Efficient Neural Common Neighbor for Temporal Graph Link Prediction

Xiaohui Zhang[*]
Institute for Artificial Intelligence,
Peking University
huihuang@stu.pku.edu.cn

Yanbo Wang[*]
Institute for Artificial Intelligence,
Peking University
wangyanbo@stu.pku.edu.cn

Xiyuan Wang
Institute for Artificial Intelligence,
Peking University
wangxiyuan@pku.edu.cn

Muhan Zhang[†]
Institute for Artificial Intelligence, Peking University
State Key Laboratory of General Artificial Intelligence, Peking University
muhan@pku.edu.cn

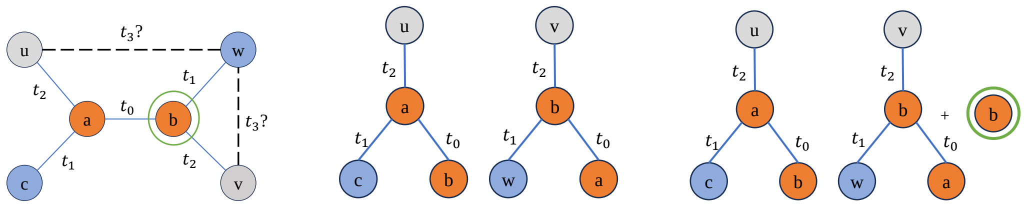
Abstract

Temporal graphs are widespread in real-world applications such as social networks, as well as trade and transportation networks. Predicting dynamic links within these evolving graphs is a key problem. Many memory-based methods use temporal interaction histories to generate node embeddings, which are then combined to predict links. However, these approaches primarily focus on individual node representations, often overlooking the inherently pairwise nature of link prediction. While some recent methods attempt to capture pairwise features, they tend to be limited by high computational complexity arising from repeated embedding calculations, making them unsuitable for large-scale datasets like the Temporal Graph Benchmark (TGB). To address the critical need for models that combine strong expressive power with high computational efficiency for link prediction on large temporal graphs, we propose Temporal Neural Common Neighbor (TNCN). Our model achieves this balance by adapting the powerful pairwise modeling principles of Neural Common Neighbor (NCN) to an efficient temporal architecture. TNCN improves upon NCN by efficiently preserving and updating temporal neighbor dictionaries for each node and by using multi-hop common neighbors to learn more expressive pairwise representations. TNCN achieves new state-of-the-art performance on Review from five large-scale real-world TGB datasets, 6 out of 7 datasets in the transductive setting and 3 out of 7 in the inductive setting on small- to medium-scale datasets. Additionally, TNCN demonstrates excellent scalability, outperforming prominent GNN baselines by up to 30.3 times in speed on large datasets. Our code is available at https://github.com/GraphPKU/TNCN.

1 Introduction

Temporal graphs are increasingly employed in contemporary real-world applications like social and transaction networks, which evolve dynamically and exhibit distinct characteristics over time.

[*]Equal Contribution
[†]Correspondence to: Muhan Zhang <muhan@pku.edu.cn>



(a) a temporal graph (b) computation tree of u and v (c) common neighbor method

Figure 1: Figure (a) shows a failure case of link prediction based on node-wise representation learning. Such methods cannot distinguish node u and v because they possess the same temporal computation tree in Figure (b), thus generating the same node representation. However, when we try to learn their pair-wise representation, i.e. (u, w) and (v, w), we can observe that v has a temporal common neighbor b with node w while u doesn't, as shown in Figure (c). Thus with the same computation graph, we only need to utilize the extra node b's embedding to distinguish (u, w) and (v, w).

Concurrently, Graph Neural Networks (GNNs) [1] have emerged as prominent tools for graph representation learning, typically learning node embeddings by iteratively aggregating information from neighbors and demonstrating strong performance on various tasks. However, the temporal dimension introduces significant challenges; the discrete or continuous timestamps associated with graph edges define an evolutionary process and impose causality constraints, rendering many static GNN methodologies not directly applicable. This has led to the development of specialized temporal GNNs. Consequently, specialized temporal GNNs have been developed, often focusing on achieving both effective representation learning and computational efficiency for these large, dynamic structures. For instance, memory-based approaches like Temporal Graph Networks (TGN) [2–4] are designed to efficiently learn short- and long-term dependencies, while Transformer-based models [5, 6] utilize attention mechanisms to capture complex relationships.

Despite their advancements, many specialized temporal GNNs primarily generate node-wise representations. While effective for certain tasks, this focus can be insufficient for link prediction, where accurately capturing the relationship between pairs of nodes is critical. Node-wise methods may fail to distinguish between nodes that have similar individual features or local neighborhoods but different propensities to connect with a target node (as illustrated in Figure 1), thereby overlooking crucial relational patterns. Acknowledging such limitations, approaches from static graph link prediction, like the labeling trick [7, 8] that emphasizes pair-wise representations, have shown considerable success. Consequently, researchers have extended these pair-wise learning concepts to temporal graphs, often by leveraging information from the evolving local neighborhoods of node pairs [9–11]. However, these more expressive graph-based temporal models frequently incur substantial computational and memory costs, arising from the need to extract and process temporal neighborhood information for each prediction, which can hinder their application to large-scale scenarios.

This trade-off between computationally demanding expressive models and more efficient but potentially less informative ones highlights a critical need. Addressing this, we propose the **Temporal Neural Common Neighbor (TNCN)** model, which is designed to achieve both high efficiency and strong expressive power for temporal link prediction. TNCN builds upon a memory-based backbone, ensuring operational efficiency comparable to sequential update models. Crucially, it incorporates a Neural Common Neighbor component [12]. This component is augmented with advanced operational techniques and extended for multi-hop common neighbor consideration, allowing TNCN to effectively model sophisticated link heuristics and learn detailed pairwise representations while retaining the efficiency of its memory-based foundation. As a result, TNCN is well-suited for large-scale temporal graph link prediction.

We conducted experiments on five large-scale real-world temporal graph datasets from TGB, where TNCN achieved new SOTA results on Review. Furthermore, evaluations on traditional small- to medium-scale datasets revealed TNCN achieving SOTA performance on 6 out of 7 datasets in the transductive setting and 3 out of 7 in the inductive setting, demonstrating its effectiveness. To assess its scalability, datasets were selected with temporal edge counts ranging from $\mathcal{O}(10^5)$ to $\mathcal{O}(10^7)$ and node counts from thousands to millions. On these large-scale datasets, TNCN achieved training speedups of 2.5x$\sim$5.9x and inference speedups of 1.8x$\sim$30.3x compared to graph-based models, while its time consumption remained comparable to that of memory-based models.

2 Preliminaries

Definition 2.1. (Temporal Graph) We mainly focus on the continuous time dynamic graph (**CTDG**). A CTDG can be typically represented as a sequence of interaction events: $\mathcal{G} = \{(u_1, v_1, t_1), \cdots, (u_n, v_n, t_n)\}$, where u, v stand for source and destination nodes and $\{t_i\}$ are chronologically non-decreasing timestamps. Note that each node or edge can be attributed, that is, there may be node feature x_u for u or edge feature $e^t_{u,v}$ attached to the event (u, v, t).

Definition 2.2. (Problem Formulation) Given the events before time t^*, i.e. $\{(u, v, t) \mid \forall\, t < t^*\}$, a link prediction task is to predict whether two specified node u^* and v^* are connected at time t^*.

Definition 2.3. (Temporal Neighborhood) Given the center node u, the k-hop temporal neighbor set ($k \geq 0$) before time t is defined as $N^t_k(u)$. A node v is in $N^t_k(u)$ if there exists a k-length path between u and v, i.e. $\exists (u, w_1, w_2, \cdots, w_{k-1}, v)$ where $w_i \neq w_j, \forall i \neq j$. We also define the $(\mathbf{i}, \mathbf{j})$**-hop common neighbor set** as follows: w is an (i, j)-hop temporal common neighbor of u and v at time t if $w \in N^t_i(u)$ and $w \in N^t_j(v)$. For simplicity we will denote the set as $\text{CN}^t_{(i,j)}(u, v) = N^t_i(u) \cap N^t_j(v)$. Note that for $i = 0$ (or $j = 0$ similarly), we define the 0-hop temporal neighbor set as $N^t_0(u) = \{u\}$, and the $(0, j)$-hop common neighbor of u and v as $\text{CN}^t_{(0,j)}(u, v) = N^t_0(u) \cap N^t_j(v) = \{u\} \cap N^t_j(v)$. Finally, the K-hop temporal neighborhood of node u at time t is defined as: $\overset{K}{\underset{k=0}{\cup}} N^t_k(u)$.

With (i, j)-hop neighborhood information, we can perceive the local structure to a large extent and distinguish the difference between multi-hop common neighbors more precisely.

Definition 2.4. Memory-based Backbone. Memory-based backbone has been widely adopted by various methods like [2, 4] to tackle dynamic graph learning. Its core component is the memory module that stores the node memory representations up to a certain time t. When a new event occurs, the memory of the source and destination nodes is updated with the message produced by the event. The computation can generally be represented as follows:

$$\begin{aligned} msg^t_{src}(u, v) &= \boldsymbol{msgfunc}_{src}(e^t_{u,v}), \quad mem^t_u = \boldsymbol{upd}_{src}(mem^{t-}_u, msg^t_{src}(u, v)); \\ msg^t_{dst}(u, v) &= \boldsymbol{msgfunc}_{dst}(e^t_{u,v}), \quad mem^t_v = \boldsymbol{upd}_{dst}(mem^{t-}_v, msg^t_{dst}(u, v)). \end{aligned} \tag{1}$$

where msg^t stands for the message of the event, mem^{t-}_u for the embedding of node u before time t.

3 Methodology

Now we introduce our **Temporal Neural Common Neighbor** model. TNCN comprises several key modules: the classic Memory Module, the Temporal NCN Module with efficient CN Extractor, and the NCN-based Prediction Head. Special attention will be given to the Temporal CN Extractor, which is designed to efficiently extract temporal neighboring structures and obtain multi-hop CN information. The pipeline is illustrated in Figure 2. A pseudocode is attached in Appendix F.

3.1 Memory Module

Different from the static GNN used in traditional NCN, our model TNCN adopts a **memory-based backbone** from [4] to efficiently store and update the node memory, eliminating the need for repeated computation of node embeddings within successive temporal batches.

Conforming to the standard pipeline in section 2.4 of processing the node memory, for later link prediction or other downstream tasks, node embeddings can be obtained from their memory:

$$emb^t_u = NN(mem^{t-}_u, \underset{v \in N^t_1(u)}{\cup} mem^{t-}_v, \underset{t' < t}{\cup} [e^{t'}_{u,v} \mid\mid \phi(t - t')]), \tag{2}$$

where $||$ stands for the concatenation operation. Here NN has multiple choices, like Identity or simple static GNN [13–15]. In our implementation, we adopt Graph Transformer Convolution [16], which can pay more attention to the relation between different nodes. And we choose the time encoding ϕ presented in Time2Vec [17] as TGN does.

3.2 Temporal NCN Module with Efficient CN Extractor

Our Temporal NCN Module can efficiently perform multi-hop common neighbor extraction with its CN extractor, and aggregate their neural embeddings to attain the node features.

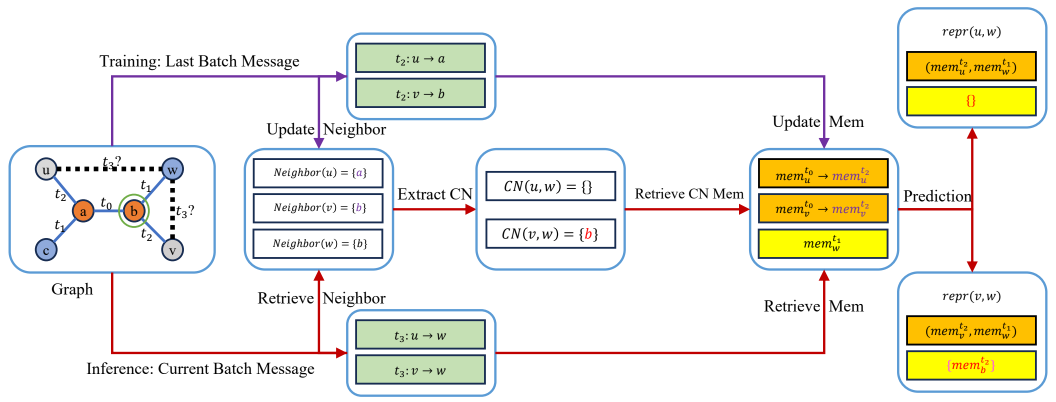


Figure 2: Pipeline of TNCN. TNCN operates through a sequential update and prediction framework that processes successive batches of messages. During the update phase, TNCN updates the neighbor dictionary and the node memory. In the prediction phase, the model retrieves neighbors to identify common neighbors, leveraging the representations of the target nodes and their CNs for prediction.

Extended Common Neighbor. The definition of multi-hop common neighbors (CN) is given in Definition 2.3, extending the traditional (1,1)-hop CN (i.e., nodes on 2-paths between u and v) to arbitrary (i, j)-hop CN. Additionally, we define the zero-hop neighbor of a central node, i.e., u is considered as a neighbor of itself, which will be utilized to calculate CNs with other nodes. Given source node u and target node v, the (0,1)-hop and (1,0)-hop CN not only records the historical interactions between two nodes, but also reveals the frequency of their interactions.

Efficient CN Extractor. The CN Extractor is a crucial component of the TNCN model, contributing significantly to its high performance and scalability. It can efficiently gather pertinent information about a given center node and extract multi-hop common neighbors for a source-destination pair.

For each relevant node u, the extractor stores its historical interactions with other nodes as both source and destination. After a batch of events is processed by the model, the storage is updated with the latest interactions. This allows us to maintain a record of all historical interactions up to a certain timestamp, effectively constructing a dynamic lookup dictionary for fast retrieval during subsequent inference. To strike a balance between memory consumption and model capacity, we save only the most recent K events and relevant nodes for each center node, where K is a hyperparameter determined by the specific dataset.

To implement an efficient batch CN extractor, we organize the historical interactions in a *Sparse Tensor*, representing the temporal adjacency matrix. Then we perform **self-multiplication** to generate high-order adjacency connectivity. Sparse matrix **hadamard product** is finally employed to obtain separate (i, j)-hop CNs. All these operations can be efficiently implemented by sparse tensor operators and are supported by GPU to facilitate fast, batch processing. Now we show the detailed procedure **as follows**. For some special and higher-order cases analysis, please refer to Appendix G.

Details of Common Neighbor Extraction. Our temporal CN extractor begins with a sparse matrix A constructed from the interactions of related nodes. We then include three stages to precisely generate arbitrary (i, j)-hop CNs:

1. Generate up to k-hop neighbors. The original matrix A only includes 1-hop neighbors. To extend this, we: (a) Use self-loops for 0-hop neighbors, denoted as A^0. (b) Perform sparse matrix multiplication (i.e., A^k) to include arbitrary k-hop neighbors. Combining them, we obtain an updated neighborhood matrix set $\hat{A} = \{A^i\}_{i=0}^{K}$.

2. Extract neighbors for each source and destination node with corresponding indices in the same batch. Assume that we require the k-th hop neighbors of node u, then vector $A^k[id(u)]$ is the result, where $id(u)$ stands for the reindexed id for node u. $A^k[id(u)][id(v)] = w > 0$ if v is a k-hop neighbor of u, otherwise this element is 0. w represents the historical interaction frequency.

3. Obtain arbitrary (i, j)-hop CNs. We can perform hadamard product of $A^i[id(u)]$ and $A^j[id(v)]$ to acquire different hops of CNs. The operator can extract corresponding CNs for source-destination node pairs in a batch parallelly.

By re-indexing the node IDs when generating $\hat{A}$ to prevent conflicts, the CN extractor can conduct the sparse matrix calculation, which are all performed in a Torch style that supports **batch operations**, thus enhancing parallelism and efficiency.

The utilization of **Multi-hop Common Neighbors** significantly boosts TNCN's performance, resulting in higher scores in temporal link prediction tasks. Furthermore, by employing sparse tensors, our model achieves substantial reductions in both storage requirements and computational complexity, thereby decreasing time consumption and enhancing efficiency. We also give a comparison between our TNCN and traditional NCN in Table 12 in Appendix D.4.

3.3 NCN-based Prediction Head

We finally construct our NCN-based representation as follows. For source and destination nodes, we perform an element-wise product. For multi-hop CN nodes, we aggregate their embeddings in each hop with sum pooling.

$$X_{u,v}^t = emb_u^t \otimes emb_v^t, \quad NCN_{(i,j)}(u,v) = \bigoplus_{w \in \mathrm{CN}_{(i,j)}^t(u,v)} emb_w^t. \tag{3}$$

These embeddings are then concatenated as the final pair-wise representation:

$$repr(u,v) = [X_{u,v}^t \mid\mid (\overset{K}{\underset{i,j}{\mid\mid}}) NCN_{(i,j)}(u,v)]. \tag{4}$$

We have used $\otimes$, $\oplus$, and $||$ to denote element-wise product, element-wise summation, and concatenation of vectors, respectively. The pair-wise representation $repr(u,v)$ for nodes u and v will be fed to a projection head to output the final link prediction.

4 Efficiency and Effectiveness of TNCN

In this section, we explore the two principal benefits of TNCN: efficiency and effectiveness. These advantages are demonstrated through an analysis of two core components within the framework for temporal graph link prediction: graph representation learning and link prediction methods.

Temporal graph representation learning aims to develop an embedding function, denoted as Emb, which learns an embedding for each node encoding its structural and feature information within the graph. Specifically, given a new event represented as (u,v,t), the function Emb leverages prior events to generate meaningful embeddings. We first categorize graph representation learning approaches into two types: memory-based and k-hop-subgraph-based, according to **their temporal scope of evolved events**.

Definition 4.1. Memory-based approach. Given a new event (u,v,t), if Emb conforms to the following form, the method is referred to as a memory-based approach, which opts to maintain a dynamic, incrementally updated embedding for each node.

$$Emb(u,t) = f_{emb}(Mem(u,t')), \quad Emb(v,t) = f_{emb}(Mem(v,t')), \tag{5}$$

where the Mem can be obtained from the pipeline in Definition 2.4, and f_{emb} is a learnable function.

Definition 4.2. k-hop-subgraph-based approach. Given a new event (u,v,t), if Emb conforms to the following form, the method is defined as a subgraph-based approach, which chooses to recalculate node embeddings by considering the entire historical events.

$$Emb(u,t) = f_{emb}(\mathcal{G}_{u,<t}^k), \quad Emb(v,t) = f_{emb}(\mathcal{G}_{v,<t}^k), \tag{6}$$

where $\mathcal{G}_{u,<t}^k$ is a subgraph induced from $\mathcal{G}$ by node u's k-hop temporal neighborhood $\bigcup_{k=0}^{K} N_k^t(u)$, containing only the edges (events) with time $t' < t$, and f_{emb} is a learnable function.

4.1 Effectiveness

The analysis begins by assessing the effectiveness of the two paradigms. To do so, we first introduce the concept of k-hop event [18].

Definition 4.3. **k-hop event & monotone k-hop event.** A *k-hop event* is a sequence of consecutive edges $\{(u_i, u_{i+1}, t_{u_i,u_{i+1}}) \mid i \in \{0, \dots, k-1\}, k \geq 1\}$ connecting the initial node u_0 to the final node u_k. For example, $\{(u, x, t'), (x, v, t)\}$ is a 2-hop event. In the case where $k = 1$, the k-hop event reduces to a single interaction (u, v, t). A *monotone k-hop event* is a k-hop event in which the sequence of timestamps $\{t_{u_i,u_{i+1}} \mid i \in \{0, \dots, k-1\}, k \geq 1\}$ is strictly monotonically increasing.

Then, we analyze the expressiveness of the two approaches in terms of encoding k-hop event.

Theorem 4.4. *(Ability to encode k-hop events). Given a k-hop event $\{(u_i, u_{i+1}, t_{u_i,u_{i+1}}) \mid i \in \{0, \dots, k-1\}, k \geq 1\}$, if the node embedding of u_0 at time t_{u_0,u_1} can be reversely recovered from the encoding Enc$(\{(u_i, u_{i+1}, t_{u_i,u_{i+1}}) \mid i \in \{0, \dots, k-1\}, k \geq 1\})$, then we say the encoding function Enc is capable of encoding the k-hop event. The following results outline the encoding capabilities of different learning paradigms:*

- *Memory-based approaches can encode any k-hop events with $k = 1$.*
- *Memory-based approaches can encode any monotone k-hop events with arbitrary k.*
- *k-hop-subgraph-based approaches can encode any k'-hop events with $k' \leq k$.*

The proof is in Appendix I. From Theorem 4.4, we can conclude that 1) memory-based approaches have superior expressiveness in encoding k-hop events compared to 1-hop-subgraph-based approaches, and 2) memory-based approaches have superior expressiveness in encoding monotone k-hop events than k'-hop-subgraph-based approaches when $k' < k$.

While the memory-based approach does not consistently rival the expressiveness of the k-hop-subgraph paradigm, it possesses advantages in monotone events and long-history scenarios (where k-hop subgraphs would be unaffordable to extract).

Corollary 4.5. *If we use up to k hop neighborhood information of central node u, then TNCN can capture at least $(k+1)$-hop subgraph information around u.*

This is because TNCN with memory-based backbone can obtain additional 1-hop information regardless of the time monotony, i.e. arbitrary central node can interact with its neighbor when the edge between them exists. This can extend TNCN's capability for free.

We also demonstrate the effectiveness of TNCN in the following theorem 4.6 with the proof in Appendix I. The first part shows that it can capture three important pairwise features commonly used as effective link prediction heuristics, namely Common Neighbors (CN), Resource Allocation (RA), and Adamic-Adar (AA) [19–21]. The experimental results in Appendix D further validate this claim. In the second part we reveal that TNCN is strictly more expressive than some traditional temporal graph networks including Jodie [2], DyRep [3], TGN [4] and TGAT [6] under the same condition, which are widely used as baselines.

Theorem 4.6. *(Expressivity of TNCN)*

1. *TNCN is strictly more expressive than CN, RA, and AA.*
2. *TNCN is strictly more expressive than Jodie with the same dimension of time encoding, DyRep with the same aggregation function, TGAT with the same attention layers and neighbors, and TGN under identical condition for all module choices.*

From this theorem we can find that TNCN extends the previous generic memory-based framework of temporal graph networks with explicitly adopting the neural embeddings of common neighbors. This method can serve as a complementary addition for learning pair-wise representations.

4.2 Efficiency

We then turn our attention to the efficiency of the two approaches. A pivotal factor is the frequency with which individual events are incorporated into computations. In memory-based approaches, each event is utilized **a single time** for learning, immediately following its associated prediction. Conversely, in the k-hop-subgraph-based method, an event may be employed **multiple times**, as it is revisited in different nodes' temporal neighborhood and repeated been processed within each subgraph's encoding (such as message passing) process. This discrepancy leads to divergent cumulative frequencies of event utilization throughout the learning process, resulting in the huge efficiency advantage of memory-based methods. We formalize this observation as:

Theorem 4.7. *(Learning method time complexity). Denote the time complexity of a learning method as a function of the total number of events processed during training. For a given graph $\mathcal{G}$ with the number of nodes designated as $|\mathcal{N}|$ and the number of edges as $|\mathcal{E}|$, the following assertions hold:*

- *For memory-based approaches, the time complexity is $\Theta\left(|\mathcal{E}|\right)$.*
- *For k-hop-subgraph-based approaches with $k = 1$, the lower-bound time complexity is $\Omega\left(\frac{|\mathcal{E}|^2}{|\mathcal{N}|}\right)$, and the upper-bound time complexity is $\mathcal{O}\left(\frac{|\mathcal{E}|^2}{|\mathcal{N}|} + |\mathcal{E}||\mathcal{N}|\right)$.*
- *For k-hop-subgraph-based approaches with $k = 2$, the upper-bound time complexity is $\mathcal{O}\left(\left(\frac{|\mathcal{E}|^2}{|\mathcal{N}|} + |\mathcal{E}||\mathcal{N}|\right)^{\frac{3}{2}}\right)$.*

The proof is attached in Appendix I with part of the proof based on a classic conclusion from de Caen [22] in the graph theory. Following Theorem 4.7, it becomes evident that the computational overhead incurred by a memory-based method is significantly lower than that of a subgraph-based method, particularly as k increases. These results highlight the advantages of memory-based methods in mitigating the computational efficiency challenges associated with large-scale temporal graphs.

So based on the memory backbone, our TNCN furthur utilizes an efficient CN extractor, eliminating the necessity for message passing on entire graphs. Consequently it achieves a unified optimization objective: **to avoid message passing on entire subgraphs in favor of non-repetitive operations**.

To summarize, TNCN introduces the extended common neighbor approach with the efficient CN extractor. This method serves as a complementarity for learning pair-wise representations, while culminating in a cohesive solution that is both efficient and effective.

5 Related Work

Memory-based Temporal Graph Representation Learning. Memory-based models learn node memory using continuous events with non-decreasing timestamps. Researchers have proposed several memory-based methods including JODIE [2], DyRep [3] and TGN [4]. These methods are superior in higher efficiency, while lacking in capturing structural information.

Graph-based Temporal Graph Representation Learning. Subsequent works have incorporated the neighborhood structure into temporal graph learning. CNE-N [23] employs a hash table to map interactions and computes co-neighbor encodings to predict future links. Unlike our TNCN which directly utilizes CN embeddings in prediction, CNE-N simply counts neighbor numbers. Additionally, while CNE-N manages recent interactions through hash tables, TNCN adopts a monotonic storage scheme. NAT [10] similarly builds a multi-hop node dictionary to compress neighbors, but it may suffer from hash collisions. DyGFormer [11] encodes one-hop neighbors and their co-occurrences, then uses a Transformer for predictions, requiring repeated neighborhood sampling and computations. It only models 1-hop neighbor frequency and does not use CN embeddings. In contrast, TNCN supports multi-hop neighbors and includes their embeddings, enabling richer representations.

We also include detailed introduction to more related works in Appendix B.

6 Experiments

This section assesses TNCN's effectiveness and efficiency by answering the following questions:

Q1: What is the performance of TNCN compared with state-of-the-art baselines?
Q2: What is the computational efficiency of TNCN in terms of time consumption?
Q3: Do the extended common neighbors bring benefits to original common neighbors?

6.1 Experimental Settings

Datasets. We evaluate our model on five large-scale real-world datasets for temporal link prediction from the ***Temporal Graph Benchmark*** [24]. These datasets span several distinct fields: co-editing network on Wikipedia, Amazon product review network, cryptocurrency transactions, directed reply network of Reddit, and crowdsourced international flight network. They vary in scales and time spans.

Table 1: Test Performance of different models under MRR metric. The top three are emphasized by red, blue and **bold** fonts. '-' denotes scenarios where a specific method was either not applied to the dataset or was unable to complete the validation and testing phases within a reasonable timeframe.

Model	Wiki	Review	Coin	Comment	Flight
JODIE	0.631 ± 1.69	**0.414 ± 0.15**	-	-	-
DyRep	0.519 ± 1.95	0.401 ± 0.59	0.452 ± 4.60	0.289 ± 3.30	0.556 ± 1.40
TGAT	0.599 ± 1.63	0.196 ± 0.23	0.609 ± 0.57	0.562 ± 2.11	-
TGN-official	0.528 ± 0.06	0.387 ± 0.02	0.737 ± 0.03	0.622 ± 0.02	0.705 ± 0.02
TGN-ns	0.689 ± 0.53	0.375 ± 0.23	0.586 ± 3.70	0.379 ± 2.10	-
CAWN	0.730 ± 0.60	0.193 ± 0.10	-	-	-
EdgeBank(tw)	0.633	0.029	0.574	0.149	0.387
EdgeBank(un)	0.525	0.023	0.359	0.129	0.167
TCL	0.781 ± 0.20	0.165 ± 1.85	0.687 ± 0.30	0.701 ± 0.83	-
GraphMixer	0.598 ± 0.39	0.369 ± 1.50	0.756 ± 0.27	**0.762 ± 0.17**	-
NAT	0.749 ± 1.00	0.341 ± 2.00	-	-	-
DyGFormer	0.798 ± 0.42	0.224 ± 1.52	0.752 ± 0.38	0.670 ± 0.14	-
CNE-N	**0.802 ± 0.20**	0.261 ± 0.25	0.772 ± 0.21	0.790 ± 0.14	-
TPNet	0.827 ± 0.01	-	0.832 ± 0.01	0.825 ± 0.06	0.884 ± 0.01
TNCN-official	0.724 ± 0.01	0.419 ± 0.09	0.770 ± 0.06	0.727 ± 0.12	**0.817 ± 0.04**
TNCN-ns	0.803 ± 0.01	0.427 ± 0.06	**0.771 ± 0.04**	0.705 ± 0.12	0.831 ± 0.03

Additional details about the datasets are provided in Appendix A. We set the evaluation metric as **Mean Reciprocal Rank (MRR)** consistent with the TGB official leaderboard.

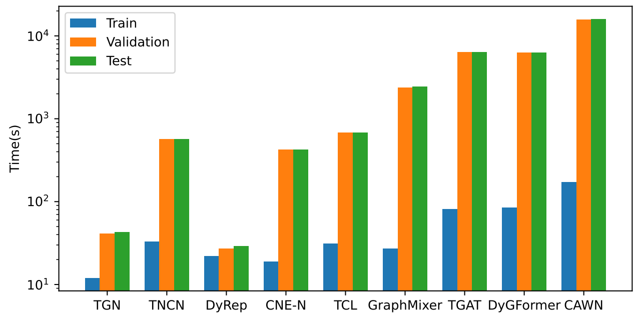
Baselines. We systematically evaluate our proposed model TNCN against a diverse set of baselines: a heuristic algorithm Edgebank [11], memory-based models Jodie [2], DyRep [3] and TGN [4] that obviate the need for frequent temporal subgraph sampling, and GraphMixer [25] which employs an MLP-mixer. We also include various graph-based models such as CAWN [9], TGAT [6], TCL [5], NAT [10], DyGFormer [11], CNE-N [23] and TPNet [26], which learn from neighborhood structure information.

Here we evaluate our TNCN under two similar but different settings, the official setting ("official") and the new setting ("ns"). "*-official" strictly complies to the **official setting** of TGB evaluation policy, using both *streaming setting* and *lag-one scheme* for both memory update and neighborhood awareness. Streaming setting means the information of the validation and test sets can only be employed for updating the memory without any back propagation. Lag-one scheme implies that the model can access only the information from **before the current batch** for predictions; in other words, the latest usable batch is the previous one. This applies to not only the memory, but also the **neighborhood awareness**. "*-ns" obeys the streaming setting but considers the interactions within the same batch before the current prediction time. This allows the model to utilize more recent neighborhood information, potentially giving it unfair advantages in datasets where recent interactions are crucial. Methods "*-official" use the former setting while others report the latter. For a fair evaluation and comparison, here we display the performance of our TNCN under both settings.

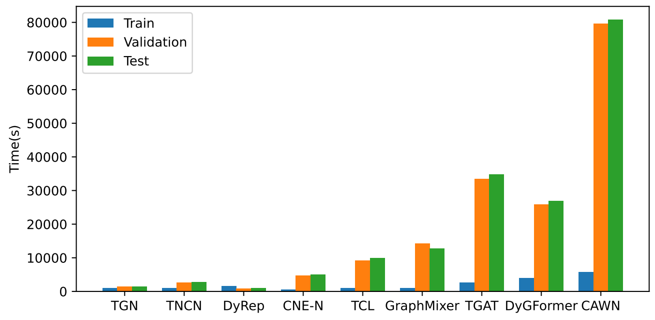
6.2 Experimental Results

Reply to Q1: TNCN possesses remarkable performance. We conducted comprehensive evaluations of prevailing methods on TGB. The main results are summarized in Table 1. It is evident from the table that TNCN attains **new SOTA performance on Review dataset**. Additionally, TNCN demonstrates competitive results on the remaining datasets, ranking 2nd or/and 3rd on Wiki, Coin and Flight. TNCN almost consistently surpasses both memory-based models such as TGN and DyRep, and graph-based methods like DyGFormer and NAT. The dataset where TNCN still exhibits a large gap from the top-three baselines is Comment. This may be ascribed to the high "surprise index" of this non-bipartite dataset, wherein prior events have a diminished correlation with subsequent events, potentially reducing the impact of CNs. (More details about *Surprise index* are in Appendix A.)

To obtain a more comprehensive evaluation, we have also conducted experiments on the previous small and medium datasets under both transductive and inductive settings. In these experiments, TNCN achieves **6 new SOTA** out of 7 datasets **in transductive setting** and **3 in inductive setting**,



(a) tgbl-wiki (log scale)



(b) tgbl-review (linear scale)

Figure 3: Time Consumption of Memory and Graph-based Method on Wiki and Review Datasets.

further facilitating its strong performance. The overall performance and some additional results (such as TGN with heuristics, *etc.*) can be referred to Appendix D.

Reply to Q2: TNCN shows great scalability on large datasets. To evaluate computational efficiency, we collected the time consumption on Wiki and Review datasets, as depicted in Figure 3. Compared with memory-based methods, TNCN exhibits a comparable order of magnitude in terms of time consumption. However, when benchmarked against graph-based models, TNCN demonstrates a substantial acceleration, achieving approximately 2.5 to 5.9 times speedup during the training phase and a 1.8 to 30.3 times increase in inference speed. In the Table 13 we provide a full comparison of the time consumption for different models over the five TGB datasets. Notably, the scalability concerns become even more evident as the size of the dataset expands; several graph-based models cannot complete the validation and testing processes within a reasonable time budget. The primary factors contributing to TNCN's efficiency are the synergistic, time-efficient design of its two core components and the implementation of the Efficient CN Extractor that facilitates batch operations through parallel processing. For detailed statistics about TNCN and NAT, please refer to Appendix E.1.

6.3 Ablation Study

Reply to Q3: Extended CN brings improvements. To elucidate the benefits of extended CN, we conducted an ablation study under **official setting** on the hop range of common neighbors. The results are shown in Table 2. Here we use notation "k-hop CN" to simply denote the CNs up to (k, k)-hop. The conventional NCN method considers only (1,1)-hop CN. However, this approach may not be universally applicable across all temporal networks. For instance, *bipartite graphs* lack such (1,1)-hop CN in their structure, necessitating the consideration of 2-hop CN. Additionally, memory-based methods may omit a notable aspect: they generally find it difficult to quantify the frequency of interactions between a given pair of nodes, which brings the need for 0-hop neighborhood.

Table 2: Test performance of TGN and TNCN with different ranges of common neighbors.

Model	Wiki	Review	Coin	Comment
TGN	0.528	0.387	0.737	0.622
TNCN-1-hop-CN	0.621	**0.419**	0.737	0.641
TNCN-0$\sim$1-hop-CN	0.720	0.298	0.739	**0.727**
TNCN-0$\sim$2-hop-CN	**0.724**	0.317	**0.770**	0.662

To address these limitations, we have expanded the original (1,1)-hop CN to 0$\sim$k-hop CN. The results indicate that TNCN utilizing 0$\sim$1-hop CN markedly surpasses the (1,1)-hop CN on various datasets. This enhancement underscores the significance of the introduced 0-hop neighbors to our architecture. Nevertheless, the inclusion of 2-hop CN yields mixed results across datasets. We also show the result of TNCN without temporal NCN module (which is TGN), revealing the effectiveness of this module.

We have also conducted experiments for parameter analysis. Please refer to Appendix D.3 for details.

7 Conclusion and Limitation

We propose TNCN for temporal graph link prediction, which employs a temporal common neighbor extractor combined with a memory-based node representation learning module. TNCN has achieved new state-of-the-art results on several real-world datasets while maintaining excellent scalability to handle large-scale temporal graphs.

However, based on our observation of TNCN's performance on the Comment dataset, there are some limitations in our model. Specifically, non-bipartite datasets with high surprise values, such as the Comment dataset, tend to make it more challenging for TNCN to accurately predict the probability of future connections. This indicates that while TNCN performs well overall, it may struggle with datasets that exhibit high variability or unexpected patterns. Further research is needed to address these challenges and improve the model's robustness in such scenarios.

Acknowledgement

This work is supported by the National Key R&D Program of China (2022ZD0160300) and National Natural Science Foundation of China (62276003).

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. 2

[2] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019. 2, 3, 6, 7, 8, 14

[3] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*, 2019. 6, 7, 8, 14

[4] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020. 2, 3, 6, 7, 8, 14, 15

[5] Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. Tcl: Transformer-based dynamic graph modelling via contrastive learning. *arXiv preprint arXiv:2105.07944*, 2021. 2, 8, 14

[6] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020. 2, 6, 8, 15

[7] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018. 2, 15

[8] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021. 2, 15

[9] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. *arXiv preprint arXiv:2101.05974*, 2021. 2, 8, 14

[10] Yuhong Luo and Pan Li. Neighborhood-aware scalable temporal network representation learning. In *Learning on Graphs Conference*, pages 1–1. PMLR, 2022. 7, 8, 15

[11] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. *Advances in Neural Information Processing Systems*, 36: 67686–67700, 2023. 2, 7, 8, 15

[12] Xiyuan Wang, Haotong Yang, and Muhan Zhang. Neural common neighbor with completion for link prediction. *arXiv preprint arXiv:2302.00890*, 2023. 2, 15

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016. 3

[14] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 3

[16] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020. 3

[17] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019. 3

[18] L. LOV´ASZ, Peter Winkler, Andr´as Luk´acs, and Andrew Kotlov. Random walks on graphs: A survey. 1993. URL `https://api.semanticscholar.org/CorpusID:10655982`. 5

[19] Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001. 6, 15

[20] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3): 211–230, 2003. 16

[21] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71:623–630, 2009. 6, 16

[22] D. de Caen. An upper bound on the sum of squares of degrees in a graph. *Discrete Mathematics*, 185(1):245–248, 1998. ISSN 0012-365X. doi: https://doi.org/10.1016/S0012-365X(97)00213-6. URL `https://www.sciencedirect.com/science/article/pii/S0012365X97002136`. 7, 23

[23] Ke Cheng, Peng Linzhi, Junchen Ye, Leilei Sun, and Bowen Du. Co-neighbor encoding schema: A light-cost structure encoding method for dynamic link prediction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 421–432, 2024. 7, 8, 15

[24] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems*, 2023. 7

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

[26] Xiaodong Lu, Leilei Sun, Tongyu Zhu, and Weifeng Lv. Improving temporal link prediction via temporal walk matrix projection, 2024. URL `https://arxiv.org/abs/2410.04013`. 8, 15

[27] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction. *Advances in Neural Information Processing Systems*, 35:32928–32941, 2022. 14

[28] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, and Zhenyu Guo. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD/PODS '21. ACM, June 2021. doi: 10.1145/3448016.3457564. URL http://dx.doi.org/10.1145/3448016.3457564. 14

[29] Xinshi Chen, Yan Zhu, Haowen Xu, Mengyang Liu, Liang Xiong, Muhan Zhang, and Le Song. Efficient dynamic graph representation learning at scale. *arXiv preprint arXiv:2112.07768*, 2021. 14

[30] Yizhou Chen, Anxiang Zeng, Guangda Huzhang, Qingtao Yu, Kerui Zhang, Cao Yuanpeng, Kangle Wu, Han Yu, and Zhiming Zhou. Recurrent temporal revision graph networks, 2023. 14

[31] Hongkuan Zhou, Da Zheng, Xiang Song, George Karypis, and Viktor Prasanna. Disttgl: Distributed memory-based temporal graph neural network training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2023. 14

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 15

[33] Yuxing Tian, Yiyan Qi, and Fan Guo. Freedyg: Frequency enhanced continuous-time dynamic graph model for link prediction. In *The Twelfth International Conference on Learning Representations*. 15, 16

[34] Harry Shomer, Yao Ma, Haitao Mao, Juanhui Li, Bo Wu, and Jiliang Tang. Lpformer: An adaptive graph transformer for link prediction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, volume 35 of *KDD '24*, page 2686–2698. ACM, August 2024. doi: 10.1145/3637528.3672025. URL http://dx.doi.org/10.1145/3637528.3672025. 15

[35] Tao Zou, Yuhao Mao, Junchen Ye, and Bowen Du. Repeat-aware neighbor sampling for dynamic graph learning, 2024. URL https://arxiv.org/abs/2405.17473. 15

[36] Dongyuan Li, Shiyin Tan, Ying Zhang, Ming Jin, Shirui Pan, Manabu Okumura, and Renhe Jiang. Dyg-mamba: Continuous state space modeling on dynamic graphs, 2024. URL https://arxiv.org/abs/2408.06966. 15

[37] Xuanwen Huang, Wei Chow, Yize Zhu, Yang Wang, Ziwei Chai, Chunping Wang, Lei Chen, and Yang Yang. Enhancing cross-domain link prediction via evolution process modeling. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 2158–2171, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712746. doi: 10.1145/3696410.3714792. URL https://doi.org/10.1145/3696410.3714792. 15

[38] Haoyang Li, Yuming Xu, Yiming Li, Hanmo Liu, Darian Li, Chen Jason Zhang, Lei Chen, and Qing Li. When speed meets accuracy: an efficient and effective graph model for temporal link prediction, 2025. URL https://arxiv.org/abs/2507.13825. 15

[39] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953. 15

[40] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd, et al. The pagerank citation ranking: Bringing order to the web. 1999. 15, 16

[41] Shuming Liang, Yu Ding, Zhidong Li, Bin Liang, Yang Wang, Fang Chen, et al. Can gnns learn heuristic information for link prediction? 2022. 15

[42] Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J Kim. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. *Advances in Neural Information Processing Systems*, 34:13683–13694, 2021. 15

[43] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Yannick Hammerla, Michael M Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. In *The eleventh international conference on learning representations*, 2022. 15, 16

[44] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34:29476–29490, 2021. 15

[45] Haitao Mao, Juanhui Li, Harry Shomer, Bingheng Li, Wenqi Fan, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Revisiting link prediction: A data perspective. *arXiv preprint arXiv:2310.00793*, 2023. 15

[46] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. *Advances in Neural Information Processing Systems*, 36, 2024. 15

[47] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi: 10.1126/science.286.5439.509. URL `https://www.science.org/doi/abs/10.1126/science.286.5439.509`. 16

[48] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279, 2003. 16

[49] SN Perepechko and AN Voropaev. The number of fixed length cycles in an undirected graph. explicit formulae in case of small lengths. *Mathematical Modeling and Computational Physics (MMCP2009)*, 148, 2009. 19

A Datasets

Table 3 shows some detailed datasets statistics of TGB and 4 shows several temporal graph datasets commonly used by previous work. Through the two tables we can observe that TGB official datasets possess temporal graphs with larger scale to 10 million, 10 times surpassing the largest previous datasets such as LastFM. With the aim to examine our TNCN model's efficiency, we choose the increasingly accepted datasets TGB in the main table.

Table 3: TGB Dataset Statistics.

Dataset	Domain	Nodes	Edges	Steps	Surprise	Edge Properties
tgbl-wiki	interact	9,227	157,474	152,757	0.108	W: ×, Di: ✓, A: ✓
tgbl-review	rating	352,637	4,873,540	6,865	0.987	W: ✓, Di: ✓, A: ×
tgbl-coin	transact	638,486	22,809,486	1,295,720	0.120	W: ✓, Di: ✓, A: ×
tgbl-comment	social	994,790	44,314,507	30,998,030	0.823	W: ✓, Di: ✓, A: ✓
tgbl-flight	traffic	18143	67,169,570	1,385	0.024	W: ×, Di: ✓, A: ✓

Here "Surprise index" [27] refers to the ratio of test edges that are not seen during training, which can be calculated as $\frac{|E_{test}/E_{train}|}{|E_{test}|}$. Low surprise index implies that memory-based methods such as Edgebank [27] may potentially achieve good performance, while high surprise may require more inductive capability. The surprise index varies across TGB datasets.

Table 4: Previous Dataset Statistics.

Datasets	Domains	Nodes	Links	N&L Feat	Bipartite	Duration	Unique Steps	Time Granularity
Wikipedia	Social	9,227	157,474	– & 172	✓	1 month	152,757	Unix timestamps
Reddit	Social	10,984	672,447	– & 172	✓	1 month	669,065	Unix timestamps
MOOC	Interaction	7,144	411,749	– & 4	✓	17 months	345,600	Unix timestamps
LastFM	Interaction	1,980	1,293,103	– & –	✓	1 month	1,283,614	Unix timestamps
Enron	Social	184	125,235	– & –	×	3 years	22,632	Unix timestamps
UCI	Social	1,899	59,835	– & –	×	196 days	58,911	Unix timestamps

B Related Work

B.1 Memory-based Temporal Graph Representation Learning

Temporal graph learning has garnered significant attention in recent years. A classic approach in this domain involves learning node memory using continuous events with non-decreasing timestamps. Kumar et al. [2] propose a coupled recurrent neural network model named JODIE that learns the embedding trajectories of users and items. Another contemporary work DyRep [3] aims to efficiently produce low-dimensional node embeddings to capture the communication and association in dynamic graphs. Rossi et al. [4] introduces a memory-based temporal neural network known as TGN, which incorporates a memory module to store temporal node representations updated with messages generated from the given event stream. Apan [28] advances the methodology by integrating asynchronous propagation techniques, markedly increasing the efficiency of handling large-scale graph queries. EDGE [29] emerges as a computational framework focusing on increasing the parallelizability by dividing some intermediate nodes in long streams each into two independent nodes while adding back their dependency by training loss. Chen et al. [30] extend the update method for the node memory module, introducing an additional hidden state to record previous changes in neighbors. Complementing these efforts, additional contributions such as Edgebank [27] and DistTGL [31] have been directed towards formalizing and accelerating memory-based temporal graph learning methods.

B.2 Graph-based Temporal Graph Representation Learning

Subsequent works have incorporated the temporal neighborhood structure into temporal graph learning. CAWN [9] employs random anonymous walks to model the neighborhood structure. TCL [5] samples a temporal dependency interaction graph that contains a sequence of temporally cascaded chronological interactions. TGAT [6] considers the temporal neighborhood and feeds the features into a temporal graph attention layer utilizing a masked self-attention mechanism. NAT [10] constructs a multi-hop neighboring node dictionary to extract joint neighborhood features and uses RNN to recursively update the central node's embedding. DyGFormer [11], instead, leverages one-hop neighbor embeddings and the co-occurrence of neighbors to generate features, which are well-patched and subsequently fed into a Transformer [32] decoder to obtain the final prediction. FreeDyG [33] also utilizes historical interaction frequency akin to DyGFormer, afterwards transforming it with Fast Fourier Transform (FFT) and IFFT through the frequency domain. LPFormer [34] attempts to adaptively learn the pairwise encodings via graph attention module, utilizing relative position, ppr value and neighboring information to obtain the score. CNE-N [23] uses a hash table to map an interaction event to its position. It calculates the co-neighbor encoding for each (neighbor - end node) pair within the local subgraph, recording the number of their common neighbors. These information are then concatenated to predict the probability of the future link. TPNet [26] constructs temporal walk matrices via random propagation to simultaneously consider both temporal and structural information. Pairwise and auxiliary feature are then decoded to make the prediction. Another work RepeatMixer [35] pays more attention to the repeat-aware neighbors. Such neighbor sequences are then leveraged and adaptively aggregated to learn the temporal patterns. DyG-Mamba [36] introduces SSM to dynamic graph learning. It first extracts the first-hop interaction sequence between the given node pairs and encodes them. The time-span encoding between any two continuous timestamps are also computed, serving as control signals for the continuous SSM to obtain the final representation. CrossLink [37] represents the graph evolution by a sequence of temporal events, where each event representation is obtained from a graph encoder. It finally utilizes a decoder-only transformer to model the sequence and predicts the next token, i.e. the link existence. EAGLE [38] aggregates the most recent neighbors of a central node and leverage temporal personalized PageRank value to capture the structural pattern, afterwards using adaptive weights to dynamically merge these features to get the prediction result.

B.3 Link Prediction Methods

Link prediction is a fundamental task in graph analysis, aiming to determine the likelihood of a connection between two nodes. Early investigations posited that nodes with greater similarity tend to be connected, which led to a series of heuristic algorithms such as Common Neighbors, Katz Index, and PageRank [19, 39, 40]. With the advent of GNNs, numerous methods have attempted to utilize vanilla GNNs for enhancing link prediction, revealing sub-optimal performance due to the inability to capture important pair-wise patterns such as common neighbors [7, 8, 41]. Subsequent research has focused on infusing various forms of inductive biases to retrieve intricate pair-wise relationships. For instance, SEAL [7], Neo-GNN [42], and NCN [12] have integrated neighbor-overlapping information into their design. BUDDY [43] and NBFNet [44] have concentrated on extracting higher-order structural information. Additionally, Mao et al. [45], Li et al. [46] have contributed to a more unified framework encompassing different heuristics.

C TNCN Model Configuration

Network Choice. In our experiment, the changeable neural networks are chosen as follows:

In Memory Module, we choose $Identity$ as *msgfunc* and GRU as upd. In inference stage we process node memory with Graph Attention Embedding to get the temporal representation. As for Prediction Head, we finally choose MLP as the $repr$ function.

Hyper-parameter. Several detailed hyper-parameters for TNCN are shown in Table 5, which can help researchers to reproduce the experiment performance as reported in this paper.

D Additional Experimental Results

D.1 Transductive and Inductive Experiments on Previously Small and Medium Datasets

In addition to the large-scale TGB dataset, we also conduct experiments on some traditional small and medium datasets previously used in dynamic graph link prediction. We follow TGN [4] to evaluate different models under both transductive and inductive settings. The transductive setting deal with the

Table 5: Some Experiment Hyper-parameters.

Dataset	num_neighbors	num_epoch	patience	mem_dim	emb_dim	$time\_dim$
Wiki	15	20	5	184	184	100
Review	15	10	3	184	184	100
Coin	10	5	3	100	100	100
Comment	10	3	2	100	100	100

future links between previously observed nodes in the training stage, and the inductive setting predicts link existence between unseen nodes. We compare TNCN with the aforementioned baselines with the addition of FreeDyG [33], which is also a competitive method in temporal graph link prediction. The overall performance of TNCN and different models are in Table 6.

From Table 6 we can find that TNCN achieves **6 new SOTA** out of 7 datasets **in transductive setting** and **3 in inductive setting**, furthur uncovering the strong performance of our model.

Table 6: Average Precision (AP) under Transductive and Inductive settings on small and medium dataset. The best is in **bold** font, and the second is underlined. ("Trans" and "Ind" are the abbreviation for transductive and inductive respectively.)

Setting	Method	Wikipedia	Reddit	Mooc	Lastfm	Enron	Social Evo.	UCI	Avg. Rank
Trans	CAWN	98.62±0.05	98.66±0.09	80.15±0.25	86.99±0.06	89.56±0.09	84.96±0.09	95.18±0.06	8.71
	JODIE	96.15±0.36	97.20±0.05	80.23±2.44	70.85±2.13	84.77±0.30	89.89±0.55	89.43±1.09	11.57
	DyRep	95.81±0.15	98.00±0.19	81.97±0.49	71.92±2.21	82.38±3.36	88.87±0.30	65.14±2.30	11.86
	TGAT	96.94±0.06	98.52±0.02	85.84±0.15	73.42±0.21	71.12±0.97	93.16±0.17	79.63±0.70	10.43
	NAT	98.68±0.04	99.10±0.09	86.54±0.02	88.56±0.02	92.42±0.09	94.43±1.67	94.37±0.21	6.29
	TCL	96.47±0.16	97.53±0.02	82.38±0.24	67.27±2.16	79.70±0.71	93.13±0.16	89.57±1.63	11.29
	DyGFormer	99.03±0.02	99.22±0.01	87.52±0.49	93.00±0.12	92.47±0.12	94.73±0.01	95.79±0.17	4.64
	FreeDyG	99.26±0.01	99.48±0.01	89.61±0.19	92.15±0.16	92.51±0.05	94.91±0.01	96.28±0.11	3.14
	TPNet	**99.32±0.03**	99.27±0.01	96.39±0.09	94.50±0.08	92.90±0.17	94.73±0.02	97.35±0.04	2.21
	EdgeBank	90.37±0.00	94.86±0.00	57.97±0.00	79.29±0.00	83.53±0.00	74.95±0.00	76.20±0.00	12.43
	GraphMixer	97.25±0.03	97.31±0.01	82.78±0.15	75.61±0.24	82.25±0.16	93.37±0.07	93.25±0.57	9.71
	CNE-N	99.09±0.04	99.22±0.01	94.18±0.07	93.55±0.12	92.48±0.10	94.60±0.03	96.85±0.08	3.64
	TGN	98.57±0.05	98.70±0.03	89.15±1.60	77.07±3.97	86.53±1.11	93.57±0.17	92.34±1.04	7.57
	TNCN	99.03±0.02	**99.79±0.02**	**96.69±0.04**	**98.65±0.06**	**97.08±0.14**	**99.95±0.04**	**97.44±0.08**	**1.50**
Ind	CAWN	98.24±0.03	98.19±0.03	81.42±0.24	89.42±0.07	86.35±0.51	79.94±0.18	92.73±0.06	7.71
	JODIE	94.82±0.20	96.50±0.13	79.63±1.92	81.61±3.82	80.72±1.39	91.96±0.48	79.86±1.48	10.00
	DyRep	92.43±0.37	96.09±0.11	81.07±0.44	83.02±1.48	74.55±3.95	90.04±0.47	57.48±1.87	11.29
	TGAT	96.22±0.07	97.09±0.04	85.50±0.19	78.63±0.31	67.05±1.51	91.41±0.16	79.54±0.48	10.50
	NAT	98.55±0.09	98.56±0.21	78.16±0.01	85.91±0.02	**94.94±1.15**	95.16±0.66	92.58±1.86	5.71
	TCL	96.22±0.17	94.09±0.07	80.60±0.22	73.53±1.66	76.14±0.79	91.55±0.09	87.36±2.03	10.93
	DyGFormer	98.59±0.03	98.84±0.02	86.96±0.43	94.23±0.09	89.76±0.34	93.14±0.04	94.54±0.12	4.71
	FreeDyG	**98.97±0.01**	98.91±0.01	87.75±0.62	94.89±0.01	89.69±0.17	94.76±0.05	94.85±0.10	3.14
	TPNet	98.91±0.01	98.86±0.01	**95.07±0.26**	95.36±0.11	90.34±0.28	93.24±0.07	**95.74±0.05**	**2.43**
	EdgeBank	/	/	/	/	/	/	/	/
	GraphMixer	96.65±0.02	95.26±0.02	81.41±0.21	82.11±0.42	75.88±0.48	91.86±0.06	91.19±0.42	9.43
	CNE-N	98.37±0.03	98.78±0.01	91.89±0.31	94.64±0.12	89.66±0.22	93.29±0.37	95.03±0.16	4.00
	TGN	97.83±0.04	97.50±0.07	89.04±1.17	81.45±4.29	77.94±1.02	90.77±0.86	88.12±2.05	8.57
	TNCN	98.31±0.05	**99.07±0.02**	91.56±0.23	**95.74±0.50**	92.04±0.22	**95.51±0.07**	94.57±0.17	2.57

D.2 Comparison with Some Classic Heuristic Methods

We exhibits the result between TGN with some classic heuristics and TNCN under official setting on tgbl-wiki dataset in the Table 7. Here heuristics consist of CN [47], RA [21], AA [20], PPR [40, 48] and ELPH [43]. In these heuristic methods, the heuristic statistics are concatenated with TGN embedding to obtain final predictions. From the table we can see that these basic heuristics such as CN and RA do not bring performance improvement. However, some sophisticated heuristics like graph sketching in ELPH can enhance the backbone's capability. Nevertheless, using these heuristics cannot outperform a more generalized model like our TNCN, which comprehensively takes neighborhood nodes' representations into account.

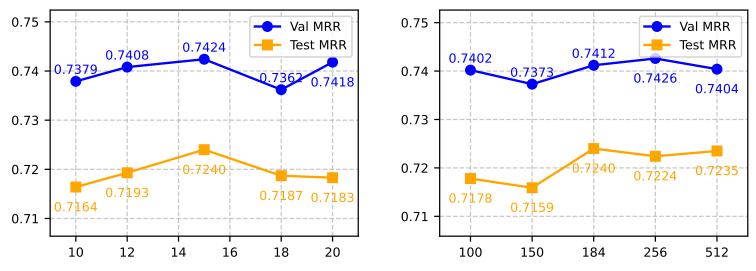
D.3 Detailed Statistics of Parameter Analysis

Here we provide some results on parameter analysis of our TNCN model. Figure 4 (a) illustrates the model performance under different **numbers of neighbors**. Incorporating more recent neighbors can retain richer information, while it may bring about some irrelevant noise. We can find that the test

Table 7: Comparison between TGN with heuristics and TNCN on tgbl-wiki Dataset.

Model	Val MRR	Test MRR	Training Time (s)	Inference Time (s)
TGN	0.569	0.528	10.33	98.74
TGN-CN	0.561	0.505	12.33	106.21
TGN-RA	0.563	0.511	16.51	115.04
TGN-AA	0.565	0.517	11.42	115.01
TGN-PPR	0.521	0.427	207.01	327.22
TGN-ELPH	0.715	0.681	240.92	1614.86
TNCN	0.742	0.724	21.45	250.49

mrr increases as num_neighbors rises from a small value, but it declines after a certain threshold. In Figure 4 (b), we examine how the **embedding and memory dimension** affect the final performance. A similar trend is observed for this parameter. On the other hand, both the training and test time consistently increase with the parameter values becoming larger. This phenomenon is revealed in the Tables 8 to 11. In Table 8 and 9, we show the detailed MRR and time consumption of our TNCN with different numbers of neighbors over Wiki and Coin dataset. In Table 10 and 11 we show how the embedding and memory dimension influence the final performance. To strike a balance between the model capability and time consumption, we finally select the intermediate and appropriate values.



(a) num_neighbors (b) emb_dim & mem_dim

Figure 4: Parameter Analysis on Wiki Dataset.

Table 8: Performance metrics for different numbers of neighbors on Wiki dataset.

num_neighbors	10	12	15	18	20
Val MRR	0.7379	0.7408	0.7412	0.7362	0.7418
Test MRR	0.7164	0.7193	0.7240	0.7187	0.7183
training time (s)	26.35	27.17	29.49	30.32	31.56
test time (s)	378.88	396.75	407.43	413.83	420.22

D.4 Comparison between TNCN and traditional NCN

Here we provide a detailed comparison between our TNCN model and the traditional NCN method in Table 12.

E Detailed Time and Memory Consumption Statistics

E.1 Time Consumption

Table 13 exhibits the detailed time consumption on TGB datasets with different models. We can observe that TNCN maintains similar time consumption to memory-based networks while achieving striking speedup compared with graph-based models.

Table 9: Performance metrics for different numbers of neighbors on Coin dataset.

num_neighbors	5	8	10	12	15
Val MRR	0.7492	0.7378	0.7430	0.7450	0.7406
Test MRR	0.7687	0.7619	0.7701	0.7662	0.7601
training time (s)	5936.19	6129.33	6406.00	6911.00	7529.01
test time (s)	56605.45	57117.99	57292.00	57745.00	57925.00

Table 10: Performance metrics for different embedding and memory dimensions on Wiki dataset.

emb_dim&mem_dim	100	150	184	256	512
Val MRR	0.7402	0.7373	0.7412	0.7426	0.7404
Test MRR	0.7178	0.7159	0.7240	0.7224	0.7235
training time (s)	22.34	25.47	29.49	32.81	34.95
test time (s)	378.28	399.77	407.43	420.45	459.86

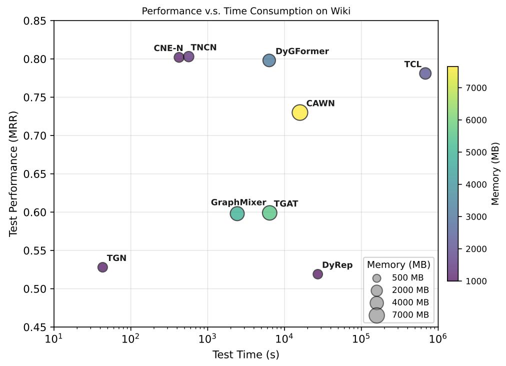
Comparison between TNCN and NAT Here we also show some experimental results in Table 14 for the comparison between TNCN and NAT model. The hardware we use is NVIDIA GeForce RTX 2080 as NAT's code isn't compatible with higher version. Note that NAT model exposes a backward as its instability, accomplishing about only 1/3 experiments when we test it.

E.2 Memory Consumption

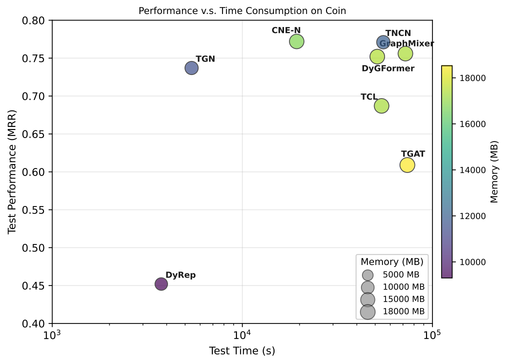
In Table 15 we have shown the memory cost of different methods on TGB datasets. On most of them (except Review) we can observe that TNCN takes similar GPU memory consumption to memory-based networks while occupying significant less memory than graph-based models.

E.3 Overall Comparison between Performance and Time/Memory Consumption

Here we provide a scatter plot to compare different methods between the final performance and the time/memory consumption. Figure 5 (a) shows the result on Wiki, and Figure 5 (b) on Coin. From these scatter plots, we can obtain a more intuitive perception that TNCN can better achieve a balance between the final performance and time/memory consumption.



(a) Wiki



(b) Coin

Figure 5: Comparison between Performance and Time/Memory Consumption.

F Pseudocode of TNCN Pipeline and CN Extraction

Algorithm 1 shows the pseudocode about the pipeline of our TNCN model, and Algorithm 2 for the procedure of the CN extraction operation.

Table 11: Performance metrics for different embedding and memory dimensions on Coin dataset.

emb_dim&mem_dim	30	50	100	150	184
Val MRR	0.7387	0.7405	0.7430	0.7436	0.7518
Test MRR	0.7591	0.7606	0.7701	0.7646	0.7699
training time (s)	6228	6340	6406	6617	6721
test time (s)	56694	57179	57292	58103	59411

Table 12: The comparison of TNCN and NCN.

	temporal scenario	backbone	arbitrary CN hops	batch-wise CN extraction
NCN	✗	traditional GNN	✗	✗
TNCN	✓	memory-based	✓	✓

G Special Cases Analysis of Common Neighbor Extraction

Here are some special cases while calculating $(1, 2)$, $(2, 1)$ and $(2, 2)$ hop CNs. Under these situations, utilizing $A^k[id(u)]$ naively in step (2) will lead to walk-based neighbors, i.e. $\exists v, \exists i \neq j, w_i = w_j,\ s.t.\ (u, w_1, w_2, \cdots, w_{k-1}, v)\ exists$. To obtain a clear version of arbitrary path-based (i, j)-hop CNs, we should avoid the repetition of neighbors. We take $(1, 2)$ as an example to analyse the detailed method to eliminate repetition. Cases like $(2, 1)$ and $(2, 2)$ hop can be similarly solved.

Assume that node x is a $(1, 2)$-hop CN of pair (u, v), thus we know $\exists w,\ s.t.\ (u, x)$ and (v, w, x) exist. There are two variants that render x to be a walk-based CN instead of a path-based one that we exactly require.

(a) $x = v$. When $x = v$, the local graph has the topology shown in igure 6 a. This situation should satisfy two conditions: w is a neighbor of v and there are historical interactions between u and v. Denote the frequency between (u, v) before time t as $q^t(u, v) = |\{(u, v, t')|t' < t\} \cup \{(v, u, t')|t' < t\}|$. So the naively computed $CN^t_{(1,2)}(u, v)[id(x)]$ value need to be subtracted by $[\sum_{w_i} q^t(w_i, v)] * q^t(u, v)$, i.e. the total interaction frequency of v before time t multiplied by the frequency between (u, v).

(b) $w = u$. The structure is exhibited in Figure 6 b. Here (u, v) has historical edges and x is a 1-hop neighbor of u. The additive substraction value is $[\sum_x q^t(x, u)^2] * q^t(u, v)$.

(c) Both (1a) and (1b) are satisfied. The ground truth is as Figure 6 c. We just need to add back the overlap value that have been diminished once more.

Note that the procedure above can only deal with CNs of **no more than** $(2, 2)$**-hop** perfectly. For higher-order (i, j)-hop CN extraction, please refer to Perepechko and Voropaev [49] for more details and complicated analysis.

H Case Study

H.1 Two Examples from TGB for Better Understanding TNCN's Effectivity

In Figure 7, we show two case studies from TGB to give a better understanding of the effectivity of our TNCN.

Figure (a) shows a case from tgbl-wiki, which is a bipartite graph. The yellow nodes 0 and 15 are a (u, v) pair. If we use a node-wise method to predict the future link of $(0, 15)$, we can find that node 15 has just 7 neighbors while node 0 has 12. So their properties may be different, thus having less chance to have an interaction. However TNCN can observe that the blue nodes are their $(1, 2)$-hop CNs and purple nodes are the $(2, 1)$-hop, and it will give a high probability over the existence of the future link.

Figure (b) shows another case from tgbl-coin. Here we need to predict the link of $(1, 8)$. Here TNCN can find that these two nodes have multiple variants of common neighbors. Node 3 is their $(1, 2)$-hop

Table 13: Time Consumption of different methods on TGB Datasets. All these experiments are conducted with NVIDIA GeForce RTX 4090.

Model(tr/val/test)(s)	Wiki	Review	Coin	Comment	Flight
TGN	12/41/43	988/1389/1411	2578/5219/5408	6599/9984/10311	2990/19834/20627
DyGFormer	85/6268/6317	3891/25831/26911	12593/50893/51276	OOM	OOM
GraphMixer	27/2385/2425	983/14188/12764	5917/71699/72103	OOM	OOM
CNE-N	19/422/424	540/4749/4931	2308/18226/19317	OOM	OOM
DyRep	22/27/29	1514/877/916	6305/3821/3746	13701/6419/6587	9352/12188/13112
TGAT	81/6359/6407	2564/33437/34814	25514/73372/73821	OOM	OOM
CAWN	173/15690/15881	5692/79566/80717	OOM	OOM	OOM
TCL	31/680/682	948/9093/9962	4383/52973/54016	OOM	OOM
TNCN	33/565/566	971/2626/2701	5926/54765/55177	9828/59439/60163	8209/50533/51064

Table 14: Comparison of Time Consumption between TNCN and NAT.

Dataset	Model	Train (s)	Val (s)	Test (s)
tgbl-wiki	TNCN	21.45	250.49	251.52
	NAT	74.92	298.6	298.41
tgbl-review	TNCN	1649	4788	4695
	NAT	422	7516	7461
tgbl-coin	TNCN	4920	28716	28805
	NAT	1896	30398	30176

Table 15: Memory Consumption of different methods on TGB Datasets. All these experiments are conducted with NVIDIA GeForce RTX 4090. ("-" stands for out of memory)

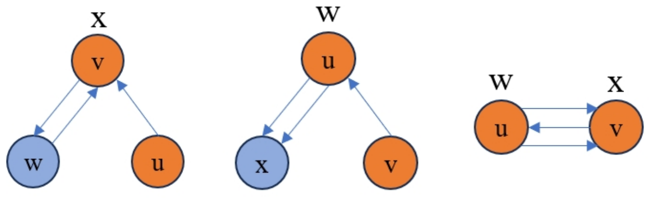
Inference Stage(MB)	Wiki	Review	Coin	Comment	Flight
TGN	1050	6416	11496	13838	1182
DyGFormer	3174	5512	17428	-	-
GraphMixer	4914	5388	17360	-	-
CNE-N	1154	4628	16760	-	-
DyRep	998	6416	9300	10418	1126
TGAT	5600	6588	18526	-	-
CAWN	7656	10898	-	-	-
TCL	2204	5276	17292	-	-
TNCN	1420	9586	11990	13842	1188

Algorithm 1 Pipeline of TNCN

```
1:  for positive batch data $(\text{posu}, \text{posv}, t)$ do
2:      neg_batch ← negative sampling
3:      for batch_data in {pos_data, neg_data} do
4:          mem, hist_events ← memory_module(batch_data);
5:                                  ▷ Get node memory and historical interactions
6:          emb ← transform(mem, hist_events);
7:                                  ▷ Get the node embedding
8:          CN_mat ← CN_extractor(hist_events);
9:                                  ▷ Obtain the CNs for given node pair in a batch
10:         NCN_emb ← AGG(emb, CN_mat);
11:                                 ▷ Aggregate the embeddings of CNs
12:         $p$ ← Pred(emb_u, emb_v, NCN_emb);
13:                                 ▷ Calculate the probability of future links
14:     end for
15:     mem ← memory_update(pos_data);
16:                                 ▷ Update the memory module
17: end for
```

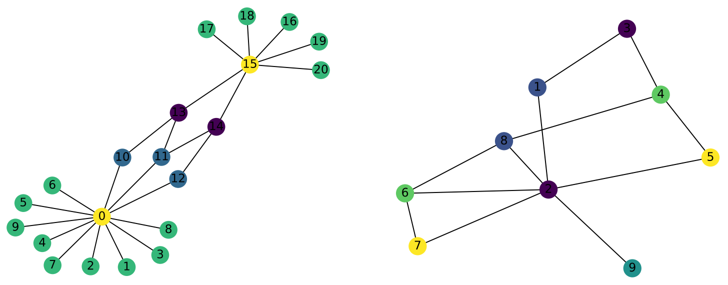
Algorithm 2 Procedure of CN Extraction

```
1: Input: Temporal adjacency matrix of node $u$ and $v$, denoted as $A \in \mathbb{R}^{N \times N}$ and $B \in \mathbb{R}^{N \times N}$. $A$
   and $B$ are the sub-matrix from the whole adjacency matrix of the temporal graph, containing $u$, $v$
   and their individual neighbors. $N$ is the total number of their temporal neighbors and themselves.
2: Output: $k_{th}$ hop Common Neighbors of $u$ and $v$, $CN_k \in \mathbb{R}^N$.
3:
4: Stage 1: Generate up to $k$-hop neighbors.
5:     $A \leftarrow A + I$, $B \leftarrow B + I$.                                  ▷ add self-loop
6:     Compute $A^{k-1}$ and $A^k$, $B^{k-1}$ and $B^k$.          ▷ obtain up to $k-1$ / $k$ hop adjacency matrix
7:     $N_{0\sim k-1}(u) = A^{k-1}[id(u)]$, $N_{0\sim k}(u) = A^k[id(u)]$.         ▷ Similar to $v$. $N(u) \in \mathbb{R}^N$
8:
9: Stage 2: Get $0 \sim k-1$ and $0 \sim k$ hop common neighbors.
10:    $CN_{0\sim k-1} = N_{0\sim k-1}(u) \odot N_{0\sim k-1}(v)$,                     ▷ $CN \in \mathbb{R}^N$
11:    $CN_{0\sim k} = N_{0\sim k}(u) \odot N_{0\sim k}(v)$.             ▷ perform sparse matrix hadamard product
12:
13: Stage 3: Get exact $k$-hop common neighbors.
14:    $CN_k = CN_{0\sim k} - CN_{0\sim k-1}$.                        ▷ perform sparse matrix subtraction
15:
16: Finally we get the exact $k$-hop common neighbors between node $u$ and $v$.
```



(a) $x = v$ (b) $w = u$ (c) $x = v$ & $w = u$

Figure 6: Here shows the special cases related to $(1, 2)$-hop CNs computation. Note that the graph is undirected, while the directed arrows implies the path direction used to determine the corresponding hop numbers.



(a) tgbl-wiki (b) tgbl-coin

Figure 7: Two case studies from TGB.

CN, node 4 and 6 are the $(2,1)$-hop, and node 2 is both their $(1,1)$ and $(2,1)$ hop. The $(2,2)$-hop CNs are node 5 and 7, while node 9 being a special $(2,2)$-hop CN that owns a shared 1-hop node 2. With the abundant CN information, TNCN will be more likely to predict it as a positive future edge.

H.2 Some Statistics about Common Neighbors on Comment Dataset

Here we show some statistics about the structural information of common neighbors on Comment dataset, which may account for the lower performance of TNCN. We have collected four variants of statistics including (1) "s-nei/d-nei": the number of recent neighbors (without repetition) of the positive src/dst nodes, (2) "cn": the number of common neighbors of the positive node pairs, and (3) "prev": their previous interaction times recently. All the values are the average from the test set. The statistics are shown in the following Table 16.

Table 16: Statistics about CNs on Comment Dataset. ("s" stands for the "surprise" event, "ns" for not-surprise; "p" for the case where the ground-truth event ranks 1st in our prediction, "np" for not.)

	s-nei	d-nei	cn	prev
s-p	8.43	9.03	0.87	0.39
s-np	8.89	8.78	0.07	0.02
ns-p	7.83	8.05	2.64	0.69
ns-np	8.76	8.88	0.17	0.07

From the table we can find that, although src/dst nodes from both surprise and not-surprise edges have similar number of neighbors, their common neighbors and the previous contact frequecy are distinct far away. Nodes from surprising edges always share less CNs and interaction times, making CN features less effective in the prediction. From another aspect, the successfully predicted events typically have their nodes with more CNs and historical interactions, which indicates that our TNCN model may be misled by the node pair with more CNs in the prediction.

I Proofs

In this section, we give proofs on theorems 4.4, 4.7 and 4.6.

Theorem I.1. *(Theorem 4.4. Ability of encoding k-hop event). Given a k-hop event $\{(u_i, u_{i+1}, t_{u_i,u_{i+1}}) \mid i \in \{0,\dots,k-1\}, k \geq 1\}$. If the node embedding of u_0 at time t_{u_0,u_1}, can be formally derived by the encoding function Enc$(\{(u_i, u_{i+1}, t_{u_i,u_{i+1}}) \mid i \in \{0,\dots,k-1\}, k \geq 1\})$, then the learning method is considered capable of encoding the k-hop event. The following outline the encoding capabilities of different learning schemes:*

- *Memory-based approach can encode any k-hop events with $k = 1$.*
- *Memory-based approach can encode any monotone k-hop events with arbitrary k.*
- *k-hop-subgraph-based approach can encode any k'-hop events with $k' \leq k$*

Proof. In the following analysis, we establish the encoding efficacy of the memory-based approach. Consider a k-hop event with the simplifying assumption that $k = 1$, which reduces the event to the tuple (u_0, u_1, t_{u_0,u_1}). By adhering to the predefined schematics of the memory-based methodology, the memory state $Mem(u_0, t_{u_0,u_1})$ is updated via the function f_{mem} such that $Mem(u_0, t_{u_0,u_1}) = f_{mem}(Mem(u_0, t'), e_{u_0,u_1}^{t_{u_0,u_1}}, t_{u_0,u_1} - t')$. Let us denote the encoding function as $Enc = f_{mem}(Mem(u_0, t'), \dots)$. It is our intention to demonstrate that this memory-based framework is capable of encoding any k-hop event for $k = 1$.

We consider the encoding of an arbitrary monotonically increasing k-hop temporal event sequence within a memory-based approach. The induction principle is applied to demonstrate the capability of this approach. For the base case, $k = 1$, the encoding has been shown to be feasible. Now, assume the proposition holds for a k'-hop event; that is, any k'-hop temporal sequence of monotonically increasing events can be encoded using a memory-based approach. This assumption implies that there exists an embedding function such that

$$Emb(u_0, t_{u_0,u_1}) = \textit{Enc}((u_i, u_{i+1}, t_{u_i,u_{i+1}}) \mid i \in 0,\dots,k'-1), \tag{7}$$

for all event sequences with k' hops, where $k' \geq 1$. Given an arbitrary $k'+1$-hop event, which can be partitioned into an initial event (u_0, u_1, t_{u_0,u_1}) and a subsequent k'-hop sequence. The existence of an encoding function for the k'-hop sequence assures that

$$\begin{aligned} Emb(u_0, t_{u_0,u_1}) &= f_{emb}(Mem(u_0, t_{u_1,u_2})) \\ &= f_{emb}(f_{mem}(Mem(u_0, t_{u_1,u_2}), \\ &\quad Mem(u_1, t_{u_1,u_2}), e_{u_0,u_1}^{t_{u_0,u_1}}, t_{u_0,u_1} - t_{u_1,u_2}), \\ Mem(u_1, t_{u_1,u_2}) &= f_{emb}^{-1}(Emb(u_1, t_{u_1,u_2})) \\ &= f_{emb}^{-1}(Enc(\{(u_i, u_{i+1}, t_{u_i,u_{i+1}}) \\ &\mid i \in \{1, \ldots, k'\}, k' \geq 1\}) \end{aligned} \tag{8}$$

Subsequently, it is demonstrated that $Emb(u_0, t_{u_0,u_1})$ provides an encoding for both the initial event and the k'-hop sequence, thereby affirming its efficacy in encoding the entire $k'+1$-hop event. This concludes the inductive step and substantiates the inductive argument.

We consider a k-hop-subgraph-based approach for our analysis. It is evident that a k-hop subgraph encompasses any k'-hop events, where $k' \leq k$. Furthermore, the aggregation methodology assimilates all nodes contained within the subgraph. Collectively, these observations substantiate the theorem in question.

□

Theorem I.2. *(Theorem 4.7. Learning method time complexity). Denote the time complexity of a learning method as a function of the total number of events processed during training. For a given graph $\mathcal{G}$ with the number of nodes designated as $|\mathcal{N}|$ and the number of edges as $|\mathcal{E}|$, the following assertions hold:*

- *For the memory-based approach, the time complexity is $\Theta(|\mathcal{E}|)$.*
- *For k-hop-subgraph-based with $k = 1$, the lower-bound time complexity is $\Omega\left(\frac{|\mathcal{E}|^2}{|\mathcal{N}|}\right)$, and the upper-bound time complexity is $\mathcal{O}\left(\frac{|\mathcal{E}|^2}{|\mathcal{N}|} + |\mathcal{E}||\mathcal{N}|\right)$*
- *For k-hop-subgraph-based with $k = 2$, the upper-bound time complexity is $\mathcal{O}\left((\frac{|\mathcal{E}|^2}{|\mathcal{N}|} + |\mathcal{E}||\mathcal{N}|)^{\frac{3}{2}}\right)$*

Proof. In the proposed theorem, the time complexity is denoted as the aggregate quantity of events processed throughout the training phase. The objective herein is to ascertain the precise count of such utilized events.

In the context of the memory-based methodology, it is evident that each event is utilized a singular time. Consequently, the cumulative number of events is expressed as $|\mathcal{E}|$, which infers that the time complexity adheres to the order of $\Theta(|\mathcal{E}|)$.

In the context of k-hop-subgraph-based algorithms wherein $k = 1$, an event (u, v, t) is exploited once for every incident event within the neighborhood of vertices u or v. Without loss of generality, we focus on all events within the 1-hop-subgraph of vertex u. The aggregate count of events processed is given by $\sum_{i=1}^{d(u)} i = \Theta\left(d(u)^2\right)$, where $d(u)$ denotes the degree of vertex u. Consequently, the computational complexity is fundamentally proportional to $\sum_{u \in \mathcal{N}} d(u)^2$. Drawing on the results of de Caen [22], the lower bound on the time complexity is established as $\Omega\left(\frac{|\mathcal{E}|^2}{|\mathcal{N}|}\right)$, whereas the upper bound is determined as $\mathcal{O}\left(\frac{|\mathcal{E}|^2}{|\mathcal{N}|} + |\mathcal{E}||\mathcal{N}|\right)$

In the context of k-hop-subgraph-based algorithms wherein $k = 2$, we adopt similar strategy where each event(u, v, t) will only be utilized once another event within the subgraph of u or v is firstly considered. The total number of events can be formulated as $\sum_{u \in \mathcal{N}} d(u) \sum_{v \in \mathcal{N}_u} \sum_{w \in \mathcal{N}_v} d(w)$. Replacing $d(u)$ as X_i, $\sum_{v \in \mathcal{N}_u} \sum_{w \in \mathcal{N}_v}$ as Y_i, we reformulated is as $\sum_{i \in |\mathcal{N}|} X_i Y_i$, satisfying $\sum_{i \in |\mathcal{N}|} X_i^2 = \sum_{u \in \mathcal{N}} d(u)^2$ and $\sum_{i \in |\mathcal{N}|} Y_i^2 = \sum_{u \in \mathcal{N}} d(u)^4$. Following Cauchy inequality and conclusions of $\sum_{u \in \mathcal{N}} d(u)^2$, we got the the upper-bound time complexity is $\mathcal{O}\left((\frac{|\mathcal{E}|^2}{|\mathcal{N}|} + |\mathcal{E}||\mathcal{N}|)^{\frac{3}{2}}\right)$ □

Theorem I.3. *(Theorem 4.6. Expressivity of TNCN)*

1. *TNCN is strictly more expressive than CN, RA, and AA.*
2. *TNCN is strictly more expressive than Jodie with the same dimension of time encoding, DyRep with the same aggregation function, TGAT with the same attention layers and neighbors, and TGN under identical condition for all module choices.*

Proof. (1) We first give definitions of these structural features under temporal settings. Given two nodes u and v, the structural features before time t are defined as follows:

$$\begin{aligned} CN(u,v,t) &= \sum_{w\in N_1^t(u)\cap N_1^t(v)} 1, \\ RA(u,v,t) &= \sum_{w\in N_1^t(u)\cap N_1^t(v)} \frac{1}{d(w)}, \\ AA(u,v,t) &= \sum_{w\in N_1^t(u)\cap N_1^t(v)} \frac{1}{\log d(w)} \end{aligned} \tag{9}$$

Given a node u, the degree of node u is the number of events e with an endpoint at node u. Without loss of generality (W.L.O.G.), we consider node u as the source node, and the events are $\{(u, v_i, t_i) \mid i \in \{0, \dots, k-1\}, k \geq 1\}$. Each time a new event is given, the embedding of node u is updated by

$$mem_u^{t_i} = \mathit{upd}_{src}(mem_u^{t_{i-1}}, \mathit{msgfunc}_{src}(e_{u,v_i}^{t_i})). \tag{10}$$

With the MPNN universal approximation theorem, *msgfunc* can be a constant function, and upd_{src} can be an addition function. Thus,

$$mem_u^{t_{k-1}} = d(u). \tag{11}$$

Then the embedding can learn arbitrary functions of node degrees, i.e.,

$$emb_u^t = f(d(u)). \tag{12}$$

Thus, the neural common neighbor
$TNCN_1(u,v) = \oplus_{w\in N_1^t(u)\cap N_1^t(v)}\, emb_w^t$ can express Equation 9.

Extending to situations where the common neighbor node has some features we want to learn, the traditional CN, RA, and AA cannot accommodate this. However, our TNCN can express these features, demonstrating that TNCN is strictly more expressive than CN, RA, and AA.

(2) We first prove that TNCN is strictly more expressive than TGN. As TNCN possesses the same memory-based framework as TGN to derive the emb_u^t, TNCN at least has the same expressivity as TGN with identical module choices. Additionally, from part (1) of this theorem we know that TNCN can capture certain heuristics such as CN where TGN fails. Hence TNCN is strictly more expressive than TGN.

Since Jodie, DyRep and TGAT are specific instances of TGN with minor changes, we next shift our object to prove that TGN has no less expressivity than the three methods.

For TGAT, it retains only the Graph Attention mechanism for embedding functions, removing all memory and message modules present in TGN. Consequently, TGAT lacks the capacity to incorporate temporal memory updates and message passing, making it less expressive than TGN.

Jodie differs from TGN in the memory updater and embedding function. Specifically, it utilizes an RNN rather than a GRU, although GRU is a general variant of RNN. Additionally, Jodie employs the time embedding as $emb_u^t = (1 + \Delta t \cdot w) \otimes mem_u^{t^-}$, where w are learnable parameters. However, TGN's Graph Attention uses not only the memory $mem_u^{t^-}$ but also the historical edge features $e_{u,v}^{t'}$, and learns the coefficient via multi-head attention, thus achieving a higher expressivity.

DyRep also modifies the memory updater to an RNN. Meanwhile, it changes the embedding function to *Identity* and message function to *Graph Attention*. Although it just swaps these two modules, the expressivity is reduced in the temporal setting. When the model attempts to make its prediction, it first generates the embeddings with the embedding function. So naturally the identity function processes less information in the emb_u^t than graph attention. After obtaining the prediction result, the new messages will be calculated by the message function and fed into memory updater, where the emb_u^t can not get access. Hence TGN is more expressive than DyRep, especially when the recent information can make large benefits.

In conclusion, TNCN is strictly more expressive than Jodie, DyRep, TGAT and TGN under the same condition.

$\square$