8-2021

Forecasting interaction order on temporal graphs

Wenwen XIA

Yuchen LI
Singapore Management University, yuchenli@smu.edu.sg

Jianwei TIAN

Shenghong LI

Forecasting Interaction Order on Temporal Graphs

Wenwen Xia*
Shanghai Jiao Tong University
xiawenwen@sjtu.edu.cn

Yuchen Li
Singapore Management University
yuchenli@smu.edu.sg

Jianwei Tian
State Grid Hunan Electric Power Company
tianjw@hn.sgcc.com.cn

Shenghong Li
Shanghai Jiao Tong University
shli@sjtu.edu.cn

ABSTRACT

Link prediction is a fundamental task for graph analysis and the topic has been studied extensively for static or dynamic graphs. Essentially, the link prediction is formulated as a binary classification problem about two nodes. However, for temporal graphs, links (or interactions) among node sets appear in sequential orders. And the orders may lead to interesting applications. While a binary link prediction formulation fails to handle such an order-sensitive case. In this paper, we focus on such an interaction order prediction (IOP) problem among a given node set on temporal graphs. For the technical aspect, we develop a graph neural network model named Temporal ATtention network (TAT), which utilizes the fine-grained time information on temporal graphs by encoding continuous real-valued timestamps as vectors. For each transformation layer of the model, we devise an attention mechanism to aggregate neighborhoods' information based on their representations and time encodings attached to their specific edges. We also propose a novel training scheme to address the permutation-sensitive property of the IOP problem. Experiments on several real-world temporal graphs reveal that TAT outperforms some state-of-the-art graph neural networks by 55% on average under the AUC metric.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Temporal reasoning**.

KEYWORDS

Temporal Graphs, Graph Neural Networks, Interaction Order Prediction

ACM Reference Format:
Wenwen Xia, Yuchen Li, Jianwei Tian, and Shenghong Li. 2021. Forecasting Interaction Order on Temporal Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467341

*The work was done when visiting Singapore Management University.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467341

1 INTRODUCTION

Graphs are ubiquitous data structure in a wide range of real-world applications, including social networks [43], trading networks [12], molecule structures [47], and recommendation systems [10]. A key task in graph analysis is link prediction, which has been extensively studied in the literature [20, 49, 50]. Most prior works study the binary classification problem of predicting the existence of future links/interactions between node pairs [49, 50]. However, many real-world graphs are highly dynamic and the edges are associated with timestamps, indicating the order of their occurrences. The binary formulation overlooks the sequential order when considering edges among node sets with more than two nodes. While this sequential order information is ubiquitous in time-critical scenarios, such as social interaction networks [40], financial transactions [25], and many others.

The IOP Problem. In this work, we focus on the interaction order prediction (IOP) among a given node set on temporal graphs. Given a node set **s** on a temporal graph G, we predict the order of edges' occurrence between nodes in **s** by learning from the historical interaction order on G. The IOP problem enable fine-grained analyses on temporal graphs. For example, the order of information propagation is important for rumor control as it enables proactive propagation route analysis and authorities may then take early actions with minimum interventions to mitigate undesired consequences [5, 6]. Another example is the sequential recommendation system (SRS), which recommends items in order based on a user's sequential interaction patterns with previous items [30, 39]. SRS predicts the interaction order between a user and a set of items, which can be modeled as an instance of the IOP problem. Unlike the link prediction which focuses on determining the existence of future edges, the above applications necessitate accurate predictions of interaction orders among specific node sets, motivating the IOD problem. In this paper, we focus on interaction orders among size-3 node sets, i.e., temporal triangles, which have been identified as critical patterns in temporal graph analyses [1, 27, 38].

Technical Challenges. Built on the success of graph neural networks (GNNs) for many graph analysis tasks, including node classification [15], link prediction [49], and graph classification [14], we aim to apply GNNs to address the focused problem. However, there are several gaps to employ existing GNN methods for the IOP problem. *Firstly,* most previous GNN models take static graphs as inputs and learn time-irrelevant representations [2, 7, 15, 35, 37]. Recently, several graph learning models tailored for temporal graphs have emerged [4, 26, 28, 33, 44]. These works mainly rely on GNNs

and recurrent architectures (e.g., LSTM, GRU) [25, 26, 33], or attention modules [28] to capture the temporal information. While these architectures lack precise processing of real-valued timestamps, mostly utilizing their chronological order information. *Secondly,* most GNN models consider a single edge between two nodes, while there may exist multiple edges with different timestamps between two nodes in a temporal graph. In IOP, it is both critical and challenging to consider multiple edges and their fine-grained temporal pattern to learn comprehensive representations. *Thirdly,* IOD requires to take a node sequence $\mathbf{s}$ as the input for labeling the interaction order pattern among nodes in $\mathbf{s}$. The sequential input imposes a unique *permutation-variant* property: the prediction labels for two different node sequences of the same node set should be different but indicate the same underlying interaction order. In contrast, existing GNNs focus on permutation-invariant tasks, e.g., graph classification [14]. How to learn a consistent prediction model which is compatible with different permutation sequences of the same node set has not been studied in prior GNN works.

Solution. To address the gaps, we propose the **T**emporal **AT**tention network (TAT) model by integrating time encodings with the graph attention mechanism. For the first gap, we discretize continuous timestamps and encode these timestamps as d-dimensional vectors to capture the time information. For the second gap, we employ the attention mechanism to compute aggregation weights for all temporal interactions between each node pair. All these temporal interactions are treated using an identical attention computation layer. Concretely, at each transformation layer, we concatenate time encodings with node's representations from the former layer for computing attention scores. For the third gap, we devise a novel permutation-aware training scheme. Given a node sequence $\mathbf{s}$ and its training label l, we randomly permute $\mathbf{s}$ to $\bar{\mathbf{s}}$. The correct interaction order for $\bar{\mathbf{s}}$ should correspond with that for $\mathbf{s}$ under a permutation. Hence $\mathbf{s}$ and $\bar{\mathbf{s}}$ will be jointly optimized under TAT to preserve the permutation-variant property. However, optimizing the model for $\mathbf{s}$ and $\bar{\mathbf{s}}$ jointly may lead to a stability issue, since the optimization target for learning the permutation relationship between $\mathbf{s}$ and $\bar{\mathbf{s}}$ is computed by the model itself and requires frequent update of model parameters. Such a training characteristic leads to oscillation of the optimization target, and hence oscillation of the gradients. Inspired by the deep Q-learning [22], we devise a double-model trick to address the stability issue.

We summarize our contributions as follows: (1) We formulate the interaction order prediction problem on temporal graphs, which is a largely ignored sequential prediction problem in previous graph learning literature; (2) We propose the TAT model which integrates time encodings with the attention mechanism to extract fine-grained temporal representations. Besides, we devise a novel training scheme to preserve the intrinsic permutation-variant property of the problem; (3) We conduct extensive experiments on several real-world temporal graph datasets. The results reveal that the proposed model outperforms several state-of-the-art GNNs by 55% for AUC and 54% for ACC on average. We conduct detailed ablation analyses to demonstrate the contributions of individual components. The implementation of the TAT model is released [1].

[1] https://github.com/xiawenwen49/TAT-code

2 RELATED WORKS

In this section, we present related works on non-temporal GNNs and temporal GNNs.

2.1 Non-temporal GNNs

GNNs have achieved great success for many graph analysis problems recently. Some pioneering works integrate the convolution operation in traditional Euclidean data space (e.g., images, audios) with non-Euclidean graph-structured data based on graph spectral theory [2, 7]. Subsequently, several seminal variants are proposed to improve GNN's performance or address their limitations in practical scenarios, including GCN [15], GraphSAGE [13], GAT [35] and many others. These GNN models fall into two board categories: *spectral* methods and *spatial* methods. Spectral methods operate in the spectral domain of graph signals and devise learnable graph spectral filters. In contrast, spatial methods focus on utilizing graph topology for information aggregation where the aggregation weights are learned [42]. Both spectral and spatial methods can be generalized under the message-passing framework in which each node aggregates messages from its neighbors and transforms to a new representation utilizing aggregated messages and learnable transformation units [45, 46]. However, most of these models only work on static graphs or dynamic graphs while ignoring temporal information, e.g., considering edge/node addition or deletion sequentially, but without timestamps [13].

2.2 Temporal GNNs

Recently many works devised for learning on temporal or dynamic graphs have surged. These models capture topological and temporal information by miscellaneous approaches, including temporal random walks [23], recurrent neural networks [26, 48], attention modules [29, 44], temporal convolutions [51] and temporal point processes [32]. Despite their diverse approaches, we can roughly categorize them as discrete temporal graph models (DTG) and continuous temporal graphs models (CTG), according to their treatment for graphs. DTG models discretize a temporal graph into multiple snapshots. Then static methods are applied to each snapshot, or each snapshot will be encoded and connected by RNNs [21, 25], attention modules [29], or convolutions [8, 48, 51]. CTG models directly utilize continuous-time information on temporal graphs. We will discuss several representative CTG models in detail. Jodie utilizes RNNs and attention modules to predict node embedding trajectories for bipartite user-item graphs [16]. Dyrep utilizes RNNs and the point process framework to learn time-varying node representations [33]. TGN uses RNNs and memory blocks to update representations of the source and destination node each time a new edge appears [26]. StemGNN adopts Graph Fourier transform, discrete Fourier transform, and convolution operations in one layer for time-series forecasting [3]. Among these works, the most relevant work with ours is the TGAT [44], which utilizes the self-attention to aggregate information from temporal edges, as with our works. While there are some key differences between our work and the TGAT. Firstly, the computation of TGAT is restricted by chronological orders, hence temporal edges not following these orders are ignored. In our model, all temporal edges observed will be considered in the forward computation, to capture more comprehensive

temporal patterns. Secondly, TGAT computes a time-varying representation for each node. Hence, one node's representation may be recomputed many times. While we compute a base representation for each node and concatenate time encodings with the base representation to obtain time-related ones. This leads to more efficient forward computations.

3 THE TAT MODEL

In this section, we first formalize the interaction order prediction (IOP) problem. Subsequently, we present the TAT model for IOP.

3.1 Problem Formulation

We first present the definition of the temporal graph, then formalize the IOP problem. As discussed in section 1, we focus on interactions among triple node sets in this paper.

Definition 3.1 (Temporal graph). Let $G = (V, E, T)$ denotes a temporal graph, where $V = \{v_1, v_2, \cdots, v_n\}$ represents the node set and E denotes the edge set. Let $T = \{T_{ij}\}, i, j \in [1, n]$, denotes the timestamps attached on all edges where $T_{ij} = \{t_1, t_2, \cdots, t_l\}$ denotes the l real-valued interaction timestamps between node i and j. Further, we define $T_{ij} = \emptyset$ if $(i, j) \notin E$ and $T_{ii} = \{0\}$ for all $i \in [1, n]$. In other words, we add self-loops in E and there exist at least one timestamp in T_{ij} if $(i, j) \in E$. Note that all timestamps in T have been observed at the current moment. We regard G as an undirected graph. Besides, nodes may be attached with d-dimensional feature vectors and we denote the feature matrix as $F \in \mathbb{R}^{n \times d}$ if it exists.

Definition 3.2 (Interaction Order Prediction (IOP)). For a given temporal graph G and a node set $\mathbf{s} = \{n_i, n_j, n_k\}$, we aim to predict in which order temporal interactions will form among these three nodes in the impending future, leveraging the observed interaction history T and the graph topology.

Concretely, to provide meaningful predictions, numbering these nodes becomes necessary. In other words, we need to list the node set as a **node sequence** $\mathbf{s} = [n_i, n_j, n_k]$. We consider the *earliest* unseen/future edge between two pairs in $\mathbf{s}$ for prediction, i.e., there are at most three node-pairs which we predict the order of their occurrence: $(\mathbf{s}_1, \mathbf{s}_2)$, $(\mathbf{s}_1, \mathbf{s}_3)$, $(\mathbf{s}_2, \mathbf{s}_3)$. One can see there exist six possible orders among three node pairs. Hence we formulate the IOP as a six-class classification problem. The model takes tuple $(G, \mathbf{s})$ as input and outputs a six-dimensional prediction vector. We encode these six orders in Figure 2. Note that the subscript $i \in \{1, 2, 3\}$ of $\mathbf{s}$ denotes the **i-th node** in the given sequence $\mathbf{s}$, while the subscript i of n indicates the **specific node** n_i in V.

3.2 Model Framework

We illustrate the framework of the model in Figure 1. The method mainly consists of three components: (1) Initial node feature construction; (2) Time encoder; (3) TAT transformation layer.

- **Initial features**: Since we may not obtain node features in G (i.e., F may not exist), we propose to carefully construct initial features for each node to represent their spatial properties.
- **Time encoder**: We encode a continuous timestamp as a k dimensional vector to utilize the time information in a fine-grained manner. The time vector will be integrated with each layer's representation in layer transformation.
- **TAT transformation layer**: We construct the TAT model by stacking several TAT transformation layers to aggregate localized neighbors' information.

3.3 Feature Construction

In IOP, the correct class is indexed based on the order of node-pairs. While these nodes are identified by their positions in the sequence $\mathbf{s}$. Hence the model should differentiate nodes with different positions in $\mathbf{s}$ and other nodes not in $\mathbf{s}$. Therefore, we propose to encode the *role* of each node on the graph with a role vector $\mathbf{r}$.

Role encoding. For nodes not in $\mathbf{s}$, we encode them with zero vectors. While for nodes in $\mathbf{s}$, we encode each of them using a 3-dimensional one-hot vector $\mathbf{e}_k$, i.e.,

$$\mathbf{r}_i = \begin{cases} \mathbf{0}, & n_i \notin \mathbf{s} \\ \mathbf{e}_k, & n_i = \mathbf{s}_k \end{cases}, \qquad \forall n_i \in V \tag{1}$$

Besides, we also construct feature vectors to represent each nodes' spatial location information in the graph. Recently, several works have proposed to construct spatial features and concatenate these features with original node features before feeding node features into GNN models [18, 50]. The construction of spatial features could provably improve the identification power of conventional message-passing GNN models. Therefore, to indicate each nodes' relative spatial location information regarding nodes in $\mathbf{s}$ as *anchors*, we propose to utilize the following spatial encoding for each node.

Spatial encoding. Spatial encoding reflects the *relative* spatial information of other nodes if given $\mathbf{s}$. In this paper, we adopt the shortest path encoding as the indicator of spatial information of nodes lying in a proximal region of $\mathbf{s}$ [18]. The shortest path encoding for node u given node set $\mathbf{s}$ can be formalized in Eq (2).

$$\begin{aligned} \zeta(u|\mathbf{s}) &= \text{ENCODE}(\{\zeta(u|v)|v \in \mathbf{s}\}) \\ \zeta(u|v) &= \text{Argmin}_k\{(A^k)_{uv} > 0\} \end{aligned} \tag{2}$$

A denotes the adjacency matrix. The ENCODE function encodes these lengths with a K-dimensional vector, where K is the hop of local subgraphs. For example, if a node n_i has shortest lengths $\zeta(n_i|\mathbf{s}) = \{1, 1, 2\}$, then the shortest path encoding vector $x_{\text{sp}}(n_i)$ should be $[0, 2, 1, 0]$. $x_{\text{sp}}(n_i)$ means there exist two nodes in $\mathbf{s}$ which have shortest path length 1 with node n_i and one node which has shortest path length 2 with node n_i. In this paper, we focus on 3-hop local subgraphs, hence we set the vector length to 4. We denote the shortest path encoding node feature matrix as X_{sp}, i.e., $x_{\text{sp}}(n_i)$ is the n_i-th row of X_{sp}.

Note that the choice of the spatial encoding can be flexible. We adopt the shortest path encoder for computational efficiency and effectiveness. One may use other schemes like the Random Walk Landing Probabilities (RWLP) [18]. The RWLP represents the probability of being reached from a specific node with a hop constraint. If the root node is determined as node n_i, the landing probability vector given hop k can be represented as $p^{(k)}$ in Eq (3).

$$\mathbf{p}^{(k)} = e_i \tilde{L}^k \tag{3}$$

where e_i is the one-hot vector and $\tilde{L} = AD^{-1}$ is the random walk normalized Laplacian matrix. D is the degree matrix. We denote

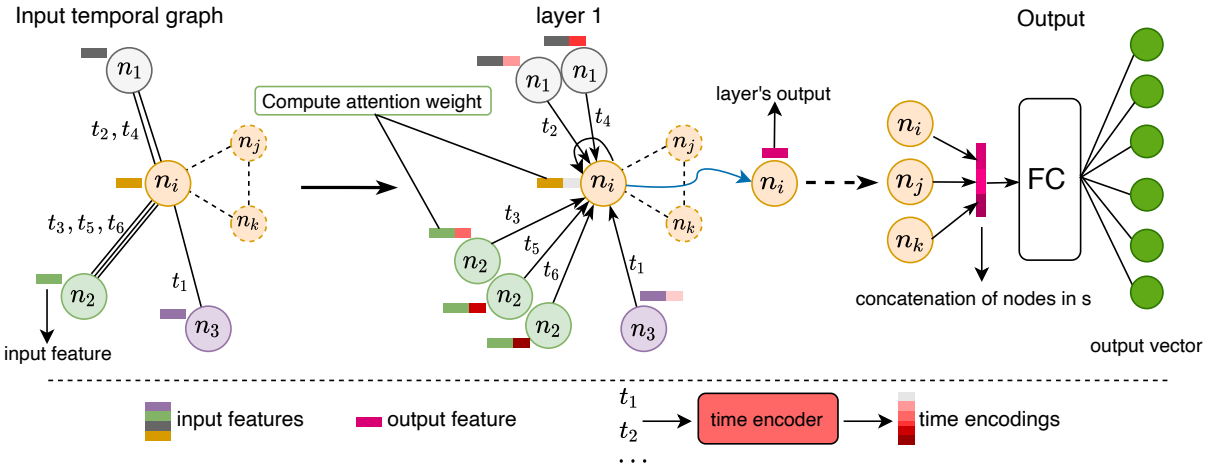


Figure 1: The framework of TAT model. Each temporal edge is attached with a timestamp. We construct initial features for all nodes and represent timestamps with time encodings. For each layer, we concatenate node representations from former layer with time encodings and utilize the attention mechanism for information aggregation.

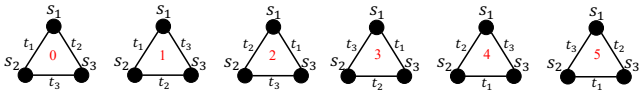


Figure 2: Possible interaction orders for the IOP. s_1, s_2, s_3 indicates the 1st, 2ed, and the 3rd node in the node sequence s. For all patterns, we have $t_1 < t_2 < t_3$, and the red numbers represent class indexes.

the spatial RWLP feature matrix as X_{rw}, which can be constructed as in Eq (4).

$$X_{\text{rw}} = [\mathbf{p}^{(0)}, \mathbf{p}^{(1)}, \cdots, \mathbf{p}^{(K)}] \tag{4}$$

where K is the hop number. For multiple nodes in $\mathbf{s}$, we can compute the X_{rw} regarding each node in $\mathbf{s}$ as the root node n_i and sum these X_{rw} up to obtain the final feature matrix. We will compare the performance of different spatial encoding features in section 6.2. Note that if the original node feature matrix F exists, we concatenate X and F as model input.

3.4 Time Encoder

In this subsection, we describe the time encoding scheme in detail, which captures the fine-grained temporal information of historical interactions. We formalize the time encoder as a function $\Phi : \mathbb{R} \to \mathbb{R}^d$ which maps a continuous time t to a d-dimensional vector, where d is a hyperparameter.

We adopt the idea of *position encoding* in Transformer [34] to encode the continuous time in our approach. Position encoding originally encodes discrete positions of each words in a sentence and is computed as in Eq (5).

$$\mathbf{pe}_{pos,2i} = \sin(\frac{pos}{L^{2i/d}}), \quad \mathbf{pe}_{pos,2i+1} = \cos(\frac{pos}{L^{2i/d}}) \tag{5}$$

where $\mathbf{pe} \in \mathbb{R}^{L \times d}$ is the position encoding matrix, L is the predetermined number of discrete positions, and d is the even encoding dimension. i ranges from 0 to $d/2 - 1$. Each discrete position pos in $[0, L-1]$ is represented by the pos-th row of $\mathbf{pe}$. While timestamps on temporal graphs are continuous real numbers. Therefore, we discrete the continuous time to integers and these integers are regarded as position indexes. The discretization interval is adaptively computed according to the observed timestamps in T. Concretely, we adopt the following *uniform discretization*,

$$\Phi(t) = [\mathbf{pe}_{\lfloor t/\Delta \rfloor, 0}, \mathbf{pe}_{\lfloor t/\Delta \rfloor, 1}, \cdots, \mathbf{pe}_{\lfloor t/\Delta \rfloor, d-1}] \tag{6}$$

where Δ is the discretization interval. By default, we set Δ as the mean time interval of two chronologically consecutive timestamps in T for a specific dataset.

One may use a more straightforward approach to encode the continuous timestamps without discretization, e.g., using the *harmonic encoder* recently proposed in [44].

$$\Phi(t) = \sqrt{\frac{2}{d}}[\cos(\omega_1 t), \sin(\omega_1 t), \cdots, \cos(\omega_{d/2} t), \sin(\omega_{d/2} t)] \tag{7}$$

where $\omega_1, \omega_2, \cdots, \omega_{d/2}$ are learnable time encoder parameters. The harmonic encoder obtains more flexibility by inducing additional learnable encoder parameters. While learnable parameters also raise model complexity. We will compare the performance of the empty time encoder (zeros vectors), the uniform discretization time encoder, and the harmonic encoder in section 6.2.

3.5 TAT Layer

The proposed TAT model stacks several transformation layers to obtain the final output. In this subsection, we present the transformation details within one message-passing layer of our model. On a high level, the output representation of each node within one layer is obtained by its neighbors' representations from the former layer and the temporal interactions information between the node and its neighbors.

More specifically, we adopt the attention mechanism to compute each interaction's weight for aggregating neighborhood information. The transformation framework is illustrated in Figure 1. For a temporal interaction with timestamp t_k between source node n_j

and target node n_i, we compute the attention score from node j to node i in layer l using Eq (8).

$$\text{score}_{i,j,t_k} = \alpha_l^T [(W_{l,1}[h_i^{l-1} \| \Phi(0)]) \| (W_{l,2}[h_j^{l-1} \| \Phi(t_k)])] \quad (8)$$

h_i^{l-1} represents node n_i's representation from layer $l-1$ and $\Phi(t_k)$ is the time encoding of time t_k. For source node n_i, we always concatenate h_i^{l-1} with $\Phi(0)$ to highlight the time encoding information of $\Phi(t_k)$. While α_l, $W_{l,1}$, and $W_{l,2}$ are learnable parameters. If there exist multiple interactions between n_i and n_j (this is common in practical datasets), we will flatten all these temporal edges and compute multiple scores, as shown in the middle of Figure 1. Note that in the aggregation process of a TAT layer, all interactions observed will be considered, instead of just chronological paths, which is a fundamental difference with TGAT in [44].

Once attention scores of all temporal interactions are computed, aggregation attention weights between node n_i and all of its (flattened) neighbors can be obtained using Eq (9).

$$\text{att}_{i,j,t_k} = \frac{\exp\{\text{score}_{i,j,t_k}\}}{\sum_{n \in \{N(i) \cup \{i\}\}} \sum_{t \in T_{in}} \exp\{\text{score}_{i,n,t}\}} \quad (9)$$

After the computation of attention weights for all temporal edges, the output representation of node n_i for layer l can be computed as follows:

$$h_i^l = \sum_{n \in \{N(i) \cup \{i\}} \sum_{t \in T_{in}} \text{att}_{i,n,t} W_{l,3}[h_n^{l-1} \| \Phi(t)] \quad (10)$$

where $W_{l,3}$ is another learnable parameter matrix. Note that we concatenate each node's hidden representation with time encoding in each layer of the proposed TAT model for attention computation, instead of only in the first input layer. Hence the model could utilize temporal information in each message-passing layer to aggregate intermediate representations.

3.6 Output Layer

After obtaining node representations of the final layer, we concatenate representations of nodes in the sequence $\mathbf{s}$ for final prediction, as illustrated in Figure 1. The merging and prediction layer is formalized as Eq (11).

$$\text{out} = \text{Softmax}(\text{Linear}([h_{s_1}^K \| h_{s_2}^K \| h_{s_3}^K])) \quad (11)$$

Note that the concatenation order of h^K must strictly follows the node order in $\mathbf{s}$.

4 TRAINING FRAMEWORK

In this section, we present a novel training framework of the TAT model, which is illustrated in Figure 3. We first discuss the *permutation-sensitive* property then present training objectives.

Permutation-sensitive property. One can see that the model takes a tuple $(G, \mathbf{s})$ as input and is trained with the corresponding label y. If we permute $\mathbf{s}$ to $\bar{\mathbf{s}}$ and feed the permuted tuple $(G, \bar{\mathbf{s}})$ into model, the ideal label should also be changed correspondingly. In other words, for different sequences with an identical node set, we expect the model to output different labels but infer an identical interaction order. For example, as illustrated in Figure 3, if $\mathbf{s} = [n_i, n_j, n_k]$ and $y = 0$, the interaction order between $\{n_i, n_j, n_k\}$ should be (n_i, n_j), (n_i, n_k), (n_j, n_k). While if $\bar{\mathbf{s}} = [n_k, n_i, n_j]$, then $y = 4$ indicates the same underlying order, according to Figure 2.

Training objectives. To handle the permutation-sensitive property, we propose the permutation-aware training scheme, which explicitly instructs the model to capture this permutation relationship by constructing the *permutation loss*.

$$\text{Loss}_{\text{perm}} = \|f_p(M(G, \bar{\mathbf{s}}), \mathbf{s}, y) - M(G, \mathbf{s})\|_2 \quad (12)$$

In Eq (12), $\bar{\mathbf{s}}$ denotes a random permutation of $\mathbf{s}$ and $M(G, \mathbf{s})$ is the output prediction vector. f_p operates on model's prediction vector, which permutes the prediction vector to align with $M(G, \mathbf{s})$. We can predetermine f_p since the number of permutations is finite. However, computing the permutation target vector $f_p(M(G, \bar{\mathbf{s}}), \mathbf{s}, y)$ using the original model M directly introduces unstability in training, since the model is updated after each minibatch, introducing large variance for the permutation loss target $M(G, \bar{\mathbf{s}})$. Hence, we propose to utilize a copy M_{target} of the original model M to compute the permutation target vector. M_{target} is periodically updated following M. The periodical copy scheme is motivated by the double Q-networks adopted in DQN [22], which aims to optimize the Bellman equation loss [31], as shown in Eq (13).

$$\text{Loss}_{\text{bellman}} = (Q(s_t, a_t) - (r_t + \max_{a \in A} Q(s_{t+1}, a)))^2 \quad (13)$$

in which the target value $r_t + \max_{a \in A} Q(s_{t+1}, a)$ is computed using a copy Q_{copy} of the current model Q. We confront an analogous scenario in the computation of permutation loss, in which the permutation target vector $f_p(M(G, \bar{\mathbf{s}}), \mathbf{s}, y)$ requires the training model M itself. Hence after utilizing a copy M_{target} of M, we amend the permutation loss as follows:

$$\text{Loss}_{\text{perm}} = \|f_p(M_{\text{target}}(G, \bar{\mathbf{s}}), \mathbf{s}, y) - M(G, \mathbf{s})\|_2 \quad (14)$$

The final optimization objective consists of three aspects, i.e., the cross-entropy, the permutation loss, and the regularization term, as shown in Eq (15).

$$\text{Loss} = \text{Loss}_{\text{CR}} + \gamma \text{Loss}_{\text{perm}} + \lambda \text{Loss}_{\text{reg}} \quad (15)$$

where $\text{Loss}_{\text{CR}} = \text{CrossEntropy}(\mathbf{e}_y, M(G, \mathbf{s}))$ and we choose L_{reg} as $\|\theta\|_2$, the L2 norm of original model's parameters. In practice, we add the $\text{Loss}_{\text{perm}}$ after several epochs, with details in the Appendix.

5 EXPERIMENTAL SETUP

In this section, we present the experimental setup, including datasets, preprocessing, baselines, and model implementation details.

5.1 Datasets and Preprocessing

We use the following temporal graph datasets. We regard temporal edges as undirected for all datasets.

- **COLLEGEMSG**. This dataset collects private message communications of an online social community at the University of California, Irvine [24]. Each temporal edge (n_i, n_j, t) represents a user n_i sends a message to n_j at time t.
- **EMAIL-EU**. This dataset collects emails between members of a European research institution [17]. Each temporal edge (n_i, n_j, t) represents an email sent from n_i to n_j at time t.
- **FBWALL** This dataset contains all of the wall posts from the Facebook New Orleans network [36]. Each temporal edge (n_i, n_j, t) records that a user n_j posted on n_i's wall with timestamp t.

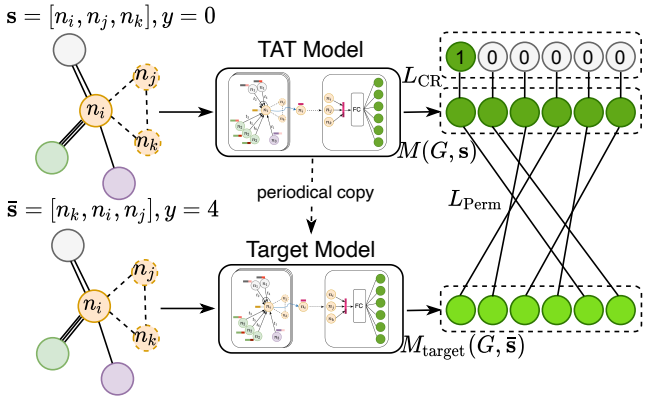


Figure 3: Training framework for the TAT model. For each training sample, we construct a corresponding permuted sample and compute its prediction using the target model. The prediction of M_{target} will be permuted to align with the prediciton of M to compute the permutation loss.

Table 1: Statistics of temporal graph datasets used in our experiments. # Triads denotes the number of observed temporal triangles.

Dataset	$\|V\|$	$\|E\|$	$\|T\|$	# Triads
COLLEGEMSG	1899	13838	59835	14319
EMAIL-EU	986	16064	332334	105461
FBWALL	45813	183412	855542	122852
SMS-A	44430	53866	548182	16401

- **SMS-A**. This dataset collects interactions of a short message service (SMS) from a mobile phone operator [41]. Each temporal edge (n_i, n_j, t) means a user n_i sent a short message to user n_j at time t.

The statistics of these datasets are listed in Table 1. We collect all triangle node sets on these temporal graphs to construct the sequence set $\mathbf{S}$ and determine the corresponding label set Y according to the earliest timestamp between each node pair.

Next, we present the preprocessing details, which mainly consist of two steps: subgraph extraction and timestamp elimination.

Subgraph extraction: One can see that for a message-passing GNN model with K layers, obtaining the output layer's representation of a specific node requires its K-hop neighbors' information. As one layer aggregates 1-hop neighbors' information, for each training sequence $\mathbf{s}$, we extract a K-hop subgraph around nodes in $\mathbf{s}$, where K is the number of layers for our proposed model and other baselines.

Timestamp elimination: Further, if we denote the earliest interaction time among nodes in $\mathbf{s}$ on graph G as $t_{\max}$, we remove all temporal interactions with timestamps larger than $t_{\max}$ on the subgraph around $\mathbf{s}$, since we aim to predict the *future* interaction order among nodes in $\mathbf{s}$. All temporal interactions occurring after $t_{\max}$ should be invisible during model training. Then we minus all timestamps using the largest timestamp to represent relative values. Finally, we regard the combination of each processed subgraph, $\mathbf{s}$, and label y as one data case. Once we collected all data cases, we sort these data cases according to $t_{\max}$ in ascending order. We use the first 70% cases as the train set and split the rest equally as the validation set and the test set.

5.2 Baselines

We compare our model with the following GNN models.

- GCN [15]. GCN adopts the normalized graph Laplacian matrix as the aggregation operator and transforms each node's representation with a shared parameter matrix.
- GAT [35]. GAT utilizes the attention mechanism to compute attention weights for each edge and aggregates neighbors' representations based on these attention weights.
- GraphSAGE [13]. GraphSAGE inductively aggregates each node's neighborhood features and then concatenates each node's feature with the aggregated feature to perform transformations.
- TAGCN [9]. TAGCN designs multiple topology-adaptive filters operating on the vertex domain in each layer. Each filter is a polynomial of the normalized graph Laplacian matrix with learnable coefficients.
- DE-GNN [18]. DE-GNN theoretically strengthens the distinguish power of general message-passing GNN models by adding spatial features for each node before feeding node features into GNN models.
- TGAT [44]. TGAT computes time-related representations for nodes given timestamps. The information flow follows the chronological order strictly.

For all baselines, we concatenate the representations of nodes in $\mathbf{s}$ in order from the final layer and adopt a 1-layer fully-connected network with a Softmax layer for prediction, as indicated in Eq (11).

5.3 Metrics & Implementation Details

We adopt three metrics to evaluate all approaches. (1) Accuracy (ACC), which measures the native prediction accuracy for all classes. (2) Top-2 Accuracy (ACC@2), which considers top-2 high-probability classes, i.e., if the model gives the label class the highest or the 2nd-highest probability, ACC@2 will count. We adopt this metric considering that the ACC may be inadequate for this multi-class prediction problem. (3) Area Under the Receiver Operating Characteristic Curve (AUC), which measures the area under curve for each class. We adopt the *one-vs-rest* manner to compute the AUC and average the results of all classes [11]. For all these metrics, a higher value indicates a better performance.

For the proposed TAT model and all other baselines, we adopt a 2-layer structure, as reported in most works [9, 13, 15, 35]. We set the number of hidden units to 128. For attention models including our TAT, GAT, and TGAT, we set the number of attention heads to 4. For DE-GNN, we choose the spatial features as the shortest path length and choose the base GNN model as the TAGCN, as reported in their original paper [18]. For our TAT model, we set the time encoding dimension to 64 by default and the maximum encoding index to 3e4. We set γ to $1e-1$, λ to $1e-4$ by default. We set the time discretization interval Δ as the mean time interval of all consecutive timestamps for each dataset. Besides, we will study the effects of several important hyperparameters in section 6.4.

Table 2: The prediction performance comparison of different methods. For the Input column, R denotes the role encoding, S denotes the spatial encoding, and T denotes the time encoding. The best and the second-best results are highlighted in bold and underlined font respectively. We denote the TAT model with (without) the permutation loss by TAT (TAT-perm).

Methods	Input	COLLEGEMSG			EMAIL-EU			FBWALL			SMS-A		
		ACC	ACC@2	AUC	ACC	ACC@2	AUC	ACC	ACC@2	AUC	ACC	ACC@2	AUC
GCN	R&S	0.349	0.632	0.766	0.280	0.519	0.665	0.288	0.559	0.700	0.238	0.451	0.602
GAT	R&S	0.347	0.645	0.765	0.274	0.518	0.665	0.270	0.548	0.685	0.227	0.430	0.207
GraphSAGE	R&S	0.338	0.633	0.757	0.276	0.514	0.677	0.301	0.566	0.701	0.231	0.435	0.582
TAGCN	R&S	0.349	0.649	0.764	0.290	0.529	0.676	0.279	0.546	0.688	0.250	0.473	0.617
DE-GNN	R&S	0.337	0.643	0.762	0.289	0.529	0.676	0.314	0.589	0.715	0.244	0.459	0.601
TGAT	R&T	0.277	0.543	0.680	0.197	0.385	0.545	0.247	0.490	0.659	0.170	0.330	0.510
TAT-perm	R&S&T	0.406	0.755	0.811	0.400	0.711	0.792	0.496	0.737	0.807	0.474	**0.781**	0.822
TAT	R&S&T	**0.440**	**0.763**	**0.825**	**0.456**	**0.769**	**0.810**	**0.569**	**0.785**	**0.845**	**0.508**	0.771	**0.825**

6 RESULTS

In this section, we discuss the experimental results. We first present the prediction performance comparison. Then we conduct the ablation study to analyze the effects of different modules in the proposed model. Further, we study the effects of several important hyperparameters. Finally, we visualize the attention heads to shed some light into the learned model.

6.1 Prediction Performance

The prediction results are presented in Table 2. We highlight the best and the second-best results in each column in bold and underlined font respectively. We also compare the performance of our proposed TAT model with permutation-aware optimization to its variant without this component, denoted by TAT and TAT-perm respectively. We can see that GCN, GAT, GraphSAGE, and other GNN models designed for static graphs achieve similar prediction performance across three temporal graphs. While TGAT performs relatively unsatisfactorily. We analyze that TGAT only aggregates information along chronological paths, which may be unsuitable for the IOP problem. We observe that the TAT performs consistently better than other baselines. Compared with the second-best model, TAT improves the ACC by 26.1%, 57.2%, 81.2% 103.2%, and improves the ACC@2 by 17.6%, 45.4%, 33.3% 65.1%, on CollegeMsg, Email-EU, Facebook-Wall, and SMS-A dataset. We can further obtain the following observations from prediction results: (1) Comparing TAT-perm with TAT, we can see that the permutation-aware training scheme improves the prediction performance, achieving about 8.4%, 14.0%, 14.7%, 7.2% improvement of the ACC on four datasets. (2) Further, compared with TGAT, TAT achieves better prediction performance, which may reveal that considering all temporal edges is more suitable than just considering chronological paths.

6.2 Ablation Study

To gain a better understanding of different components in the TAT model, including distinct time encoders, spatial features, and the attention mechanism, we present ablation experiments as follows.

Time encoders. We compare the performance of TAT with the empty time encoder, the uniform discretization time encoder, and the harmonic time encoder. The results on four datasets are shown in Figure 4(a). Empty time encoder performs the worst on all datasets since it ignores all temporal information. Concretely, the empty encoder performs 10%, 24.8%, 22.6%, 34.6% worse than the uniform encoder for the AUC on four datasets. The worse performance indicates that historical temporal information is critical for reasonable predictions. For the harmonic time encoder, the encoder parameters are learned on the fly. However, the results suggest that the adopted uniform discretization time encoder achieves about 4.8%, 28.8%, 18.7%, 25.7% better than the harmonic encoder for the AUC metric on four datasets. The results reveal that the harmonic encoder may be more difficult to be learned for appropriate temporal representations. While encoding temporal information adopting the uniform discretization may be more straightforward and suitable for the IOP problem.

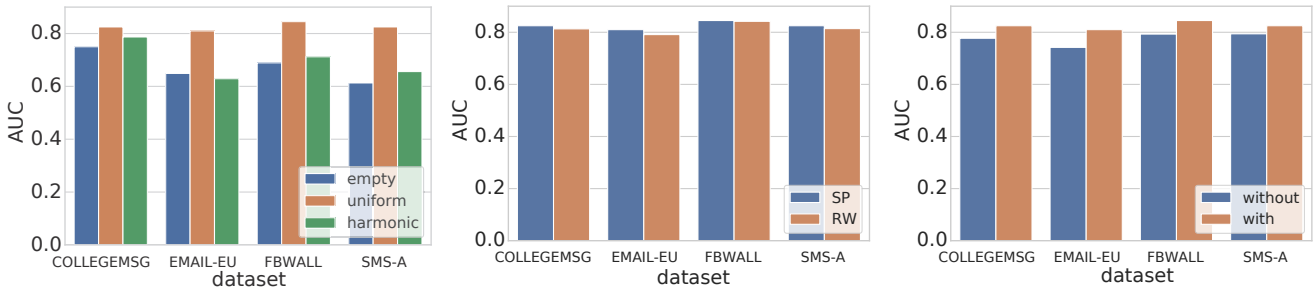
Spatial features. We compare the performance of TAT with shortest-path encoding features and with random walk landing probability features. The performance results on three datasets are shown in Figure 4(b). Figure 4(b) reveals that both spatial features offer similar performance. SP slightly outperforms RW by 1.5%, 2.4%, 0.4%, 1.4% for the AUC metric on four datasets. We adopt the SP as the default spatial feature in the TAT for computational efficiency and convenience as stated in section 3.3.

Attentions. We also compare the performance of TAT between with and without the attention mechanism. For TAT without attention, we replace the learnable attention weight α_l in Eq (8) with constants as in the GCN model. The performance results on three datasets are shown in Figure 4(c). We can see the attention mechanism is critical for better predictions. TAT model with the attention mechanism surpasses its counterpart without the mechanism by about 6.2%, 9.2%, 6.6%, 3.9% for the AUC metric on four datasets. The performance comparison in Figure 4(c) reveals that different temporal interactions should be treated differently when aggregating node hidden representations.

6.3 Attention Analysis

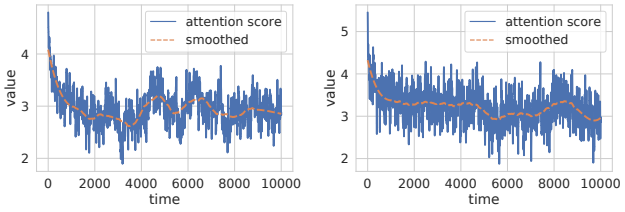
To have a better understanding of learned patterns of the TAT, we analyze the learned attention weights in this subsection. We train the TAT model on the COLLEGEMSG dataset and all hyperparameters are set as in section 5.3.

We adopt the learned model parameters to compute the attention scores for timestamp indexes from 0 to 10000 for visualization.



(a) Time encoders (b) Spatial features (c) Attention mechanism

Figure 4: Effects of different components of TAT model.



(a) Attention head 0 (b) Attention head 1

Figure 5: Visualization of learned attention heads.

Table 3: The hyperparameters.

Hyperparameter	Description	Values
layers	Number of model layers	1, **2**, 3, 4
d	Time encoding dimension	32, **64**, 128, 256
γ	Weight of $\text{Loss}_{\text{perm}}$	1e-2, **1e-1**, 1, 10

Table 4: Effects of different hyperparameters.

layers		d		γ	
value	AUC	value	AUC	value	AUC
1	0.810	32	0.800	1e-2	0.807
2	**0.825**	64	**0.825**	1e-1	0.825
3	0.813	128	0.810	1	**0.83**
4	0.813	256	0.801	10	0.793

These attention scores are computed using Eq (8). The linear transformation matrix and the attention vector are loaded using the first layer's parameter of the trained model for visualization. We plot the scores of two heads in Figure 5. We can see that while each attention head reveals slightly different patterns, all of them decrease w.r.t time indexes. These figures reveal that the learned model pays less attention to temporal interactions chronologically far from the current observation time. Other attention heads reflect analogous patterns and we put them in the Appendix.

6.4 Hyperparameter Analysis

In this subsection, we study the sensitivity of several important hyperparameters. We list the information of these hyperparameters in Table 3. We conduct experiments on the COLLEGEMSG dataset and fix other hyperparameters to their default values.

Model layer. It controls how many hops of neighbors' information one model can aggregate. The experimental results in Table 4 reveal that too many layers (i.e., three or four layers) may degrade the prediction performance. As indicated in [19], too many layers may degrade the performance of general message-passing GNNs, which coincides with experimental results in Table 4. However, much few layers (i.e., 1 layer) may also result in unsatisfactory performance. We suggest that one-hop neighbors may not provide sufficient spatial and interaction information for this prediction problem. Proper layers of stacking is important for an acceptable performance of our TAT model.

Time encoding dimension d. It controls the resolution of time encodings. From Eq (5), we can see that for each dimension in $[0, d-1]$, the changing angular velocity of two consecutive time encodings is $\frac{1}{L^{2i/d}}$. A larger dimension d leads to a larger angular velocity, which means the value of each dimension changes more steeply as the time index changes. In other words, two consecutive time encodings are more distinguishable. However, a larger dimension also results in larger parameter complexity. From the results in Table 4, we can see that the effect of time encoding dimension is analogous with that of the number of layers. Extreme values (small or large) lead to unsatisfactory performance.

Permutation loss weight γ. It determines the importance of permutation optimization objective in section 4. From Figure 4, we can see that the prediction performance is positively correlated with γ in the range of [1e-2, 1], which reveals that the proposed permutation optimization scheme is effective to promote the TAT's prediction performance. However, as the permutation objective is computed by the target model, which may induce considerable bias if the weight γ is too large. For example, we can see from Table 4 that setting γ to a large value 10 degrades the performance. While setting γ to 1e-1 or 1 performs better than that of 1e-2 or 10. Hence we set the default value to 1e-1, considering the balance between prediction performance and bias.

7 CONCLUSIONS

In this paper, we consider forecasting the order of future interactions among specific node sets based on historical temporal interactions and local graph topologies. The proposed TAT model utilizes time encodings to capture fine-grained time information and the attention mechanism to aggregate neighbor's information based on their temporal interactions. From the experimental results, we conclude that time information is critical for an accurate prediction of interaction orders. The permutation-aware optimization improves the TAT model's prediction performance further. Visualizations of learned attention heads suggest that the model tends to pay more

attention to recent interactions when making decisions. Furthermore, the problem studied in this paper suggests that training GNN models and learning graph representations related to a predetermined node or edge set deserve more research efforts. In the future, we aim to verify the TAT model's scalability for larger node sets, e.g., 4-node set, which has more edges and much more orders than the 3-node set case, and of course, much more complexity.

Acknowledgements This research work is funded by the National Nature Science Foundation of China under Grant 61971283 and U20B2072, Shanghai Municipal Science and Technology Major Project under Grant 2021SHZDZX0102, and 2020 Industrial Internet Innovation Development Project of MIIT of P.R. China "Smart energy Internet security situation awareness platform project". Wenwen Xia is also supported by the China Scholarship Council.

REFERENCES

[1] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and deep locally connected networks on graphs. In *ICLR*.

[3] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Conguri Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. 2020. Spectral Temporal Graph Neural Network for Multivariate Time-series Forecasting. In *NIPS*. 1–13.

[4] Huiyuan Chen and Jing Li. 2018. Exploiting structural and temporal evolution in dynamic link prediction. In *CIKM*.

[5] Tong Chen, Xue Li, Hongzhi Yin, and Jun Zhang. 2018. Call attention to rumors: Deep attention based recurrent neural networks for early rumor detection. In *PAKDD*.

[6] Anh Dang, Abidalrahman Moh'd, Aminul Islam, and Evangelos Milios. 2019. Early detection of rumor veracity in social media. In *HICSS*.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.

[8] Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. 2019. In *Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting*.

[9] Jian Du, Shanghang Zhang, Guanhang Wu, José M.F. Moura, and Soummya Kar. 2017. Topology adaptive graph convolutional networks. *arXiv* (2017). arXiv:1710.10370

[10] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*.

[11] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[12] Eva Fraňková, Jan Fousek, Lukáš Kala, and Jan Labohý. 2014. Transaction network analysis for studying Local Exchange Trading Systems (LETS): Research potentials and limitations. *Ecological Economics* 107 (2014), 266 – 275.

[13] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 59.

[14] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. 2016. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design* 30, 8 (2016), 595–608.

[15] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[16] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*.

[17] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2–es.

[18] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning. In *NIPS*.

[19] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*.

[20] L. L. Linyuan and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications* 390, 6 (2011), 1150–1170.

[21] Franco Manessi, Alessandro Rozza, and Mario Manzo. 2020. Dynamic graph convolutional networks. *Pattern Recognition* 97 (2020), 107000.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[23] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-Time Dynamic Network Embeddings. In *WWW*.

[24] Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. 2009. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology* 60, 5 (2009), 911–932.

[25] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. 2020. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*.

[26] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. (2020). arXiv:2006.10637

[27] Martin Rosvall, Alcides V. Esquivel, Andrea Lancichinetti, Jevin D. West, and Renaud Lambiotte. 2014. Memory in network flows and its effects on spreading dynamics and community detection. *Nature Communications* 5 (2014).

[28] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2019. Dynamic graph representation learning via self-attention networks. In *ICLRW*.

[29] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 519–527.

[30] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*.

[31] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[32] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2018. Representation learning over dynamic graphs. arXiv:1803.04051

[33] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *ICLR*.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*.

[35] Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

[36] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. 2009. On the Evolution of User Interaction in Facebook. In *WOSN*.

[37] Hongyang Wang, Qingfei Meng, Ju Fan, Yuchen Li, Laizhong Cui, Xiaoman Zhao, Chong Peng, Gong Chen, and Xiaoyong Du. 2020. Social Influence Does Matter: User Action Prediction for In-Feed Advertising. In *AAAI*.

[38] Jingjing Wang, Yanhao Wang, Wenjun Jiang, Yuchen Li, and Kian-Lee Tan. 2020. Efficient Sampling Algorithms for Approximate Temporal Motif Counting. In *CIKM*. 1505–1514.

[39] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z Sheng, and Mehmet Orgun. 2019. *Sequential Recommender Systems: Challenges, Progress and Prospects*. Technical Report.

[40] Yanhao Wang, Qi Fan, Yuchen Li, and Kian-Lee Tan. 2017. Real-Time Influence Maximization on Dynamic Social Streams. *Proceedings of the VLDB Endowment* 10, 7 (2017).

[41] Ye Wu, Changsong Zhou, Jinghua Xiao, Jürgen Kurths, and Hans Joachim Schellnhuber. 2010. Evidence for a bimodal distribution in human communication. *Proceedings of the national academy of sciences* 107, 44 (2010), 18803–18808.

[42] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[43] Wenwen Xia, Yuchen Li, Wu Jun, and Li Shenghong. 2021. DeepIS : Susceptibility Estimation on Social Networks. In *WSDM*.

[44] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. In *ICLR*.

[45] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.

[46] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-Ichi Kawarabayashi, and Stefanie Jegelka. 2018. *Representation Learning on Graphs with Jumping Knowledge Networks*. Technical Report.

[47] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NIPS*.

[48] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI*.

[49] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. In *NIPS*.

[50] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2020. Revisting Graph Neural Networks For Link Prediction. In *arXiv*.

[51] Qi Zhang, Jianlong Chang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2020. Spatio-Temporal Graph Structure Learning for Traffic Forecasting. In *AAAI*.

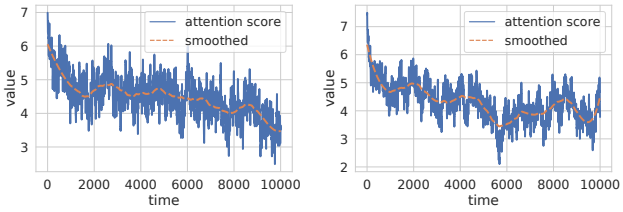
A ALGORITHMIC PIPELINE

The overall training pipeline for the TAT model is shown in Algorithm 1. Note that in algorithm 1, we add the permutation loss after several training epochs, instead of adding it from the scratch. We observe from empirical studies that, adding the permutation loss after several epochs leads to better performance, compared with adding it from the beginning. We analyze the reason may be that after several epochs of training, the model has learned relatively meaningful parameters for the prediction problem. Hence the permutation loss computed by the target model could provide meaningful targets and gradients instructing the model to capture the permutation relationship.

Algorithm 1 The training pipeline for proposed TAT model.

Input: Graph $G = (V, E, T)$, sequence set $\mathbf{S} = \{\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \cdots\}$, labels $\mathbf{Y} = \{y^{(1)}, y^{(2)}, \cdots\}$, λ, γ, e_{th}, e_{total}, e_{update}.
Initialization: Initialize θ in M and $\theta_{\text{target}} \leftarrow \theta$ in M_{target}.
for $i = 0, 1, 2, \cdots, e_{\text{total}}$ **do**
  for minibatch (S, Y) **do**
    $\hat{Y} = M(G, S)$
    if $i > e_{\text{th}}$ **then**
      Permute all $\mathbf{s} \in S$ as $\bar{S}$, compute $\hat{\bar{Y}} = M_{\text{target}}(G, \bar{S})$
      Compute Loss = CrossEntropy$(\hat{Y}, Y) + \gamma \|f_p(\hat{\bar{Y}}, S, Y) - \hat{Y}\|_2 + \lambda\|\theta\|_2$
    else
      Compute Loss = CrossEntropy$(\hat{Y}, Y) + \lambda\|\theta\|_2$
    end if
    Compute gradients $\frac{\partial \text{Loss}}{\partial \theta}$ and update θ with one step
  end for
  if $i \bmod e_{\text{update}} == 0$ **then**
    $\theta_{\text{target}} \leftarrow \theta$
  end if
end for

B VISUALIZATION OF OTHER ATTENTION HEADS



(a) Attention head 2 (b) Attention head 3

Figure 6: Visualization of learned attention heads.

In Eq (8), we replace the representation h_i and h_j with zero vectors, and replace the $\Phi(t_k)$ with time encodings of time indexes from 0 to 10000 for visualization, as stated in section 6.3. Figure 6 visualizes the rest two attention heads of the learned TAT model's first layer. These patters reflect subtle differences with each other, but resemble others in a large picture. All attention heads together reveal that the model learned to focus on more recent interactions.