements above DyGFormer. Moreover, the performance of both two methods are close to 1 on four datasets, especially on StackOverflow. This indicates that predicting the repeat edges is generally easy to implement and has been well investigated by existing researches.

— *Non-repeat edges on temporal networks are challenging to predict, where LightDyG obviously improves the performance by the baseline.* In addition, we can first observe that the performance of LightDyG and DyGFormer in both transductive and inductive settings shows large gaps

Table 8. Temporal Link Prediction Performance of LightDyG and the Baselines in Terms of AP and AUC on Temporal Networks with Abundant Repeat Edges

Task	Method	Wikipedia		Reddit		MOOC		UCI	
		AP	AUC	AP	AUC	AP	AUC	AP	AUC
Transductive	TCL	96.47$_{\pm 0.16}$	95.84$_{\pm 0.18}$	97.53$_{\pm 0.02}$	97.42$_{\pm 0.02}$	82.38$_{\pm 0.24}$	83.12$_{\pm 0.18}$	89.57$_{\pm 1.63}$	87.82$_{\pm 1.36}$
	CAWN	98.76$_{\pm 0.03}$	98.54$_{\pm 0.04}$	99.11$_{\pm 0.01}$	99.01$_{\pm 0.01}$	80.15$_{\pm 0.25}$	80.38$_{\pm 0.26}$	95.18$_{\pm 0.06}$	93.87$_{\pm 0.08}$
	DyGFormer	**99.03**$_{\pm 0.02}$	98.91$_{\pm 0.02}$	**99.22**$_{\pm 0.01}$	**99.15**$_{\pm 0.01}$	87.52$_{\pm 0.49}$	87.91$_{\pm 0.58}$	95.79$_{\pm 0.17}$	94.49$_{\pm 0.26}$
	LightDyG	99.02$_{\pm 0.02}$	**98.92**$_{\pm 0.02}$	**99.22**$_{\pm 0.01}$	99.14$_{\pm 0.01}$	**87.96**$_{\pm 0.30}$	**88.39**$_{\pm 0.34}$	**96.27**$_{\pm 0.06}$	**95.31**$_{\pm 0.10}$
Inductive	TCL	96.22$_{\pm 0.17}$	95.57$_{\pm 0.20}$	94.09$_{\pm 0.07}$	93.80$_{\pm 0.07}$	80.60$_{\pm 0.22}$	81.43$_{\pm 0.19}$	87.36$_{\pm 2.03}$	84.49$_{\pm 1.82}$
	CAWN	98.24$_{\pm 0.03}$	98.03$_{\pm 0.04}$	98.62$_{\pm 0.01}$	98.42$_{\pm 0.02}$	81.42$_{\pm 0.24}$	81.86$_{\pm 0.25}$	92.73$_{\pm 0.06}$	90.40$_{\pm 0.11}$
	DyGFormer	98.59$_{\pm 0.03}$	98.48$_{\pm 0.03}$	**98.84**$_{\pm 0.02}$	**98.71**$_{\pm 0.01}$	86.96$_{\pm 0.43}$	87.62$_{\pm 0.51}$	94.54$_{\pm 0.12}$	92.63$_{\pm 0.13}$
	LightDyG	**98.81**$_{\pm 0.02}$	**98.68**$_{\pm 0.02}$	98.81$_{\pm 0.02}$	98.68$_{\pm 0.02}$	**87.80**$_{\pm 0.38}$	**88.46**$_{\pm 0.42}$	**95.28**$_{\pm 0.08}$	**93.83**$_{\pm 0.10}$

The best baseline and the best performer in each column are underlined and bold, respectively.

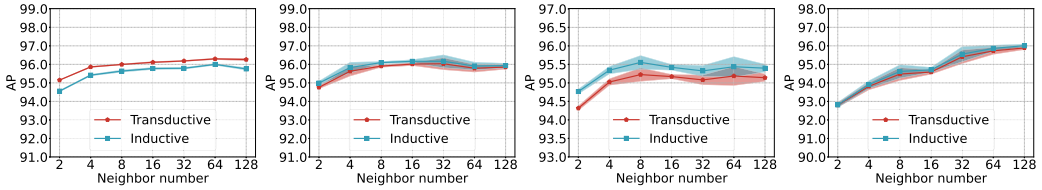
between non-repeat edges and repeat edges. This adequately validates our motivation that dynamic graph learning on general scenarios of temporal networks without abundant repeat edges is challenging and requires to be investigated. Furthermore, comparing the performance of LightDyG and DyGFormer on the non-repeat edges, we can observe that by adopting our designed time-invariant temporal encoding strategy with including both interaction timestamp and appearance timestamp, LightDyG shows obviously superior performance than DyGFormer.

5.5 Performance on Temporal Multigraphs

To further examine the effectiveness of LightDyG on the temporal multigraphs with abundant repeat edges, we follow previous researches [48, 55, 60] to adopt their four commonly used datasets, i.e., Wikipedia, Reddit, MOOC, and UCI for experiments. Specifically, we compare the performance of our proposed LightDyG and three baselines including TCL, CAWN, and DyGFormer which show competitive performance in Section 5.1. The results of LightDyG and three baselines TCL, CAWN, and DyGFormer in terms of AP and AUC on four temporal multigraphs are presented in Table 8. From Table 8, we can obtain the following observations:

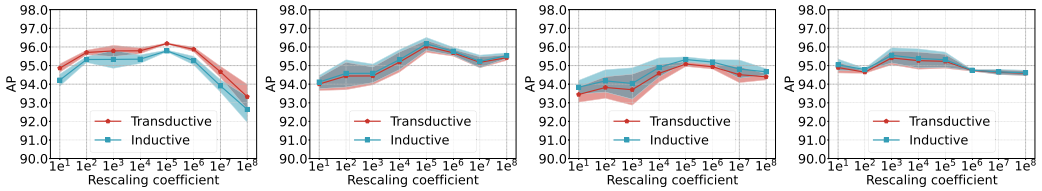
— *LightDyG achieves superior performance to the baselines on temporal multigraphs.* From Table 8, we can see that LightDyG achieves the best performance in terms of AP and AUC in most cases on four datasets. This indicates that LightDyG not only achieves the state-of-the-art performance on temporal networks without abundant repeat edges but also shows superior performance to the baselines on temporal multigraphs with abundant repeat edges.

— *LightDyG shows larger improvements on datasets with small average edge weights.* In addition, comparing the performance improvements of LightDyG above the baselines, we can see that on Wikipedia and Reddit, the performance of LightDyG and the best baseline DyGFormer are similar, and LightDyG slightly performs better than DyGFormer in some cases. However, differently on MOOC and UCI, LightDyG shows obviously better performance than DyGFormer. We analyze this phenomenon is due to that the average edge weights of four temporal multigraph datasets are different. Specifically, as shown in Table 3, the average weights of Wikipedia and Reddit are much larger than MOOC and UCI. This indicates that LightDyG can achieve larger performance improvements on datasets with small average edge weights.



(a) AP on MathOverflow. **(b) AP on AskUbuntu.** **(c) AP on SuperUser.** **(d) AP on StackOverflow.**

Fig. 5. Performance of LightDyG with different neighbor number M.



(a) AP on MathOverflow. **(b) AP on AskUbuntu.** **(c) AP on SuperUser.** **(d) AP on StackOverflow.**

Fig. 6. Performance of LightDyG with different rescaling coefficient λ.

5.6 Hyper-Parameter Analysis

5.6.1 Impact of Neighbor Number M. To analyze the impact of the sampled neighbor number M on the temporal link prediction performance, we range the sampled neighbor number in $\{2, 4, \ldots, 128\}$ and present the performance of LightDyG with various neighbor numbers in both transductive and inductive settings in Figure 5. From Figure 5, we can observe that on MathOverflow, AskUbuntu, and SuperUser, with the neighbor number increasing from 2 to 128, the performance of LightDyG and DyGFormer in both transductive and inductive scenarios consistently first increases until around the peak performance at $M = 32$ and then begins to decline. Differently on the StackOverflow dataset, the performance of both two models first increases and then remains stable after $M = 32$. In general, the best performance of LightDyG and DyGFormer is generally achieved around the neighbor number 32. We analyze this may be due to that small numbers of historical neighbors such as 2–8 cannot adequately model the structural and temporal information of nodes, while biased or outdated neighbors may be included in long historical sequences with large numbers of neighbors like 64 or 128.

5.6.2 Impact of Rescaling Coefficient λ. To analyze the impact of the rescaling coefficient λ on the temporal link prediction performance, we range the rescaling coefficient in $\{1e^1, 1e^2, \ldots, 1e^8\}$ and present the performance of LightDyG with various rescaling coefficients in both transductive and inductive settings in Figure 6. From Figure 6, we can observe that on when the rescaling coefficient increases from $1e^1$ to $1e^8$, the performance of LightDyG in both transductive and inductive scenarios generally first increases and then begins to decline. Moreover, the peak performance of LightDyG on MathOverflow, AskUbuntu, and SuperUser is generally achieved at the rescaling coefficient $\lambda = 1e^5$, while differently on StackOverflow, LightDyG achieves the best performance at $\lambda = 1e^3$. We analyze the possible reason is that the timespan of four temporal networks are different as shown in Table 2, where the timespan 1 month of StackOverflow is shorter than the other three datasets with the same timespan 1 year. Thus, a smaller rescaling coefficient is suitable for StackOverflow compared with the other three temporal networks.

6 Conclusions and Future Work

In this article, we focus on predicting future links in general temporal networks without specific attention on special cases of multigraphs with abundant repeat edges and propose an efficient and effective dynamic graph learning method named LightDyG. In detail, LightDyG decouples the SCs between source/target nodes and their individual substructures for fast convergence and designs an incremental structure encoding strategy for saving the computational cost of structure encoding. Moreover, LightDyGs achieves comprehensive modeling of temporal characteristics by considering both the interaction and appearance timestamps of each sampled neighbor and eliminates the temporal bias by a time-invariant temporal encoding strategy. Extensive experiments on four real-world datasets validate the effectiveness and efficiency of LightDyG.

As to future works, we would like to take the deletion of nodes and edges in temporal networks into consideration. In addition, we are also interested in utilizing the graph condensation strategy [8, 16] on temporal networks for both fast training and inference.

References

[1] Lada A. Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social Networks* 25, 3 (2003), 211–230.

[2] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. 2021. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM '21)*, 3747–3756.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the International Conference on Machine Learning (ICML '20)*, 1597–1607.

[4] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural ordinary differential equations. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '18)*, 6572–6583.

[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?. In *Proceedings of the International Conference on Learning Representations (ICLR '23)*.

[6] Claudio D. T. Barros, Matheus R. F. Mendonça, Alex B. Vieira, and Artur Ziviani. 2023. A survey on embedding dynamic graphs. *ACM Computing Surveys* 55, 1 (2023), 1–37.

[7] Yang Fang, Xiang Zhao, Peixin Huang, Weidong Xiao, and Maarten de Rijke. 2022. Scalable representation learning for dynamic heterogeneous information networks via metagraphs. *ACM Transactions of Information Systems* 40, 4 (2022), 1–27.

[8] Xinyi Gao, Tong Chen, Yilong Zang, Wentao Zhang, Quoc Viet, Hung Nguyen, Kai Zheng, and Hongzhi Yin. 2023. Graph condensation for inductive node representation learning. arXiv:2307.15967. Retrieved from https://arxiv.org/abs/2307.15967

[9] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. 2018. Unsupervised representation learning by predicting image rotations. In *Proceedings of the International Conference on Learning Representations (ICLR '18)*.

[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] Alessio Gravina, Giulio Lovisotto, Claudio Gallicchio, Davide Bacciu, and Claas Grohnfeldt. 2024. Long range propagation on continuous-time dynamic graphs. In *Proceedings of the International Conference on Machine Learning (ICML '24)*.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD '16)*, 855–864.

[13] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna R. Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational graph recurrent neural networks. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '19)*, 10700–10710.

[14] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '17)*, 1024–1034.

[15] Ming Jin, Yuan-Fang Li, and Shirui Pan. 2022. Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '22)*.

[16] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Graph condensation for graph neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR '22)*.

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

[18] Shima Khoshraftar and Aijun An. 2024. A survey on graph representation learning methods. *ACM Transactions on Intelligent Systems and Technology* 15, 1 (2024), 19:1–19:55.

[19] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR '17)*.

[20] Sanjay Kumar, Abhishek Mallik, Anavi Khetarpal, and Bhawani Sankar Panda. 2022. Influence maximization in social networks using graph embedding and graph neural network. *Information Sciences* 607 (2022), 1617–1636.

[21] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD '19)*, 1269–1278.

[22] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM '17)*, 387–396.

[23] Yunyu Liu, Jianzhu Ma, and Pan Li. 2022. Neural predicting higher-order patterns in temporal networks. In *Proceedings of the International World Wide Web Conference (WWW '22)*, 1340–1351.

[24] Zhijun Liu, Chao Huang, Yanwei Yu, and Junyu Dong. 2021. Motif-preserving dynamic attributed network embedding. In *Proceedings of the International World Wide Web Conference (WWW '21)*, 1629–1638.

[25] Francois Lorrain and Harrison C. White. 1971. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology* 1, 1 (1971), 49–80.

[26] 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 International Conference on Information and Knowledge Management (CIKM '19)*, 469–478.

[27] Yao Ma, Ziyi Guo, Zhaochun Ren, Jiliang Tang, and Dawei Yin. 2020. Streaming graph neural networks. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*, 719–728.

[28] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR '13)*.

[29] Mark E. J. Newman. 2001. Clustering and preferential attachment in growing networks. *Physical Review E* 64, 2 (2001), 025102.

[30] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of The AAAI Conference on Artificial Intelligence (AAAI '20)*, 5363–5370.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD '14)*, 701–710.

[32] Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '07)*, 1177–1184.

[33] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. arXiv:2006.10637. Retrieved from https://arxiv.org/abs/2006.10637

[34] 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 ACM International Conference on Web Search and Data Mining (WSDM '20)*, 519–527.

[35] Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node embedding over temporal graphs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '19)*, 4605–4612.

[36] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '15)*, 2440–2448.

[37] Sheng Tian, Ruofan Wu, Leilei Shi, Liang Zhu, and Tao Xiong. 2021. Self-supervised representation learning on dynamic graphs. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM '21)*, 1814–1823.

[38] Yuxing Tian, Yiyan Qi, and Fan Guo. 2024. FreeDyG: Frequency enhanced continuous-time dynamic graph model for link prediction. In *Proceedings of the International Conference on Learning Representations*.

[39] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. MLP-Mixer: An all-MLP architecture for vision. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '21)*, 24261–24272.

[40] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning representations over dynamic graphs. In *Proceedings of the International Conference on Learning Representations (ICLR '19)*.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '17)*, 5998–6008.

[42] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of the International Conference on Learning Representations (ICLR '18)*.

[43] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD '16)*, 1225–1234.

[44] Hongwei Wang and Jure Leskovec. 2022. Combining graph convolutional neural networks and label propagation. *ACM Transactions of Information Systems* 40, 4 (2022), 1–27.

[45] Jianling Wang, Kaize Ding, Liangjie Hong, Huan Liu, and James Caverlee. 2020. Next-item recommendation with sequential hypergraphs. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*, 1101–1110.

[46] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming graph neural networks via continual learning. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM '20)*, 1515–1524.

[47] Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. 2021. TCL: Transformer-based dynamic graph modelling via contrastive learning. arXiv:2105.07944. Retrieved from https://arxiv.org/abs/2105.07944

[48] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive representation learning in temporal networks via causal anonymous walks. In *Proceedings of the International Conference on Learning Representations (ICLR '21)*.

[49] Yanbang Wang, Pan Li, Chongyang Bai, and Jure Leskovec. 2021. TEDIC: Neural modeling of behavioral patterns in dynamic social interaction networks. In *Proceedings of the International World Wide Web Conference (WWW '21)*, 693–705.

[50] Chunyu Wei, Jian Liang, Bing Bai, and Di Liu. 2022. Dynamic hypergraph learning for collaborative filtering. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM '22)*, 2108–2117.

[51] Lanning Wei, Huan Zhao, Zhiqiang He, and Quanming Yao. 2024. Neural architecture search for GNN-based graph classification. *ACM Transactions of Information Systems* 42, 1 (2024), 1:1–1:29.

[52] Yuxia Wu, Yuan Fang, and Lizi Liao. 2024. On the feasibility of simple transformer for dynamic graph modeling. In *Proceedings of the International World Wide Web Conference (WWW '24)*, 870–880.

[53] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

[54] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. 2019. Self-attention with functional time representation learning. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '19)*, 15889–15899.

[55] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. In *Proceedings of the International Conference on Learning Representations (ICLR '20)*.

[56] Cheng Yang, Chunchen Wang, Yuanfu Lu, Xumeng Gong, Chuan Shi, Wei Wang, and Xu Zhang. 2022. Few-shot link prediction in dynamic networks. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM '22)*, 1245–1255.

[57] 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 International Conference on Knowledge Discovery and Data Mining (KDD '21)*, 1975–1985.

[58] Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. 2022. Algorithm and system co-design for efficient subgraph-based graph representation learning. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2788–2796.

[59] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD '18)*, 974–983.

[60] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. 2023. Towards better dynamic graph learning: New architecture and unified library. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS '23)*.

[61] Yanping Zheng, Hanzhi Wang, Zhewei Wei, Jiajun Liu, and Sibo Wang. 2022. Instant graph neural networks for dynamic graphs. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD '22)*, 2605–2615.

[62] Le-Kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '18)*. Sheila A. McIlraith and Kilian Q. Weinberger (Eds.), 571–578.

[63] Tao Zou, Yuhao Mao, Junchen Ye, and Bowen Du. 2024. Repeat-aware neighbor sampling for dynamic graph learning. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD '24)*, 4722–4733.

Received 30 October 2024; revised 10 April 2025; accepted 1 June 2025