ıLoRE: Dynamic Graph Representation with Instant Long-term Modeling and Re-occurrence Preservation

Siwei Zhang
swzhang22@m.fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai, China

Yun Xiong*
yunx@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai, China

Yao Zhang
yaozhang@fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai, China

Xixi Wu
21210240043@m.fudan.edu.cn
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai, China

Yiheng Sun
sunyihengcn@gmail.com
Tencent Weixin Group
Shenzhen, China

Jiawei Zhang
jiawei@ifmlab.org
IFM Lab, Department of Computer Science, University of California, Davis
CA, USA

ABSTRACT

Continuous-time dynamic graph modeling is a crucial task for many real-world applications, such as financial risk management and fraud detection. Though existing dynamic graph modeling methods have achieved satisfactory results, they still suffer from three key limitations, hindering their scalability and further applicability. i) **Indiscriminate updating.** For incoming edges, existing methods would indiscriminately deal with them, which may lead to more time consumption and unexpected noisy information. ii) **Ineffective node-wise long-term modeling.** They heavily rely on recurrent neural networks (RNNs) as a backbone, which has been demonstrated to be incapable of fully capturing node-wise long-term dependencies in event sequences. iii) **Neglect of re-occurrence patterns.** Dynamic graphs involve the repeated occurrence of neighbors that indicates their importance, which is disappointedly neglected by existing methods.

In this paper, we present **ıLoRE**, a novel dynamic graph modeling method with **i**nstant node-wise **Lo**ng-term modeling and **Re**-occurrence preservation. To overcome the indiscriminate updating issue, we introduce the Adaptive Short-term Updater module that will automatically discard the useless or noisy edges, ensuring ıLoRE's effectiveness and instant ability. We further propose the Long-term Updater to realize more effective node-wise long-term modeling, where we innovatively propose the Identity Attention mechanism to empower a Transformer-based updater, bypassing the limited effectiveness of typical RNN-dominated designs. Finally, the crucial re-occurrence patterns are also encoded into a graph module for informative representation learning, which will further improve the expressiveness of our method. Our experimental results on real-world datasets demonstrate the effectiveness of our ıLoRE for dynamic graph modeling.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Computing methodologies** → **Learning latent representations**; *Neural networks.*

KEYWORDS

Dynamic Graphs; Representation Learning; Data Mining

ACM Reference Format:
Siwei Zhang, Yun Xiong, Yao Zhang, Xixi Wu, Yiheng Sun, and Jiawei Zhang. 2023. ıLoRE: Dynamic Graph Representation with Instant Long-term Modeling and Re-occurrence Preservation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614926

1 INTRODUCTION

In real-world scenarios, graphs are often constantly evolving over time, where objects (nodes) and their interactions (edges) can emerge and change along a temporal sequence. Such graphs are known as continuous-time dynamic graphs[1] [34]. Graph Neural Networks (GNNs) for modeling static graphs [14, 18, 26] fail to encode the temporal dependencies, leading to inferior performance when applied to dynamic graphs. Fortunately, Temporal Graph Networks (TGNs) [11, 19, 23, 33, 36] proposed in recent years effectively learn the temporal representation of dynamic graphs. TGNs focus on developing effective aggregation methods for incorporating historical neighbors, such as self-attention [33] and summation [23]. Most TGNs utilize a memory module to record nodes' historical behavior, enabling them to make predictions about future events. Despite their effectiveness, existing TGNs still have some key limitations:

Indiscriminate updating. TGNs indiscriminately update the memory of every node by encoding the information from each

*Corresponding author

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614926

[1] For simplicity, we use "dynamic graph" in the following text.

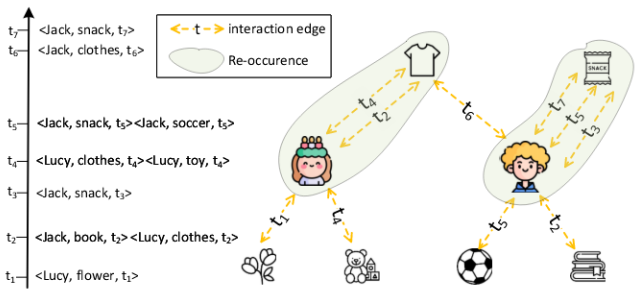


Figure 1: An example of purchase event sequences with time order (left) and the corresponding dynamic graph (right).

incoming edge [19, 33]. Indiscriminate updating would increase the redundant computational time consumption for the models and also diminish the models' ability to provide results instantly. It significantly limits their deployment in concrete industrial application tasks, especially in those that take instant ability into consideration such as financial risk management or fraud detection [24, 29]. Another issue is that indiscriminate updating may introduce useless or noisy edges [4], which will further pollute and adversely affect the quality of representation generation.

Ineffective node-wise long-term modeling. Unlike the adjacency matrix of static graphs, dynamic graphs are represented as event sequences with time order, as shown in Figure 1. There can be a large number of events occurring around certain nodes, resulting in abundant historical neighbors. These nodes are so-called "big nodes" [4], e.g. Lucy and Jack in Figure 1, and the frequency of updates increases with the number of edges connected to them. Existing methods heavily rely on Recurrent Neural Networks (RNNs) [5, 20], and fail to fully capture the node-wise long-term dependencies, particularly in the case of big nodes. Hence, more effective modeling of node-wise long-term dependencies is necessary.

Neglect of re-occurrence patterns. In dynamic graphs, events around two nodes can occur at different time. As shown in Figure 1, this phenomenon is referred to as "re-occurrence", where multiple edges can exist between two nodes. Intuitively, the frequency of re-occurrence can serve as an indication of the importance. For instance, in the purchase dynamic graph, re-occurrence reflexes the interests of consumers. As depicted in Figure 1, Jack, who previously purchased snacks multiple times, is more likely to make another snack purchase in the future. However, TGNs have not leveraged the valuable patterns, limiting their overall effectiveness.

To address the aforementioned limitations, in this paper, we propose a novel dynamic graph modeling method named **iLoRE** (Dynamic Graph Representation with **i**nstant Node-wise **Lo**ng-term Modeling and **Re**-occurrence Preservation). iLoRE consists of three main components: i) Adaptive Short-term Updater. To estimate the effect of indiscriminate updating, a state module is proposed to adaptively determine the utility of incoming edge for short-term modeling, allowing us to either incorporate or discard it accordingly. ii) Long-term Updater. To achieve node-wise long-term modeling, we employ a Transformer-based updater instead of typical RNN-based designs. However, the distribution of a certain node in event sequences is scattered. Applying full attention in such a case will increase the learning difficulty and hampers our ability to capture node-wise long-term dependencies effectively. Therefore, we propose Identity Attention, which can re-sort, pad, chunk, and apply time-aware attention within a chunk in event sequences. For more time-sensitive cases, we employ Gaussian Range Encoding [12] and time encoding [33] to preserve the temporal information. iii) Re-occurrence Graph Module. To encode the re-occurrence patterns in dynamic graphs, we fetch the re-occurrence number of historical neighbors to indicate their importance to the central node. Specifically, we apply a graph module that leverages crucial re-occurrence features to generate the informative temporal representation for downstream tasks.

In summary, our main contributions are:

- We propose a novel dynamic graph modeling method iLoRE in this paper. Different from existing TGNs, iLoRE focuses on instant node-wise long-term modeling and re-occurrence preservation.
- We introduce a state module to determine the utility of incoming edges and enable us to selectively discard useless or noisy ones, which ensures instant ability and the effectiveness of our method.
- We propose Identity Attention, which empowers our Transformer-based updater for node-wise long-term modeling in event sequences.
- We incorporate the valuable re-occurrence features with graph module to generate more informative temporal representation.
- We conduct extensive experiments on dynamic graphs, demonstrating that iLoRE has robust performance in various tasks.

2 RELATED WORK

2.1 Dynamic Graph Modeling

As research on dynamic graphs has become increasingly in-depth, dynamic graph modeling has seen rapid development in recent years [30, 31, 35, 36]. These methods can be roughly divided into two categories: sequential models and graph models.

Early works [7, 11, 23] belong to sequential models, which regard dynamic graphs as event sequences, limiting each node to receiving information from at most one-hop historical neighbors. To address this problem, the authors [33] present the first graph model that proposes a temporal attention layer to capture information from multi-hop historical neighbors, which achieves perfect results. Subsequently, many graph models emerged such as [4, 19, 22, 29, 31, 36, 38], further increasing the popularity of dynamic graph modeling. However, recent research has found that the "graph module" in graph models is not necessary [6, 28]. In [6], the authors simply use a Multi-Layer Procedure (MLP) to model one-hop historical neighbors' information and achieve best results than previous graph models, causing researchers to reconsider the necessity of graph modules.

Currently, there are few models that consider both these two types of methods. Our proposed method is based on Transformer for long-term modeling in event sequences, followed by a graph module with re-occurrence features for representation generation.

2.2 Transformers for Graph Learning

Transformer [25] is an innovative model for processing sequential data. Its self-attention mechanism allows it to perceive longer sequences, which is of great importance in the field of long-sequence modeling. Currently, Transformer has been successfully applied in many fields, such as computer vision [3, 9, 16], natural language processing [1, 8, 15], and time series prediction [13, 32, 37].

In static graphs, researchers have proposed many Transformer-based methods for static graph modeling [14, 18, 21]. The authors [10] propose a graph transformer layer with Laplacian Eigenvectors to encode graph structure. In [27], the authors utilize a graph transformer attention layer to extract information and capture the neighboring correlations, which achieves effective performance.

Currently, most works in the field of dynamic graphs are based on RNNs, and there are few works that use the Transformer as the backbone. Therefore, our proposed model extends Transformer into node-wise long-term modeling in dynamic graphs, opening up new possibilities in the field of dynamic graph modeling.

3 NOTATOIN AND TERMINOLOGY PRELIMINARIES

Definition 3.1. **Dynamic Graph.** A dynamic graph is a graph whose edges contain temporal information, *i.e.*, timestamps. We denote a dynamic graph as a sequence of timestamped evolving graphs $\mathcal{G} = (\mathcal{G}(t_0), \mathcal{G}(t_1), ...)$, where $t_k < t_{k+1}$ and $\mathcal{G}(t_{k+1})$ is generated from $\mathcal{G}(t_k)$ with the edges whose timestamp is t_{k+1}. We represent an edge between nodes i and j at time t as a tuple (i, j, t) with an edge feature $\mathbf{e}_{ij}(t)$.

A dynamic graph can also be viewed as event sequences $\mathcal{E}$. Each event $(i, j, t_{k+1}) \in \mathcal{E}$ can be seen as the new edge of $\mathcal{G}(t_{k+1})$ compared to $\mathcal{G}(t_k)$, and all of the events are sorted by timestamps. For the remaining part of this paper, in referring to the incoming dynamic graph edge sequences, we will misuse the terminologies of "dynamic graph edge set" and "event sequence" interchangeably without distinguishing their differences.

Definition 3.2. **Dynamic Graph Modeling.** Given a dynamic graph edge set or event sequences $\mathcal{E}$, for each event $(i, j, t) \in \mathcal{E}$, the goal of dynamic graph modeling is to learn a mapping function $f : (i, j, t) \mapsto \mathbf{z}_i(t), \mathbf{z}_j(t)$, where $\mathbf{z}_i(t), \mathbf{z}_j(t) \in \mathbb{R}^d$ respectively represent temporal representation of nodes i and j, and d is the vector dimension.

Besides the terminologies defined above, several other important notations used in this paper are summarized in Table 1.

4 PROPOSED METHOD

We first define the short- and long-term behavior of nodes with the window-split technique in event sequences, which are encoded as short- and long-term memory, respectively. Our proposed ILoRE has three main parts, including i) the Adaptive Short-term Updater, which achieves instant short-term modeling within a window; ii) the Long-term Updater, which captures nodes' long-term dependencies across multiple windows; iii) and the Re-occurrence Graph Module, which encodes re-occurrence patterns within a graph module for representation.

Table 1: Important notations

Symbol	Definition
$\mathcal{M}_i^S(t)$	Short-term memory of node i at t
$\mathcal{M}_i^L(t)$	Long-term memory of node i at t
$\mathcal{S}_i(t)$	Node state of node i at t
$\mathbf{X}_{i,\mathcal{R}}(t)$	Re-occurrence features of node i's neighbors at t
$\mathbf{z}_i(t)$	Temporal representation of node i at t
n	Chunk size (hyper-parameter)
b	Block number of Transformer (hyper-parameter)

As illustrated in Figure 2, in the Adaptive Short-term Updater, a state module is proposed to automatically discard useless or noisy edges to ensure the effectiveness and instant ability of our method. Meanwhile, in the Long-term Updater, to empower node-wise long-term modeling ability for event sequences, Identity Attention is proposed to optimize the Transformer-based updater, which can re-sort, pad, chunk, and apply time-aware attention within a chunk. For more time-sensitive cases, we employ Gaussian Range Encoding [12] and time encoding [33] to preserve the temporal information. What's more, in the Re-occurrence Graph Module, we incorporate the valuable re-occurrence features into a graph attention module for informative temporal representation generation. We will introduce these components in the following subsections.

4.1 Node-wise Short- and Long-term Modeling

4.1.1 Window-split Technique. To perform node-wise long-term modeling, we propose to split the event sequence into subsequences according to a pre-defined window size s. Given event sequences $\mathcal{E} = \{e_1, e_2, ..., e_r\}$ where r is the event length, we define a window set $\mathbf{w} = \{w_1, w_2, ..., w_{\lceil r/s \rceil}\}$, where $w_i = \{e_{i \cdot s - s + 1}, e_{i \cdot s - s + 2}, ..., e_{i \cdot s} | i \leq \lceil r/s \rceil\}$ contains s events.

In this paper, we use short- and long-term memory, $\mathcal{M}^S$ and $\mathcal{M}^L$, to embed the short- and long-term behavior of each node, respectively. The memory of each node i, $\mathcal{M}_i^S$ and $\mathcal{M}_i^L$, is initialized as the zero vector and will be updated over time. For given node i at time t, we use the events within the same window to perform short-term modeling, *i.e.*, updating $\mathcal{M}_i^S(t)$, and perform long-term modeling across multiple windows, *i.e.*, updating $\mathcal{M}_i^L(t)$. Note that once the long-term memory is updated, the short-term memory will be reset to zero.

4.1.2 Message Generation. Given an event of node i at time t in window w_i, a message $\mathbf{m}_i(t)$ is generated to update the short-term memory of i, $\mathcal{M}_i^S(t)$. Assume that nodes i and j have an event at time t, (i, j, t), with the feature vector $\mathbf{e}_{ij}(t)$, we generate two messages with the long-term memory of i and j, $\mathcal{M}_i^L(t)$ and $\mathcal{M}_j^L(t)$:

$$\begin{aligned} \mathbf{m}_i(t) &= \textsc{Msg}\left(\mathcal{M}_i^L(t), \mathcal{M}_j^L(t), \mathbf{e}_{ij}(t), \Phi\left(t - t_i^-\right)\right), \\ \mathbf{m}_j(t) &= \textsc{Msg}\left(\mathcal{M}_j^L(t), \mathcal{M}_i^L(t), \mathbf{e}_{ij}(t), \Phi\left(t - t_j^-\right)\right), \end{aligned} \tag{1}$$

where $\textsc{Msg}(\cdot)$ is the message function and t_*^- is the time that node i/j last updated. $\Phi(\cdot)$ is the time encoding used in [19]. The reason why we conduct long-term memory for message generation is that it contains more expressive and valuable information compared

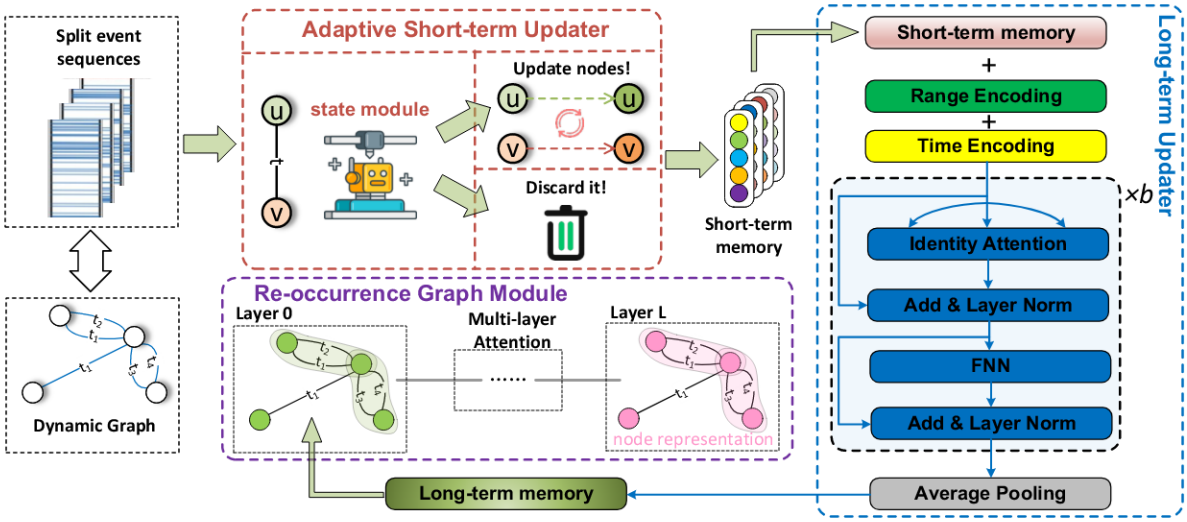


Figure 2: A schematic view of our proposed model for dynamic graph modeling. A dynamic graph can be represented as the event sequences. With the window-split technique in event sequences, in the Adaptive Short-term Updater, a state module is proposed to automatically determine whether to use the incoming event to update nodes for short-term modeling or discard it. Meanwhile, in the Long-term Updater, we propose Identity Attention to empower a Transformer-based updater for node-wise long-term modeling. Gaussian Range Encoding and time encoding are utilized to make our updater more time-sensitive. Finally, we generate the node temporal representation for downstream tasks by applying a multi-layer graph attention module based on the re-occurrence features with nodes' long-term memory.

with short-term one. For simplicity, we implement the widely-used *identity* message function that outputs the inputs. Moreover, in each window w_i, we apply the simplest *most recent* message aggregator that only considers the most recent message for each node [19].

4.2 Adaptive Short-term Updater

To model the node-wise short-term behavior meanwhile ensuring instant ability, we *adaptively update* the short-term memory of node i before t, $\mathcal{M}_i^S(t^-)$, with the message of i at t, $\mathbf{m}_i(t)$. We propose the node state module.

In this module, each node i has a node state at time t, $\hat{\mathcal{S}}_i(t) \in (0,1)$, which is evolving along with timestamps. We have:

$$\mathcal{S}_i(t) = \text{Bernoulli}\left(\hat{\mathcal{S}}_i(t)\right), \tag{2}$$

where Bernoulli$(\cdot)$ denotes sampling from a Bernoulli distribution parameterized by $\hat{\mathcal{S}}_i(t)$, and $\mathcal{S}_i(t) \in \{0,1\}$.

Then, $\mathcal{S}_i(t)$ is utilized to determine whether we update the short-term memory of node i before time t, $\mathcal{M}_i^S(t^-)$:

$$\mathcal{M}_i^S(t) = \mathcal{S}_i(t) \cdot \textsc{Upd}\left(\mathcal{M}_i^S(t^-), \mathbf{m}_i(t)\right) + (1 - \mathcal{S}_i(t)) \cdot \mathcal{M}_i^S(t^-) \tag{3}$$

where $\textsc{Upd}(\cdot)$ is a learnable update module for node-wise short-term modeling, and we use GRU [5] in practice. Afterward, we update the node state with its short-term memory in the following timestamps, t^+:

$$\Delta\hat{\mathcal{S}}_i(t) = \sigma\left(\mathbf{W}_p \cdot \mathcal{M}_i^S(t) + \mathbf{b}_p\right) \tag{4}$$

$$\hat{\mathcal{S}}_i(t^+) = (1 - \mathcal{S}_i(t)) \cdot \Delta\hat{\mathcal{S}}_i(t) + \mathcal{S}_i(t) \cdot \left(\hat{\mathcal{S}}_i(t) - \alpha \min\left(\Delta\hat{\mathcal{S}}_i(t), \mathcal{S}_i(t)\right)\right) \tag{5}$$

where $\mathbf{W}_p$ and $\mathbf{b}_p$ are learnable parameters, $\sigma(\cdot)$ is the sigmoid function, and $\alpha \in (0,1)$ is a control hyper-parameter that ensures the node state is positive. The node state module encodes the observation that the likelihood of a new update operation decreases with the frequency of node-wise updating. Whenever $\mathcal{M}_i^S(t)$ updates, the pre-activation of the node state for the following timestamp, $\hat{\mathcal{S}}_i(t^+)$, is decreased by $\Delta\hat{\mathcal{S}}_i(t)$. On the other hand, if the update is omitted, the accumulated value is flushed and $\hat{\mathcal{S}}_i(t^+) = \Delta\hat{\mathcal{S}}_i(t)$. In this way, we can selectively update the nodes with incoming edges, ensuring effectiveness and instant ability.

4.3 Long-term Updater

We consider both node-wise short- and long-term modeling in this paper. As mentioned in section 4.2, the recent behavior of node i at time t in window w_i is recorded by short-term memory, $\mathcal{M}_i^S(t)$, with the window-split technique. In this section, we introduce a Transformer-based updater that can embed the node-wise long-term behavior, *i.e.*, updating $\mathcal{M}^L$, using the short-term memory in multiple windows.

4.3.1 Gaussian Range Encoding and time encoding. The order is pretty important in event sequences. Most Transformer-based methods use positional encoding [25] that is defined on a single point:

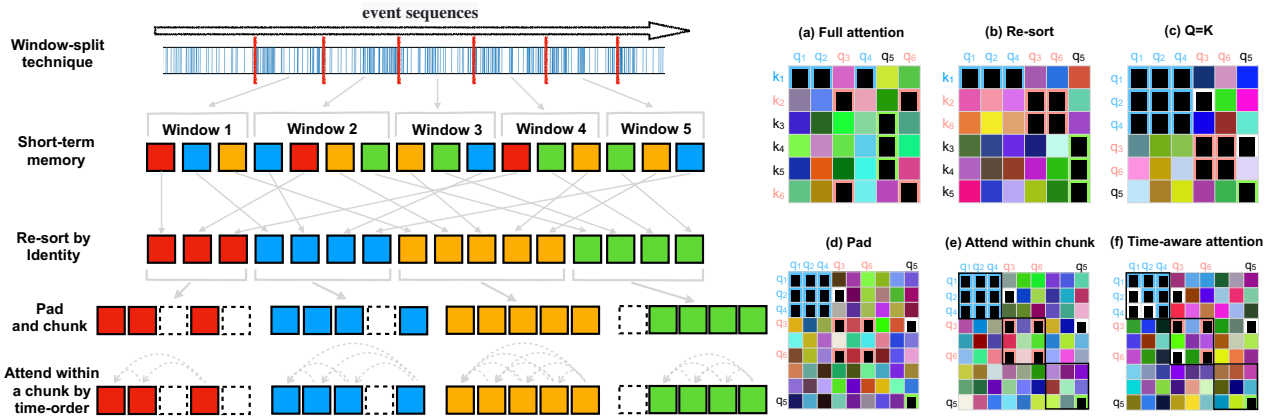


Figure 3: Simplified description of Identity Attention (left) and attention matrices that need to be learned in each step (a-f on right). With the window-split technique, we can take node-wise long-term modeling in multiple widows, e.g., 5 windows, with nodes' short-term memory. Note that different colors denote different node identities. The Identity Attention re-sorts, pads, chunks, and attends within a chunk with time order, which can densify the attention matrix in a chunk, greatly reducing the difficulty to learn the attention matrix and thus increasing our ability to capture node-wise long-term dependencies effectively.

They employ a highly discriminative encoding for every single point. It can not align with the nature of time in event sequences because the timestamps are continuous. To make the model more order-aware, we use a *range-based* encoding method. Therefore, we employ Gaussian Range Encoding [12].

Formally, we propose $\mathbf{B} \in \mathbb{R}^{d\times k}$ as the normalized weights from k Gaussian distributions, where d denotes the dimension of the input vector. It can be shown as follows:

$$\mathbf{B} = \text{softmax}\left(B\right), \tag{6}$$

where $B \in \mathbb{R}^{d\times k}$ is a matrix whose attributes are sampled from k ranges. In matrix B, each cell b_{ij} shows the contribution of the j-th Gaussian ranges for position i, which can be represented as:

$$b_{ij} = -\frac{\left(i-\mu^{(j)}\right)^2}{2\sigma^{(j)2}} - \log\left(\sigma^{(j)}\right), \tag{7}$$

where $\mu^{(j)}$ and $\sigma^{(j)}$ are the mean and standard deviation of j-th Gaussian ranges, respectively. For implementation, we set these two parameters to be learnable. Then, the Gaussian Range Embedding is generated by adding the range embeddings to the input vector X:

$$\textsc{Gaussian}(X) = X + \mathbf{B}\cdot\mathbf{E}, \tag{8}$$

where $\mathbf{E} \in \mathbb{R}^{k\times d}$ is a learnable matrix. This approach uses k learnable Gaussian ranges to express different positions, which makes our position encoding more continuous. Moreover, we adopt classic time encoding [33] widely used in dynamic graph modeling to better preserve temporal information.

4.3.2 Identity Attention. Figure 3 illustrates the motivation and the process of Identity Attention. Figure 3a expresses the attention matrix that needs to be learned when using full attention for node-wise long-term modeling, where different colors of k and q represent different nodes' identities. Since the distribution of a node at different times is scattered in the sequence, the attention matrix for full attention is typically *sparse*, making it difficult to learn. Therefore, we propose Identity Attention, which can *densify* the attention matrix within a chunk by re-sorting (Figure 3b, c), padding (Figure 3d), chunking (Figure 3e), and attending within a chunk (Figure 3f), greatly reducing the learning difficulty and enhancing our ability to node-wise long-term modeling in event sequences.

We first rewrite the equation of full attention. For a query position i, its attention to position j can be represented as $\mathbf{o}_{ij}$:

$$\mathbf{o}_{ij} = \sum_{j\in\mathcal{P}_i} \exp\left(q_i\cdot k_j + \mathbf{z}\left(i,\mathcal{P}_i\right)\right) v_j, \tag{9}$$

where $\mathcal{P}_i = \{j : j \le i \ or \ j > i\}$. Note that $\mathcal{P}_i$ represents the set that the query position i can attend to, and $\mathbf{z}$ denotes the partition function, e.g., softmax. Notably, we omit the parameter $\sqrt{d_k}$ [25].

We can use a positional encoding function $\mathbf{m}(\cdot,\cdot)$ to fit the attention between position i and position j that i can attend to:

$$\mathbf{o}_{ij} = \sum_{j\in\mathcal{P}_i} \exp\left(q_i\cdot k_j + \mathbf{m}\left(j,\mathcal{P}_i\right) + \mathbf{z}\left(i,\mathcal{P}_i\right)\right) v_j, \tag{10}$$

where $\mathbf{m}(\cdot,\cdot)$ usually applies a single point positional encoding [25].

Now we turn to Identity Attention, which we can consider as the constraint of $\mathcal{P}_i$.

Re-sort. This step aims to cluster temporal nodes in the same identity. We use bucket sort to rearrange the entire sequence according to the order of identity, where position i changes after sorting, *i.e.*, $i \mapsto c_i$. In the sorted attention matrix, nodes with the same identity will be clustered, as shown in Figure 3b. We have:

$$\mathcal{P}_i = \left\{j : \mathbf{Id}\left(q_i\right) = \mathbf{Id}\left(k_j\right)\right\}, \tag{11}$$

where $\mathbf{Id}(\cdot)$ denotes the identity of a correlated temporal node. For simplicity, we let $Q = K$ as represented in Figure 3c.

Pad and chunk. Since the frequency of node updating is different in different windows, the number of temporal nodes in each bucket is unequal. In practice, we employ zero vectors to pad the temporal nodes that do not have updated in corresponding windows

as depicted in Figure 3d. Moreover, we apply batching approaches to chunk and concentrate attention within each chunk (after sorting and padding), shown in Figure 3e. Formally, we have:

$$\widetilde{\mathcal{P}}_i = \left\{ j : \left\lfloor \frac{c_i}{n} \right\rfloor - 1 \leq \left\lfloor \frac{c_j}{n} \right\rfloor \leq \left\lfloor \frac{c_i}{n} \right\rfloor \right\}, \tag{12}$$

where $n \in \mathbb{R}^+$ is a hyper-parameter that represents the chunk size. In Long-term Updater, n is also the number of windows where Transformer performs long-term modeling at once in event sequences.

Attend within a chunk by time-order. Then, we implement Gaussian Range Encoding and time-aware attention within a chunk as illustrated in Figure 3f. Formally, we also use $\mathbf{m}(\cdot,\cdot)$ as our positional encoding function, which is defined as:

$$\mathbf{m}\left(j, \widetilde{\mathcal{P}}_i\right) = \begin{cases} \textsc{Gaussian}\left(i, \widetilde{\mathcal{P}}_i\right), & \text{if } j \in \widetilde{\mathcal{P}}_i \\ -\infty, & \text{otherwise,} \end{cases} \tag{13}$$

where $\textsc{Gaussian}(\cdot,\cdot)$ is Gaussian Range Encoding in Section 4.3.1. Considering that in event sequences, the event that happened at time t can only attend to the past events before t, we propose time-aware attention, which is defined as:

$$\mathbf{t}\left(j, \widetilde{\mathcal{P}}_i\right) = \begin{cases} \textsc{Time}(i,t), & \text{if } j \in \widetilde{\mathcal{P}}_i \text{ and } j \leq i \\ -\infty, & \text{otherwise,} \end{cases} \tag{14}$$

where $\textsc{Time}(\cdot,\cdot)$ is the time encoding in Section 4.3.1.

Summarily, the final Identity Attention can be represented as:

$$\mathbf{o}_{ij} = \sum_{j \in \widetilde{\mathcal{P}}_i} \exp\left(q_i \cdot k_j + \mathbf{m}\left(j, \widetilde{\mathcal{P}}_i\right) + \mathbf{t}\left(j, \widetilde{\mathcal{P}}_i\right) + \mathbf{z}\left(i, \widetilde{\mathcal{P}}_i\right)\right) v_j. \tag{15}$$

Each component that is negative infinity will force our attention to zero. Similar to full attention, we can also apply the multi-head technique in Identity Attention.

4.3.3 Transformer. We employ a standard Transformer encoder for node-wise long-term modeling. Transformer is equipped with stacking b Multi-head Identity Attention (MIA) and Feed-Forward Network (FFN) blocks. Each block employs a residual connection. We use ReLU between the two MLPs in each FFN block and apply Layer Normalization (LN) before each block.

Thanks to the window-split and chunk technique, the input of Transformer is the short-term memory $\mathcal{M}^S$ that is updated by the events in recent n windows before w_i, *i.e.*, $\{w_{i-n+1}, ..., w_i | i \leq \lceil r/s \rceil\}$, and it is denoted as $\mathbf{Z}^0 \in \mathbb{R}^{l_i \times d}$ where l_i is the length of events. The output embedding of the b-th layer is denoted by $\boldsymbol{H} = \boldsymbol{Z}^b \in \mathbb{R}^{l'_i \times d}$ where l'_i is the length of sequence after padding.

The long-term memory of node i at time t, $\mathcal{M}_i^L(t)$, is derived by averaging their related embedding in $\boldsymbol{H}$:

$$\mathcal{M}_i^L(t) = \text{MEAN}\left(\boldsymbol{H}\left[i,:\right]\right) \in \mathbb{R}^d. \tag{16}$$

4.4 Re-occurrence Graph Module

We aim to encode the re-occurrence features into the graph module, which refers to the property that two nodes may interact at different timestamps. Intuitively, the re-occurrence number of a historical neighbor indicates its importance to the central node.

For given a node i and its historical neighbors at time t, $\mathcal{N}_i(t)$, we count the number of re-occurrence of each neighbor, which is represented as $\mathcal{R}_i(t) \in \mathbb{R}^{|\mathcal{N}_i(t)| \times 1}$. Then, we apply a function $f(\cdot)$ to encode the re-occurrence features of historical neighbors by:

$$\mathbf{X}_{i,\mathcal{R}}(t) = f\left(\mathcal{R}_i(t)\right) \in \mathbb{R}^{|\mathcal{N}_i(t)| \times d}, \tag{17}$$

where $f(\cdot)$ is a three-layer perceptron with ReLU activation, whose input and output dimensions are 1 and d, respectively.

For node i at time t, we compute the embedding $\mathbf{z}_i(t)$ with its long-term memory $\mathcal{M}_i^L(t)$. We aggregate its historical neighbors' long-term memory, $\mathcal{M}_j^L(t_j)$ where $j \in \mathcal{N}_i(t)$, using an attention mechanism as follows:

$$h_i^l(t) = \text{MLP}^{(l)}\left(\mathbf{h}_i^{l-1}(t) \,\|\tilde{\mathbf{h}}_i^l(t)\right), \tag{18}$$

$$\tilde{\mathbf{h}}_i^l(t) = \text{Att}^{(l)}\left(\bigodot_{j \in \mathcal{N}_i(t)} \left(\mathbf{h}_j^{l-1}(t) \,\|\mathbf{e}_{ij}(t_j) \,\|\Phi(t - t_j) \,\|\mathbf{X}_{j,\mathcal{R}}(t_j)\right)\right) \tag{19}$$

where $\bigodot$ denotes the stacking operation and $\text{Att}(\cdot)$ is the graph attention used in [19]. Note that the input $\mathbf{h}_i^0(t) = \mathcal{M}_i^L(t)$ and the node representation $\mathbf{z}_i(t) = \mathbf{h}_i^L(t)$ where L is the layer number.

Algorithm 1: Traning LoRE (one epoch).

input : Dynamic graph edge set $\mathcal{E}$; Short-term memory $\mathcal{M}^S$; Long-term memory $\mathcal{M}^L$; Node state $\hat{\mathcal{S}}$; Chunk size n.

```
Initialize M^S, M^L, Ŝ ← 0 ;
foreach batch {(i, j, t)} ⊆ E do
    Split batch into n windows, {w_1, ..., w_n} ;
    Initialize M(t) ← 0 ;
    foreach w_i ∈ {w_1, ..., w_n} do
        Sample S(t) ~ Bernoulli(Ŝ(t)) ;
        Update short-term memory M^S(t) by Equation 3 ;
        Update node state Ŝ(t) by Equation 4 ;
        Record M^S(t) to M(t) ;
    end
    Update M^L(t) ← Upd(M(t)) with Identity Attention ;
    Compute X_{i,R}(t), X_{j,R}(t) by Equation 17 ;
    Compute z_i(t), z_j(t) ← Emb(M^L(t), X_{i,R}, X_{j,R}) ;
    Compute p_ij(t), p_ik(t) by Equation 20 ;
    Compute temporal link prediction loss L by Equation 21
      and backward ;
end
```

4.5 Training

4.5.1 Error Grandients. Our method is differential except for the Bernoulli process in Equation 2, which is a binary value as the output. We employ the widely-used straight-through estimator [2], which implements the identity to approximate the step function for gradients computation during the backward pass: $\frac{\partial \text{Bernoulli}(x)}{\partial x} = 1$.

4.5.2 Loss Function. We take temporal link prediction as our self-supervised task. For the representation of nodes i and j at time t, $\mathbf{z}_i(t)$ and $\mathbf{z}_j(t)$, we compute the probability of having interaction

Table 2: Average Precision (AP(%) ± Std) for temporal link prediction in transductive and inductive setting. The result %d that is bolded is the best result and the second is %d.

(a) Transductive Setting.

	Wikipedia	Reddit	MOOC	LastFM
CTDNE	79.42 ± 0.4	73.76 ± 0.5	65.34 ± 0.7	57.25 ± 1.0
JODIE	94.62 ± 0.5	91.11 ± 0.3	76.50 ± 1.8	68.77 ± 3.0
TGAT	95.34 ± 0.1	98.12 ± 0.2	60.97 ± 0.3	53.36 ± 0.1
DyRep	94.59 ± 0.2	97.98 ± 0.1	75.37 ± 1.7	68.77 ± 2.1
TGN	98.46 ± 0.1	98.70 ± 0.1	85.88 ± 3.0	80.69 ± 0.2
CAW	98.63 ± 0.1	98.39 ± 0.1	80.15 ± 0.3	81.29 ± 0.1
TIGER	98.38 ± 0.1	99.04 ± 0.1	89.64 ± 0.9	87.85 ± 0.9
GraMixer	97.95 ± .03	97.31 ± .01	82.78 ± 0.2	67.27 ± 2.1
PINT	98.78 ± 0.1	99.03 ± .01	85.14 ± 1.2	88.06 ± 0.7
Ours	**98.98 ± 0.3**	**99.11 ± 0.4**	**90.44 ± 1.0**	**91.39 ± 0.1**

(b) Inductive Setting.

	Wikipedia	Reddit	MOOC	LastFM
CTDNE	-	-	-	-
JODIE	93.11 ± 0.4	94.36 ± 1.1	77.83 ± 2.1	82.55 ± 1.9
TGAT	93.99 ± 0.3	96.62 ± 0.3	63.50 ± 0.7	55.65 ± 0.2
DyRep	92.05 ± 0.3	95.68 ± 0.2	78.55 ± 1.1	81.33 ± 2.1
TGN	97.81 ± 0.1	97.55 ± 0.1	85.55 ± 2.9	84.66 ± 0.1
CAW	98.24 ± .03	97.81 ± 0.1	81.42 ± 0.2	85.67 ± 0.5
TIGER	98.45 ± 0.1	98.39 ± 0.1	89.51 ± 0.7	90.14 ± 1.0
GraMixer	96.65 ± .02	95.26 ± .02	81.41 ± 0.2	82.11 ± 0.4
PINT	98.38 ± .04	98.25 ± .04	85.39 ± 1.0	91.76 ± 0.7
Ours	**98.60 ± 0.3**	**98.65 ± 0.3**	**89.75 ± 0.8**	**93.29 ± 0.8**

Table 3: AUC (AUC(%) ± Std) for evolving node classification task on Wikipedia, Reddit and MOOC. The result %d that is bolded is the best result and the second is %d.

	CTDNE	JODIE	TGAT	DyRep	TGN	TIGER	GraMixer	PINT	Ours
Wikipedia	75.89 ± 0.5	84.84 ± 1.2	83.69 ± 0.7	84.59 ± 2.2	87.81 ± 0.3	86.92 ± 0.7	86.80 ± .01	87.59 ± 0.6	**91.37 ± 0.2**
Reddit	59.43 ± 0.6	61.83 ± 2.7	65.56 ± 0.7	62.91 ± 2.4	67.06 ± 0.9	69.41 ± 1.3	64.22 ± .03	67.31 ± 0.2	**71.82 ± 1.6**
MOOC	67.54 ± 0.7	66.87 ± 0.4	53.95 ± 0.2	67.76 ± 0.5	69.54 ± 1.0	72.35 ± 2.3	67.21 ± .02	68.77 ± 1.1	**73.89 ± 2.0**

between them by a two-layer MLP:

$$\hat{p}_{ij}(t) = \sigma\left(\text{MLP}\left(\mathbf{z}_i(t) \,\|\, \mathbf{z}_j(t)\right)\right), \tag{20}$$

where $\sigma(\cdot)$ is the sigmoid function. Then, we set the cross-entropy as the loss function:

$$\mathcal{L} = -\sum_{(i,j,t)\in\mathcal{E}} \left[\log \hat{p}_{ij}(t) + \log\left(1 - \hat{p}_{ik}(t)\right)\right], \tag{21}$$

where k is the negative destination node by random sampling. The pseudo-code of the ıLoRE is provided in Algorithm 1.

5 EXPERIMENTS

5.1 Datasets and Baselines

For better comparison, we conduct experiments with four widely-used public datasets [11] including Wikipedia, Reddit, MOOC, and LastFM. Notably, all datasets have no node feature, and MOOC and LastFM have no edge feature, where we assign zero vectors in each of these datasets. Except for LastFM, others share evolving node labels of source nodes, and we can conduct the node classification task on them. All datasets are split with 70%-15%-15% for training, validation, and testing as [19].

For evaluation, we choose nine dynamic graph modeling methods to compare with ours, including CTDNE [17], DyRep [23], JODIE [11], TGAT [33], TGN [19], CAW [31], TIGER [36], GraMixer [6], PINT [22]. Note that CTDNE can not be applied in the inductive setting, and CAW can not be conducted in the evolving node classification task.

5.2 Temporal Link Prediction

Firstly, we evaluate our model on the temporal link prediction task. Similar to previous dynamic graph modeling methods, we test our model under two settings: transductive and inductive. In the transductive setting, we test edges whose nodes have been seen in the training splits, while in the inductive setting, we examine the unseen nodes for temporal link prediction. We use average precision (AP) as our evaluation metric and select an equal number of negative edges as we did in Equation 21.

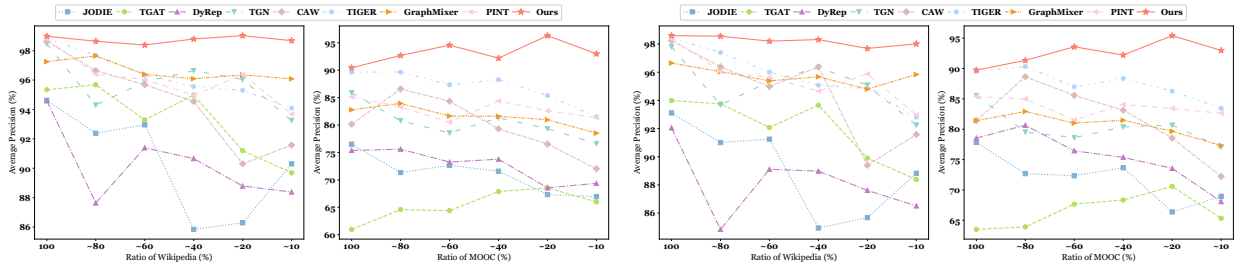
The results are shown in Table 2. Our model outperforms all baselines on all datasets in both transductive and inductive settings. This observation proves the excellent effectiveness and expressiveness of our method. For all baselines, existing sequential models, *i.e.*, [6, 11, 17, 23], perform worse than graph models. This may be owing to the fact that graph models, whose nodes can attend to multi-hop neighbors, have preserved the longer neighbors' information during training. It also gives us the motivation that there is still considerable room for improvement in sequential models.

5.3 Evolving Node Classification

To further evaluate the effectiveness of our model, we use the learned temporal representation for the evolving node classification task. In practice, we utilize temporal link prediction as a pre-training task for the models. We use Wikipedia, Reddit, and MOOC for testing as only these datasets have evolving node labels. Following [19], we input the temporal representation of node i, $\mathbf{z}_i(t)$, into a two-layer MLP to obtain the class probability of the temporal nodes and then design a training signal in Equation 21.

Table 4: P-value of the chi-square independence test on Wikipedia and MOOC.

	JODIE	TGAT	DyRep	TGN	CAW	TIGER	GraMixer	PINT	Ours
Wikipedia	0.006	0.015	0.011	0.031	0.008	0.042	0.029	0.040	**0.185**
MOOC	0.010	0.005	0.018	0.034	0.019	0.010	0.033	0.022	**0.115**



(a) Transductive temporal link prediction. **(b) Inductive temporal link prediction.**

Figure 4: The ability to node-wise long-term modeling for temporal link prediction task on Wikipedia and MOOC.

The results are presented in Table 3. Our method achieves the best performance on all datasets, further confirming the powerful dynamic graph modeling capabilities of our method. The satisfactory outcomes demonstrate that the learned representations of our method are effective for downstream tasks.

5.4 Ability to Node-wise Long-term Modeling

To validate the node-wise long-term modeling ability of models, we design experiments focusing on big nodes in dynamic graphs. Specifically, we sort all the nodes in dynamic graphs by the number of their edges, *i.e.*, node frequency, and select nodes whose node frequency is in the top $k \in \{100\%, 80\%, 60\%, 40\%, 20\%, 10\%\}$ to generate some subgraphs separately. It is worth noting that the smaller k of the subgraph, the higher proportion of big nodes, the more challenging node-wise long-term modeling. Moreover, considering that the unequal number of samples in these subgraphs may affect the credibility of the conclusion, we employ the chi-square independence test. We first conduct a contingency table with the number of successful and failed predictions that are generated from the model in each subgraph, then we calculate the P-value of the chi-square independence test p_v. Our null hypothesis is "the subgraphs and the success or failure of the predictions are independent". If $p_v > 0.05$, we can accept the null hypothesis. It indicates that different subgraphs, which contain different proportions of big nodes, have little impact on the performance of the model, confirming the model's ability to node-wise long-term modeling.

As shown in Figure 4 and Table 4, Our model outperforms all other baselines in all subgraphs. As the proportion of big nodes increases, the performance of our model remains stable, while other baselines decline, demonstrating the strong node-wise long-term modeling capability of our model in dynamic graphs. Only our model has a P-value greater than 0.05. The chi-squared test rules out concerns that may have arisen from differences in sample number among subgraphs, enhancing the credibility of the conclusion.

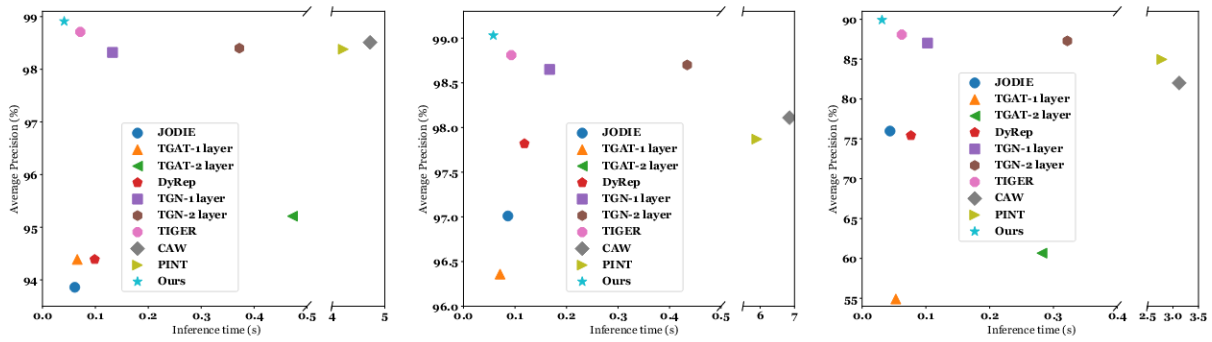
5.5 Analysis of Inference Time

To verify the effectiveness of discarding edges and the model's efficiency, we conduct comparative experiments on the inference time and the performance of models. Our experiments are performed on a Linux PC with an Intel i7 CPU (6 cores, 2.6 GHz), using the original public implementations of baselines. In industry, the inference time of a model is much more important than its training time. In the online payment platform, for example, there are billions of transaction data that are generated daily. Industry research institutes do not necessarily train models frequently, but they need to process these large amounts of daily data frequently for downstream tasks such as financial risk management [29], leading to redundant time consumption. Consequently, a model with a lower inference time has more commercial value. Thus, we compare the inference time of a batch (batch size is 100) and the performance of models.

The results are shown in Figure 5, where the closer to the upper left corner, the shorter the inference time and the better performance of the model. Our model outperforms other baselines in both inference time and performance, mainly due to the successful removal of some useless or noisy edges. We also find that [22, 31] have significantly longer inference time compared to other methods. This may be because they search for neighbors through temporal walks, which is extremely time-consuming during inference.

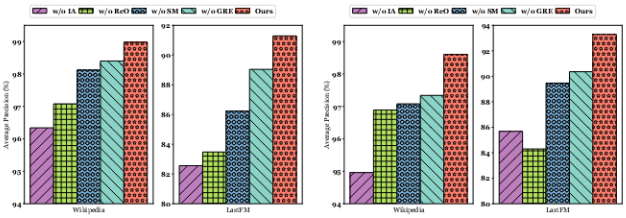
5.6 Ablation Study

We conduct an ablation study to further investigate the impact of the main innovative components in our model, including the state module (SM) in Section 4.2, Gaussian Range Encoding (GRE) in Section 4.3.1, Identity Attention (IA) in Section 4.3.2, and the Reoccurrence features (ReO) in Section 4.4. We propose four variants: w/o SM, w/o GRE, w/o IA, and w/o ReO, respectively. The w/o SM variant does not discard temporal edges and performs indiscriminate updating; The w/o GRE variant replaces the Gaussian Range Encoding with the default positional encoding used in Transformer



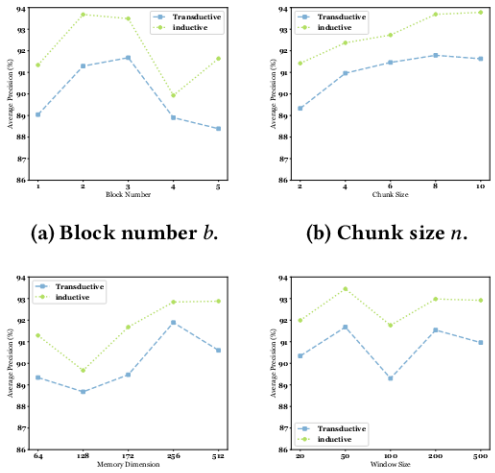
(a) Wikipedia. (b) Reddit. (c) MOOC.

Figure 5: Analysis between the inference time of a batch and the performance in transductive temporal link prediction task.



(a) Transductive setting. (b) Inductive setting.

Figure 6: Ablation study for transductive and inductive temporal link prediction task on Wikipedia and LastFM.



(a) Block number b. (b) Chunk size n.

(c) Memory dimension d. (d) Window size s.

Figure 7: Parameter study in transductive and inductive temporal link prediction task on LastFM.

[25]; The w/o IA variant replaces Identity Attention with full attention [25]; The w/o ReO variant removes the re-occurrence features from the graph module.

We report the results for link prediction on Wikipedia and LastFM, as shown in figure 6. Our model achieves the best performance when using all components, and the performance decreases when each component is removed or replaced with the default one. The Identity Attention and re-occurrence features have the most significant impact on the model, indicating that they may be able to better extract valuable information in dynamic graphs.

5.7 Parameter Study

We conduct a parameter study to better investigate the impact of main hyper-parameters in our model, including the block number of Transformer b in Section 4.3.3, the memory dimension d in Equation 16, the chunk size or window length for long-term modeling n in Equation 12, and the window size for short-term modeling s in Section 4.1.1. We conduct experiments on lastFM, and we find that increasing the value of b does not lead to better performance and the best cost-effectiveness is achieved when $b = 2$ or 3. Through our analysis of n, our model shows improved performance in modeling longer sequences, indicating that our model is able to effectively capture longer dependencies. Other hyper-parameters have varying degrees of impact on our results.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose ıLoRE, a dynamic graph modeling method with instant long-term modeling and re-occurrence preservation. We introduce a state module to enhance inference efficiency and prevent noisy information. We further propose Identity Attention to empower a Transformer-based updater for long-term modeling and successfully encode re-occurrence features into the graph module. For future work, we hope to design a dynamic graph modeling method based entirely on Transformer architecture in the future.

ACKNOWLEDGMENTS

This work is funded in part by the National Natural Science Foundation of China Projects No. U1936213, No. 62206059, the Shanghai Science and Technology Development Fund No.22dz1200704, NSF through grants IIS-1763365 and IIS-2106972, and also supported by CNKLSTISS.

REFERENCES

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[2] Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. 2017. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834* (2017).

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, 213–229.

[4] Xinshi Chen, Yan Zhu, Haowen Xu, Mengyang Liu, Liang Xiong, Muhan Zhang, and Le Song. 2021. Efficient Dynamic Graph Representation Learning at Scale. *arXiv preprint arXiv:2112.07768* (2021).

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

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

[7] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. 2016. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv preprint arXiv:1609.03675* (2016).

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Vijay Prakash Dwivedi and Xavier Bresson. 2020. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699* (2020).

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

[12] Bing Li, Wei Cui, Wei Wang, Le Zhang, Zhenghua Chen, and Min Wu. 2021. Two-stream convolution augmented transformer for human activity recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 286–293.

[13] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems* 32 (2019).

[14] Xiao Liu, Shiyu Zhao, Kai Su, Yukuo Cen, Jiezhong Qiu, Mengdi Zhang, Wei Wu, Yuxiao Dong, and Jie Tang. 2022. Mask and Reason: Pre-Training Knowledge Graph Transformers for Complex Logical Queries. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1120–1130.

[15] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

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

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

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

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

[20] Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena* 404 (2020), 132306.

[21] Yu Shi, Shuxin Zheng, Guolin Ke, Yifei Shen, Jiacheng You, Jiyan He, Shengjie Luo, Chang Liu, Di He, and Tie-Yan Liu. 2022. Benchmarking graphormer on large-scale molecular modeling datasets. *arXiv preprint arXiv:2203.04810* (2022).

[22] Amauri Souza, Diego Mesquita, Samuel Kaski, and Vikas Garg. 2022. Provably expressive temporal graph networks. *Advances in Neural Information Processing Systems* 35 (2022), 32257–32269.

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

[24] Rafaël Van Belle, Bart Baesens, and Jochen De Weerdt. 2023. CATCHM: A novel network-based credit card fraud detection method using node representation learning. *Decision Support Systems* 164 (2023), 113866.

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

[26] Petar Veličković. 2023. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology* 79 (2023), 102538.

[27] Hanrui Wang, Pengyu Liu, Jinglei Cheng, Zhiding Liang, Jiaqi Gu, Zirui Li, Yongshan Ding, Weiwen Jiang, Yiyu Shi, Xuehai Qian, et al. 2022. QuEst: Graph Transformer for Quantum Circuit Reliability Estimation. *arXiv preprint arXiv:2210.16724* (2022).

[28] 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 preprint arXiv:2105.07944* (2021).

[29] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. 2021. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 international conference on management of data*. 2628–2638.

[30] Yiwei Wang, Yujun Cai, Yuxuan Liang, Henghui Ding, Changhu Wang, Siddharth Bhatia, and Bryan Hooi. 2021. Adaptive data augmentation on temporal graphs. *Advances in Neural Information Processing Systems* 34 (2021), 1440–1452.

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

[32] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems* 34 (2021), 22419–22430.

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

[34] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, and Ruochen Kong. 2022. Dynamic network embedding survey. *Neurocomputing* 472 (2022), 212–223.

[35] Yao Zhang, Yun Xiong, Dongsheng Li, Caihua Shan, Kan Ren, and Yangyong Zhu. 2021. CoPE: Modeling Continuous Propagation and Evolution on Interaction Graph. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2627–2636.

[36] Yao Zhang, Yun Xiong, Yongxiang Liao, Yiheng Sun, Yucheng Jin, Xuehao Zheng, and Yangyong Zhu. 2023. TIGER: Temporal Interaction Graph Embedding with Restarts *(WWW '23)*. Association for Computing Machinery, New York, NY, USA, 478–488. https://doi.org/10.1145/3543507.3583433

[37] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 11106–11115.

[38] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. Tgl: A general framework for temporal gnn training on billion-scale graphs. *arXiv preprint arXiv:2203.14883* (2022).