STAM: A Spatiotemporal Aggregation Method for Graph Neural Network-based Recommendation

Zhen Yang[*]
DCST, Tsinghua University
Beijing, China
yangz21@mails.tsinghua.edu.cn

Ming Ding[*]
DCST, Tsinghua University
Beijing, China
dm18@mails.tsinghua.edu.cn

Bin Xu[§]
DCST, Tsinghua University
Beijing, China
xubin@tsinghua.edu.cn

Hongxia Yang
Alibaba Group
Hangzhou, China
yang.yhx@alibaba-inc.com

Jie Tang[§]
DCST, Tsinghua University
Beijing, China
jietang@tsinghua.edu.cn

ABSTRACT

Graph neural network-based recommendation systems are blossoming recently, and its core component is aggregation methods that determine neighbor embedding learning. Prior arts usually focus on how to aggregate information from the perspective of spatial structure information, but temporal information about neighbors is left insufficiently explored.

In this work, we propose a spatiotemporal aggregation method STAM to efficiently incorporate temporal information into neighbor embedding learning. STAM generates spatiotemporal neighbor embeddings from the perspectives of spatial structure information and temporal information, facilitating the development of aggregation methods from spatial to spatiotemporal. STAM utilizes the Scaled Dot-Product Attention to capture temporal orders of one-hop neighbors and employs multi-head attention to perform joint attention over different latent subspaces. We utilize STAM for GNN-based recommendation to learn users and items embeddings. Extensive experiments demonstrate that STAM brings significant improvements on GNN-based recommendation compared with spatial-based aggregation methods, e.g., 24% for MovieLens, 8% for Amazon, and 13% for Taobao in terms of *MRR*@20.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**; • **Networks** → **Network algorithms**.

KEYWORDS

Spatiotemporal Aggregation Method; Self-Attention; GNN-based Recommendation

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512041

ACM Reference Format:
Zhen Yang, Ming Ding, Bin Xu, Hongxia Yang, Jie Tang. 2022. STAM: A Spatiotemporal Aggregation Method for Graph Neural Network-based Recommendation. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512041

1 INTRODUCTION

Recommender systems are a critical tool to perform personalized information filtering [6, 56], having been applied to many online services. The essence of recommendation is to learn latent representations for users and items from the past user-item interactions and predict items that a user interacts with in the future. Most works take recommendation as the matrix completion task [25]. Collaborative filtering (CF) [37, 38] is a traditional recommendation method to predict scores between users and items. Matrix factorization (MF) [9, 32] learns the latent vectors for users and items to reconstruct the interaction matrix. Due to the success of deep learning, recent works use neural networks[8, 21], such as multi-layer perceptions (MLP), to capture the nonlinear interaction between users and items [18]. Recent years have witnessed tremendous interest in graph neural networks (GNNs) [14, 24, 44], and its information propagation mechanism improves the downstream tasks, demonstrating promising prospects in many challenging tasks.

Owing to the superiority of graph neural networks in learning on graph data, GNN-based recommendations [1, 17, 48, 56] model the user-item interactions as a graph and leverage GNNs to incorporate the spatial structure information into the embeddings. Massive GNN-based recommendation works have investigated good aggregation methods to learn the embeddings for users and items from the perspective of spatial structure information. Existing aggregation methods roughly fall into four groups [52]: (1)"mean pooling" treats the neighbors equally; (2) "degree normalization" assigns weights to nodes based on the graph structure; (3) "attentive pooling" differentiates the importance of neighbors with attention mechanism and (4) "central node augmentation" considers the affinity between nodes and uses the central node to filter the neighbors' message. However, the abovementioned methods omit the neighbor's temporal information which is a vital signal that contributes

[*] Equal contribution. Codes are available at https://github.com/zyang-16/STAM.
[§] Corresponding Authors.

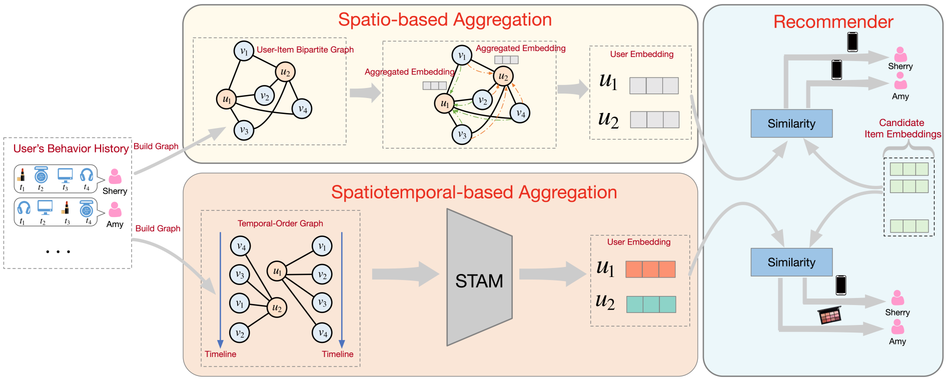

Figure 1: A motivating example of STAM, demonstrating the importance of temporal information for neighbor aggregation.

significantly to aggregation in GNN-based recommendations but is not encoded in neighbor embedding learning.

In preceding GNN-based recommendations [1, 17, 48, 56], the neighbor aggregation methods only touch upon spatial structure information but ignore the neighbor's temporal information and cannot capture the user's dynamic interests. The fundamental capability of the E-commerce platform is to collect abundant user's behavior history along with temporal information. However, the previous aggregation methods in GNN-based recommendations have not yet fully utilized this temporal information to learn high-quality embeddings and only demonstrate the intrinsic interest of users. To deal with the abovementioned issue, we integrate the temporal information into the aggregation methods to facilitate the development of aggregation from spatial to spatiotemporal.

To fully understand the motivation of this work, we give a motivating example of STAM to show the important role of temporal information in GNN-based recommendation. In Figure 1, we select two users (Amy and Sherry) from the user's behavior history to build a user-item bipartite graph and a temporal-order graph for GNN-based recommendation, respectively. In spatial-based aggregation, the aggregated neighbor embeddings for Amy and Sherry are identical because they interacted with the same items. Similar to LightGCN, we omit non-linear transformation and utilize the aggregated neighbor embeddings as one-hop user embeddings. Thus, the recommender recommends an identical item for Amy and Sherry in spatial-based aggregation. However, recommended items for Amy and Sherry are different in spatiotemporal-based aggregation. In spatiotemporal-based aggregation, we incorporate temporal information into neighbor embedding learning, where temporal orders play a vital role in capturing users' dynamic interests and user cluster changes over time.

In this work, we give prominence to the issue that temporal information is still not utilized for neighbor aggregation in GNN-based recommendation. Instead of directly aggregating neighbor's information from spatial structure, we propose a novel aggregation method named **S**patio**T**emporal **A**ggregation **M**ethod (STAM) to integrate temporal information into neighbor embedding learning, promoting the development of aggregation methods from spatial to spatiotemporal. STAM generates spatiotemporal neighbor embeddings from the perspectives of spatial structure and temporal order. In STAM, the Scaled Dot-Product Attention is applied to capture the temporal orders of one-hop neighbors. To further improve the expressiveness of STAM, we learn multiple attention heads in STAM that perform joint attention on various latent subspaces.

STAM cannot alter the framework of GNN-based recommendation, which can be naturally plugged into existing GNN-based recommendation models. We apply STAM for GNN-based recommendation and compare its performance with representative GNN-based and sequential recommendation models. The experimental results show that STAM outperforms the state-of-the-art baselines, such as average relative increase of 24% for MovieLens, 8% for Amazon, and 13% for Taobao in terms of *MRR*@20, showing the importance of temporal information for aggregation methods. Moreover, we also conduct comparative experiments between STAM and five representative aggregation methods (four spatial-based aggregators and a BiLSTM aggregator) to verify the effectiveness of STAM.

To summarize, the main contributions of this work are as follows:

- We highlight the significance of temporal information for neighbor aggregation in GNN-based recommendation, facilitating the development of aggregation methods from spatial to spatiotemporal.
- We propose a novel aggregation method STAM to incorporate temporal information into neighbor embedding learning, which can be naturally plugged into existing GNN-based recommendation models.
- We conduct extensive experiments to demonstrate the superiority of STAM over spatial-based aggregation methods.

2 PRELIMINARIES AND PROBLEM

In this section, we first review the framework of GNN-based recommendation. We then give the detailed problem statement that concerns the aggregation method with temporal information.

2.1 GNN-based recommendation

2.1.1 ***Embedding Layer.*** GNN-based recommendation models maintain an item embedding matrix $\mathbf{E}_{\mathcal{V}} \in \mathbb{R}^{N \times d}$ and a user embedding matrix $\mathbf{E}_{\mathcal{U}} \in \mathbb{R}^{M \times d}$ to project the one-hot representations to low-dimensional representations, where M and N denote the number of user nodes and item nodes. For any user u (an item v), the look-up operation performs to obtain an embedding vector $\mathbf{e}_u \in \mathbb{R}^d$ ($\mathbf{e}_v \in \mathbb{R}^d$), where d denotes the embedding size. Such user and item embedding matrices serve as an initial state and will be updated by aggregation and propagation.

2.1.2 ***Embedding Aggregation Layer.*** The embedding aggregation layer is responsible for collecting and aggregating neighbors' information, which is a significant component of GNN-based recommendation [17, 48, 56]. In the user-item graph, there are two types of aggregation operations: item aggregation and user aggregation:

$$\begin{aligned} \mathbf{n}_u &= f_{u \leftarrow v}(\mathbf{e}_v | v \in \mathcal{N}_u), \\ \mathbf{n}_v &= f_{v \leftarrow u}(\mathbf{e}_u | u \in \mathcal{N}_v). \end{aligned} \tag{1}$$

where $\mathbf{e}_u$, $\mathbf{e}_v$ are the initial embeddings of user u and item v, $\mathcal{N}_u$ denotes the set of items interacted with the user u (or u's neighbors in the user-item graph), and $\mathcal{N}_v$ represents the set of users who have interacted with the item v (or v's neighbors). $\mathbf{n}_u/\mathbf{n}_v \in \mathbb{R}^d$ is the aggregated neighbor embedding for user u/item v. $f(\cdot)$ is the aggregation function.

2.1.3 ***Embedding Propagation Layer.*** To capture higher-order interactions between user and item, multiple propagation layers are stacked to propagate embeddings in the user-item graph. Let $\mathbf{h}_u^{(l)}$ and $\mathbf{h}_v^{(l)}$ denote user u's and item v's embedding at the l-th propagation layer, respectively. Thereafter, the embeddings in $(l+1)$-th layer depends on two steps: an aggregation operation that aggregates neighbors' embeddings at l-th layer into a fixed-length embedding vector $\mathbf{n}^{(l+1)}$, an update step that takes the aggregated neighbor embedding and its own embedding vector at l-th layer as inputs, and utilizes update function $g(\cdot)$ to obtain the embeddings at $(l+1)$-th layer. Mathematically, the abovementioned two steps could be defined as:

$$\begin{aligned} \mathbf{n}_u^{(l+1)} &= f_{u \leftarrow v}(\mathbf{h}_v^{(l)} | v \in \mathcal{N}_u), \\ \mathbf{h}_u^{(l+1)} &= g(\mathbf{n}_u^{(l+1)}, \mathbf{h}_u^{(l)}). \end{aligned} \tag{2}$$

Similarly, the item embedding at $(l+1)$-th layer $\mathbf{h}_v^{(l+1)}$ also be updated by the abovementioned two steps.

2.1.4 ***Prediction Layer.*** Propagating with L layers, each user/item gather multiple representations $\{\mathbf{h}_u^{(1)}, \cdots, \mathbf{h}_u^{(L)}\}/\{\mathbf{h}_v^{(1)}, \cdots, \mathbf{h}_v^{(L)}\}$. The final user/item embedding $\mathbf{e}_u^*/\mathbf{e}_v^*$ is calculated by the fusion function $o(\cdot)$, which can be formulated as:

$$\begin{aligned} \mathbf{e}_u^* &= o(\mathbf{h}_u^{(1)}, \cdots, \mathbf{h}_u^{(L)}), \\ \mathbf{e}_v^* &= o(\mathbf{h}_v^{(1)}, \cdots, \mathbf{h}_v^{(L)}). \end{aligned} \tag{3}$$

Some works directly use the embedding in the last layer as the final one [50] while some integrate the embeddings of all layers with concatenation [48] or weighted-pooling operations [17].

Thereafter, the inner product is applied to estimate the user's preference towards the target item:

$$\hat{r}_{uv} = \mathbf{e}_u^{*\top} \mathbf{e}_v^* \tag{4}$$

Note that the inner product is also used as the similarity score for recommendation to retrieve top-K candidate items.

2.1.5 ***Joint Training.*** Given a training sample (u, v) with the embedding vectors $(\mathbf{e}_u^*, \mathbf{e}_v^*)$, the likelihood of the user u and the interacted item v can be formulated as:

$$P_\theta(v|u) = \frac{\exp(\mathbf{e}_u^{*\top} \mathbf{e}_v^*)}{\sum_{v_n \in \mathcal{V}} \exp(\mathbf{e}_u^{*\top} \mathbf{e}_{v_n}^*)} = \frac{\exp(\mathbf{e}_u^{*\top} \mathbf{e}_v^*)}{\exp(\mathbf{e}_u^{*\top} \mathbf{e}_v^*) + \sum_{v_n \in \mathcal{V}_u^-} \exp(\mathbf{e}_u^{*\top} \mathbf{e}_{v_n}^*)} \tag{5}$$

where $\mathcal{V}$ denotes the set of all items, $\mathcal{V}_u^-$ is the item set containing items that have not interacted with the user u, v_n denotes a negative item that the user u has not interacted with.

The loss function of GNN-based recommendation is to minimize the following negative log-likelihood:

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} -\log P_\theta(v|u) \tag{6}$$

where $\mathcal{D}$ denotes the all user-item interactions, that is, all observed edges in the user-item graph.

However, the sum operator of equation (5) is computationally expensive. Negative sampling acts as a critical point to deal with this issue and speeds up the training process. The loss function is usually simplified by negative sampling as:

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} -\log \frac{\exp(\mathbf{e}_u^{*\top} \mathbf{e}_v^*)}{\exp(\mathbf{e}_u^{*\top} \mathbf{e}_v^*) + \sum_{v_n \sim p_n(\cdot)} \exp(\mathbf{e}_u^{*\top} \mathbf{e}_{v_n}^*)} \tag{7}$$

where $v_n \sim p_n(\cdot)$ represents the negative sampling strategy proposed in some advanced works [46, 55, 58], the number of sampled negative items is much less than the number of total items.

2.2 Problem Statement

GNN-based recommendation transforms the interactions between users and items into the bipartite graph and leverages graph learning approaches to obtain user/item embeddings. Aggregation methods play a decisive role in the information propagation mechanism for GNN-based recommendation. However, prior works only focus on aggregating neighbors' information from the perspective of spatial structure information but ignore the temporal information.

Thus, we collect the temporal information from user's behavior history and build a user's temporal order $T_u = \{v_1, \cdots, v_S\}$ and an item's temporal order $T_v = \{u_1, \cdots, u_S\}$ for a certain user-item pair (u, v), where S is the number of one-hop neighbors and v_t/u_t records the t-th interacted item/user. In this paper, we improve the existing aggregation methods by leveraging such temporal information, facilitating the development of aggregation methods from spatial to spatiotemporal.

3 METHOD

STAM is a universal aggregation method that incorporates temporal information into neighbor embedding learning, which can be naturally plugged into existing GNN-based recommendation models. Instead of aggregating neighbors' information from spatial structure, STAM simultaneously aggregates neighbors' information from the perspectives of spatial structure and temporal order of one-hop neighbors.

In this section, we firstly present the proposed STAM to learn spatiotemporal neighbor embeddings for each user-item pair (u, v) by leveraging temporal information. Then, we utilize STAM for GNN-based recommendation to learn high-quality embeddings for users and items. Thereafter, the optimization of STAM with negative sampling is demonstrated. Finally, we give the model analysis about the relation of STAM with previous related works and give the analysis of time complexity for STAM.

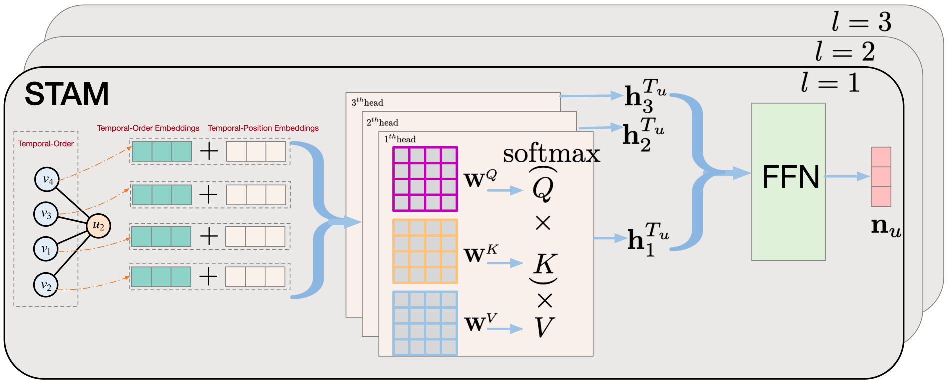

Figure 2: The overall architecture of STAM.

3.1 STAM

To incorporate temporal information into aggregation methods for GNN-based recommendation models, we design a novel spatiotemporal aggregation method (STAM) for neighbor embedding learning. Figure 2 depicts the overall architecture of STAM, which takes temporal orders T_u and T_v as input and outputs spatiotemporal neighbor embeddings generated from one-hop neighbors. Specifically, we firstly construct two critical temporal orders from the connected one-hop neighbors for each user-item pair (u, v), consisting of user's temporal order $T_u = \{v_1, \cdots, v_S\}$ and item's temporal order $T_v = \{u_1, \cdots, u_S\}$ where S is the length of temporal order. Then, the look-up operation is performed to obtain the initial embeddings for each temporal order, i.e., temporal-order embeddings $X_u = \{\mathbf{e}_v^1, \mathbf{e}_v^2, \cdots, \mathbf{e}_v^S\}$ and $X_v = \{\mathbf{e}_u^1, \mathbf{e}_u^2, \cdots, \mathbf{e}_u^S\}$, $\mathbf{e}_u^t/\mathbf{e}_v^t \in \mathbb{R}^d$ with dimensionality d. Note that the inital embeddings $\mathbf{e}_u^t$ and $\mathbf{e}_v^t$ are obtained from the *Embedding Layer* mentioned in Section 2.1.

The foremost objective of STAM is to learn spatiotemporal neighbor embeddings from one-hop neighbors for each user-item pair (u, v). To achieve this objective, we utilize the *Scaled Dot-Product Attention* [43] which have become a critical part of sequence modeling, taking the queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$ as the input representations. The queries, keys and values are projected into different spaces through linear projection matrices $\mathbf{W}_Q \in \mathbb{R}^{d \times D'}$, $\mathbf{W}_K \in \mathbb{R}^{d \times D'}$ and $\mathbf{W}_V \in \mathbb{R}^{d \times D'}$, respectively. Moreover, we also employ the positional encoding [10] to encode temporal information into the *Scaled Dot-Product Attention*, i.e., temporal orders T_u and T_v are equipped with absolute temporal-position embeddings $P_u = \{\mathbf{p}_v^1, \mathbf{p}_v^2, \cdots, \mathbf{p}_v^S\}$ and $P_v = \{\mathbf{p}_u^1, \mathbf{p}_u^2, \cdots, \mathbf{p}_u^S\}$ respectively, where $\mathbf{p}_u^t/\mathbf{p}_v^t \in \mathbb{R}^d$ denotes the positional vectors at position t. The temporal-position embeddings P_u/P_v are combined with the temporal-order embeddings X_u/X_v to obtain user's and item's temporal input embeddings $Z_u = \{\mathbf{e}_v^1 + \mathbf{p}_v^1, \mathbf{e}_v^2 + \mathbf{p}_v^2, \cdots, \mathbf{e}_v^S + \mathbf{p}_v^S\}$ and $Z_v = \{\mathbf{e}_u^1 + \mathbf{p}_u^1, \mathbf{e}_u^2 + \mathbf{p}_u^2, \cdots, \mathbf{e}_u^S + \mathbf{p}_u^S\}$, respectively. Here, we pack the temporal input embeddings Z_u/Z_v together into matrices $\mathbf{Z}_u \in \mathbb{R}^{S \times d}$ and $\mathbf{Z}_v \in \mathbb{R}^{S \times d}$ respectively, which are feed as input to STAM for spatiotemporal neighbor embedding learning. Thus, take user's temporal order T_u as an example, the *Scaled Dot-Product Attention* function is formulated as:

$$\mathbf{h}^{T_u} = \text{softmax}\left(\frac{(\mathbf{Z}_u\mathbf{W}_Q)(\mathbf{Z}_u\mathbf{W}_K)^T}{\sqrt{D'}}\right)(\mathbf{Z}_u\mathbf{W}_V) \tag{8}$$

where $\mathbf{Z}_u$ is a temporal input embedding matrix, $\mathbf{h}^{T_u} \in \mathbb{R}^{S \times D'}$ denotes the output embedding matrix for one-hop neighbors, $\mathbf{W}_Q \in \mathbb{R}^{d \times D'}$, $\mathbf{W}_K \in \mathbb{R}^{d \times D'}$ and $\mathbf{W}_V \in \mathbb{R}^{d \times D'}$ are shared weight transformations applied to each user-item pair (u, v).

Similarly, the output embedding matrix $\mathbf{h}^{T_v} = \{\mathbf{h}_u^1, \mathbf{h}_u^2, \cdots, \mathbf{h}_u^S\}$ for item's temporal order T_v can also be represented as:

$$\mathbf{h}^{T_v} = \text{softmax}\left(\frac{(\mathbf{Z}_v\mathbf{W}_Q)(\mathbf{Z}_v\mathbf{W}_K)^T}{\sqrt{D'}}\right)(\mathbf{Z}_v\mathbf{W}_V) \tag{9}$$

To further improve the expressivity of STAM, we adopt *Multi-Head Attention* instead of performing a single attention function to capture temporal information from different latent perspectives. Multi-head attention has been applied in previous works [7, 27, 43], which jointly focuses on information from different representation subspaces. Such an attention technique is widely utilized to improve the diversity of attention mechanisms by leveraging multiple independent attention heads that operate on the input embedding along with different, learnable linear projection matrices. Besides, Multi-head attention also promotes the capacity and stability of STAM. To be specific, multi-head attention firstly projects the temporal input embeddings $\mathbf{Z}_u/\mathbf{Z}_v$ into k subspaces with a variety of linear projection matrices and then employs k *Scaled Dot-Product Attention* functions in parallel to generate the output embedding matrix for one-hop neighbors. These embeding matrix can be concatenated to produce a combined neighbor embedding matrix. Lastly, we apply a feed-forward neural network for dimensionality transformation. The spatiotemporal neighbor embeddings for the central user u and the central item v can be calculated as:

$$\begin{aligned} \mathbf{h}_u &= \text{FFN}(\mathbf{h}_1^{T_u} \cdots ||\mathbf{h}_i^{T_u} \cdots ||\mathbf{h}_k^{T_u}), \\ \mathbf{h}_v &= \text{FFN}(\mathbf{h}_1^{T_v} \cdots ||\mathbf{h}_i^{T_v} \cdots ||\mathbf{h}_k^{T_v}). \end{aligned} \tag{10}$$

where $\mathbf{h}_u \in \mathbb{R}^{S \times d}$ and $\mathbf{h}_v \in \mathbb{R}^{S \times d}$ denote spatiotemporal neighbor embeddings generated by multi-head attention, $\mathbf{h}_i^{T_u}/\mathbf{h}_i^{T_v} \in \mathbb{R}^{S \times D'/k}$ represents the output embedding matrix for one-hop neighbors from i-th *Scaled Dot-Product Attention* function. FFN$(\cdot)$ can be represented as FFN$(\mathbf{x}) = \mathbf{x}\mathbf{W}_0 + \mathbf{b}_0$, where $\mathbf{W}_0 \in \mathbb{R}^{D' \times d}$ and $\mathbf{b}_0 \in \mathbb{R}^d$ are trainable parameters. Note that spatiotemporal neighbor embeddings can also be represented in the form of temporal orders, such as $\mathbf{h}_u = \{\mathbf{h}_{v_1}, \cdots \mathbf{h}_{v_S}\}$ and $\mathbf{h}_v = \{\mathbf{h}_{u_1}, \cdots, \mathbf{h}_{u_S}\}$.

Thereafter, we calculate the aggregated neighbor embedding from the abovementioned spatiotemporal neighbor embeddings.

Here, we simply utilize the mean-pooling operation to aggregate spatiotemporal neighbor embeddings. In our experiments, we find that the mean-pooling for spatiotemporal neighbor embeddings leads to a good performance in general. The aggregated neighbor embedding $\mathbf{n}_u/\mathbf{n}_v$ for the central user/item can be formulated as:

$$\mathbf{n}_u = \frac{1}{S}\sum_{i=1}^{S}\mathbf{h}_{v_i},\ \mathbf{n}_v = \frac{1}{S}\sum_{i=1}^{S}\mathbf{h}_{u_i}. \tag{11}$$

3.2 STAM for GNN-based Recommendation

In Section 3.1, we design a spatiotemporal aggregation method (STAM) to simultaneously learn neighbor embeddings from the perspective of spatial structure and temporal order. Here, we utilize the proposed STAM for GNN-based recommendation to learn users and items embeddings. Like many existing GNN-based recommendation models elaborated in Section 2.1, we also stack multiple STAM to capture higher-order interactions in the user-item bipartite graph. Similar to LightGCN, a state-of-the-art GNN-based recommendation model, we also omit non-linear transformation and aggregate spatiotemporal neighbor embeddings as the central node embedding.

However, the abovementioned STAM will suffer from the exponentially increasing memory consumption when propagating the spatiotemporal neighbor embeddings from the one-hop neighbors layer-by-layer. Here, we learn a spatiotemporal attention weight matrix $\Phi \in \mathcal{R}^{(M+N)\times(M+N)}$ from the spatiotemporal neighbor embeddings $\mathbf{h}_u/\mathbf{h}_v$, and integrate it to adjacent matrix $\mathbf{A} \in \mathbb{R}^{(M+N)\times(M+N)}$. Mathematically, the matrix form of the spatiotemporal propagation layer can be formulated as:

$$\mathbf{h}^{(l+1)} = (\mathbf{D}^{-\frac{1}{2}}(\mathbf{A}\odot\Phi)\mathbf{D}^{-\frac{1}{2}})\mathbf{h}^{(l)} \tag{12}$$

where $\mathbf{h}^{(l+1)}$ is the central embedding in $(l+1)$-th layer, $\odot$ denotes the element-wise product, $\mathbf{D}$ is the diagonal matrix, in which each entry D_{ii} denotes the number of nonzero entries in the i-th row vector of the adjacency matrix $\mathbf{A}$.

The final embeddings for the central user and item can be computed by weighted-pooling operation. Specifically, the pooling function is applied to generate the final user/item embeddings $\mathbf{e}_u^*/\mathbf{e}_v^*$ by operating on propagated L layers:

$$\mathbf{e}_u^* = \sum_{l=0}^{L}\alpha_l\mathbf{h}_u^{(l)}, \mathbf{e}_v^* = \sum_{l=0}^{L}\alpha_l\mathbf{h}_v^{(l)} \tag{13}$$

where $\alpha_l \geq 0$ denotes the importance of the l-th layer representation in constituting the final embedding. Similar to LightGCN [17], we uniformly set α_l as $1/(L+1)$ since the concentration of this work is not on the choice of α_l.

3.3 Optimization with STAM

To optimize the parameters of STAM, we use a widely-adopted BPR loss [34]:

$$\mathcal{L} = -\sum_{\substack{(u,v)\in\mathcal{D}\\ v_n\sim p_n(\cdot|u)}} \ln\sigma(\mathbf{e}_u^{*\top}\mathbf{e}_v^* - \mathbf{e}_u^{*\top}\mathbf{e}_{v_n}^*) + \lambda||\Theta||_2^2 \tag{14}$$

where $\sigma(\cdot)$ is the sigmoid function, Θ is trainable parameters, and λ controls the L_2 regularization strength. $v_n \sim p_n(\cdot|u)$ represents the negative sampling strategy.

3.4 Model Analysis

In this work, we propose STAM to aggregate one-hop neighbors by integrating temporal information and stack multiple STAM to capture higher-order interactions in the user-item graph for GNN-based recommendation. STAM can be directly plugged into existing GNN-based recommendation models by substituting the default aggregation method. Here, we present the connection and discrimination of STAM with previous related works.

- **STAM vs LightGCN.** LightGCN [17] is a recent representative graph convolutional network model for recommendation, which learns user and item embeddings by *linearly aggregating* their neighbors in the user-item interaction graph. Both STAM and LightGCN aim to learn fine-grained user and item embeddings for GNN-based recommendation. Unlike LightGCN that only capture spatial structural information, we support to *simultaneously capture spatial and temporal information* by leveraging STAM.
- **STAM vs SASRec.** SASRec [22] is a recent variant of Transformer [43] that uses a set of trainable position embeddings to encode the order of items in sequence for sequential recommendation. Although STAM and SASRec both utilize temporal information to learn user and item embeddings, the types of recommendation are different. Different from SASRec for sequence recommendation, STAM focuses on GNN-based recommendation. Besides, item embeddings in SASRec are performed from *an embedding look-up table*, while in STAM are iteratively learned by *the spatiotemporal aggregation method*.
- **STAM vs BERT4Rec.** BERT4Rec [40] is a sequential recommendation model that uses deep bidirectional self-attention to model user behavior sequences. Despite STAM and BERT4Rec both apply *Scaled Dot-Product Attention* to model temporal information, STAM focuses on improving the aggregation method by simultaneously aggregating one-hop neighbor's information from the perspectives of spatial structure and temporal order. Different targets lead to different training methods. BERT4Rec predicts the masked items using *Cloze objective*, while STAM directly conducts BPR loss *on the sampled training pairs*.
- **STAM vs DySAT.** DySAT [36] is a novel neural architecture to capture dynamic graph structural evolution. DySAT models the dynamic graph structure into a sequence of graph snapshots. However, temporal information in STAM is collected by user's behavior history. Different from DySAT that computes node representations by *stacking structural and temporal self-attentional layers*, STAM utilizes the proposed *spatiotemporal aggregation method* to simultaneously learn spatiotemporal neighbor embeddings from the perspectives of spatial structure and temporal order.

3.5 Time Complexity

As many existing GNN-based recommendation models, the spatiotemporal aggregation operations on the user-item graph are the main time cost. Since the multi-head attention used in the proposed STAM is parallelizable, the time complexity for STAM with k heads can be expressed as $O(kS)$ where S is the length of temporal input T_u/T_v.

4 EXPERIMENTS

In this section, we present the details of experiment setups and the corresponding experimental results of STAM plugged into GNN-based recommendation in comparison with the state-of-the-art recommendation models. We then conduct the ablation studies, especially in comparison with previous spatial-based aggregation methods. Finally, we analyze the impact of the hyperparameters.

4.1 Experimental Setups

4.1.1 ***Datasets.*** We conduct experiments on three widely-used datasets collected from real-world platforms with different densities. Table 1 shows the statistics of all datasets. Introductions about the datasets are in Appendix A.1. Moreover, we also describe the dataset splitting method for STAM in Appendix A.2.

Table 1: The statistics of the experimental datasets.

Dataset	User	Item	Interactions	Density	Avg.length
MovieLens	6,040	3,416	999,611	4.362%	165.5
Amazon	158,650	128,939	4,701,968	0.021%	29.6
Taobao	196,840	285,869	24,938,383	0.040%	126.7

4.1.2 ***Evaluation Metrics.*** We evaluate all the models with Mean Reciprocal Rank (MRR), Normalized Discounted Cumulative Gain (NDCG), and Hit Ratio (HR), which are widely-used evaluation protocols [18, 54] in top-K recommendation. We report the average metrics for all users in the test set and compute the metrics by *ranking all items that are not interacted by a user.* In our experiments, the length of recommended list K is set to 20 and 50. The detailed evaluation protocol is provided in Appendix A.3.

4.1.3 ***Baselines.*** To demonstrate the effectiveness of STAM, we utilize STAM for GNN-based recommendation and conduct comparative experiments with different kinds of recommendation models, including traditional (MostPopular, BPRMF), Neural Network-based (NeuMF), and GNN-based (GC-MC, PinSage, NGCF, and LightGCN). Besides, we also evaluate the performance of STAM in comparison with four representative sequential recommendation models, including GRU4Rec, Caser, SASRec, and BERT4Rec. A detailed description of each baseline is provided in Appendix A.4.

4.2 Performance Comparison

4.2.1 ***Comparison with Representative Models.*** We utilize STAM for GNN-based recommendation and compare its performance with three types of representative models in Table 2. In general, STAM achieves a significant improvement over baselines, which confirms our claim that the significance of temporal information for neighbor aggregation in GNN-based recommendation. Besides, we elaborate exhaustive observations as follows:

- The traditional recommendation models (MostPop and BPRMF) achieve poor performance in all cases. Compared with MostPop, BPRMF shows better performance. The reason is that BPRMF utilizes inner product to model user-item interactions with latent features, while MostPop only utilizes the property of the dataset for recommendation.
- Obviously, NeuMF consistently outperforms traditional models across all datasets, demonstrating the importance of nonlinear feature interactions between user and item embeddings.
- GNN-based models utilize GNN to learn embeddings for users and items, and achieve superior performance over the above baselines in most cases. Different from GC-MC that only adopts the mean-pooling to aggregate first-order neighbors, NGCF and LightGCN exploit message propagation through the graph structure and achieve significant improvements in performance. LightGCN outperforms both the GNN-based models, which might be attributed to the discarding of non-linear transformation. Instead of operating on the full graph, PinSage employs neighborhood sampling to sample the fixed-size neighbors for aggregation. Such sampling strategy may result in performance degradation due to the sacrifice of a part of graph information.
- Compared to other GNN-based recommendation models that do not integrate temporal information into aggregation, STAM utilized for GNN-based recommendation achieves the best performance on three datasets and all metrics. STAM is a universal spatiotemporal aggregation method that can be naturally plugged into existing GNN-based recommendation models. Here, we summarize some reasons that contribute to the improvement in performance: (1) it introduces temporal information to simultaneously model neighbor embedding learning from the perspectives of spatial structure and temporal order; (2) it utilizes a powerful attention mechanism (Scaled Dot-Product Attention) to capture temporal order from one-hop neighbors and employs multi-head attention to improve the expressivity; (3) it also uses message propagation mechanism to capture higher-order user-item interactions by stacking multiple STAM.
- As demonstrated in Table 2, STAM plugged with GNN-based recommendation accomplishes the significant performance gains of 24.32% on the MoiveLens dataset, 7.78% on the Amazon dataset, and 12.5% on the Taobao dataset in terms of $MRR@20$ metric against the strongest baselines. Such improvement enables us to integrate temporal information into aggregation to benefit users and items embedding learning for GNN-based recommendation.

4.2.2 ***Comparison with Sequential Models.*** Here, we also conduct experiments to compare STAM with four representative sequential models, including GRU4Rec, Caser, SASRec, and BERT4Rec. Table 3 presents the overall performance of STAM and the adopted baselines, from which we have the following observations:

- SASRec outperforms both RNN-based model GRU4Rec and CNN-based model Caser in most cases, which confirms the superiority of self-attention mechanism to model temporal order. By comparing BERT4Rec with SASRec, we find the superiority of a bidirectional model for temporal order. In STAM, we thus utilize *Scaled Dot-Product Attention* to capture the temporal order of one-hop neighbors and learn spatiotemporal neighbor embeddings from the perspectives of spatial structure and temporal order.
- According to the results demonstrated in Table 3, STAM outperforms the best baseline BERT4Rec in most cases while it performs worse than BERT4Rec in some evaluation metrics. Despite STAM and BERT4Rec both applying Scaled Dot-Product Attention to model temporal information, STAM only utilizes one layer of Attention to capture temporal information in each propagation layer while BERT4Rec stacks multiple Attention layers to learn more complex item transition patterns. In general, STAM accomplishes better performance than BERT4Rec. Such improvement

Table 2: Performance comparison between STAM and a variety of baselines. All the numbers in the table are percentage numbers with '%' omitted.

Methods	MovieLens						Amazon						Taobao					
	Metrics@20			Metrics@50			Metrics@20			Metrics@50			Metrics@20			Metrics@50		
	MRR	NDCG	HR	MRR	NDCG	HR	MRR	NDCG	HR	MRR	NDCG	HR	MRR	NDCG	HR	MRR	NDCG	HR
MostPop	0.81	5.05	45.00	0.99	7.26	67.98	0.15	0.41	2.67	0.17	0.62	5.41	0.10	0.61	9.80	0.11	0.78	18.07
BPRMF	1.10	6.75	57.73	1.30	9.13	80.58	0.37	0.99	6.39	0.42	1.44	11.57	0.09	0.58	10.03	0.10	0.82	20.31
NeuMF	1.79	7.96	65.96	2.07	9.15	85.40	0.47	1.24	7.69	0.53	1.85	14.32	0.15	0.96	14.73	0.17	1.24	26.60
GC-MC	1.67	10.22	64.67	1.80	11.96	82.81	0.67	1.71	9.97	0.75	2.41	17.18	0.14	1.01	15.24	0.17	1.39	28.31
PinSage	1.73	11.10	65.28	2.03	14.26	83.81	0.60	1.55	9.25	0.67	2.33	17.23	0.14	0.91	14.09	0.16	1.25	26.67
NGCF	1.75	10.78	66.82	2.04	13.58	85.38	0.74	1.87	10.75	0.83	2.71	19.30	0.21	1.34	19.89	0.24	1.76	34.17
LightGCN	1.85	11.00	66.97	2.14	13.80	86.28	0.90	2.29	12.65	1.00	3.31	22.17	0.24	1.51	21.18	0.27	1.97	36.12
STAM	**2.30**	**13.86**	**71.25**	**2.61**	**16.86**	**89.04**	**0.97**	**2.46**	**13.17**	**1.09**	**3.54**	**23.25**	**0.27**	**1.65**	**22.58**	**0.31**	**2.08**	**37.52**
%Improv.	24.32%	26.00%	6.39%	21.96%	22.17%	3.20%	7.78%	7.42%	4.11%	9.00%	6.04%	4.87%	12.5%	9.27%	6.61%	14.81%	5.58%	3.88%

Table 3: Results of STAM with sequential models.

Methods	MovieLens			Amazon			Taobao		
	MRR	NDCG	HR	MRR	NDCG	HR	MRR	NDCG	HR
GRU4Rec	2.23	11.33	66.23	0.54	1.37	8.67	0.17	1.14	16.80
Caser	2.26	11.58	67.91	0.45	1.21	7.99	0.18	1.29	18.11
SASRec	2.13	11.18	68.29	0.81	2.13	12.81	0.22	1.45	20.69
BERT4Rec	**2.38**	12.30	71.02	0.89	2.42	**13.46**	0.25	1.57	21.93
STAM	2.30	**13.86**	**71.25**	**0.97**	**2.46**	13.17	**0.27**	**1.65**	**22.58**

might be attributed to the spatial graph structure, which can utilize message propagation to propagate embeddings in the user-item graph.

4.3 Ablation Study

We first conduct an ablation study to verify the crucial role of temporal information in GNN-based recommendation, and then explore the impact of propagation layer numbers and input lengths.

4.3.1 ***STAM vs Previous Aggregation Methods.*** To verify the superiority of STAM, we compare STAM with previous four representative spatial-based aggregation methods, including "mean pooling", "attentive pooling", "degree normalization", and "central node augmentation". The detailed descriptions of spatial-based aggregation methods are presented in Appendix A.5. Specifically, we utilize STAM for GNN-based recommendation and substitue STAM with the abovementioned spatial-based aggregation methods for comparative experiments. As depicted in Figure 3, STAM is obviously superior to all spatial-based aggregation methods, verifying that capturing temporal order of one-hop neighbors is advantageous for neighbor embedding learning. Moreover, we also conduct a critical comparison between STAM and LSTM aggregator proposed in GraphSAGE [14]. We adopt BiLSTM [12] to replace LSTM [13, 39] for improving the expressiveness of aggregator. Compared to the BiLSTM aggregator, STAM achieves better performance which suggests that the self-attention mechanism is a more powerful technology for temporal order learning.

4.3.2 ***Impact of Layer Numbers.*** To analyze the impact of the propagation layer numbers, we vary the number of propagation layers L in the range of $\{1, 2, 3, 4\}$. As presented in Table 4, STAM utilized for GNN-based recommendation benefits from stacking multiple STAM to propagate spatiotemporal neighbor embedding on the user-item graph. Similar to many GNN-based recommendation models, stacking too many STAM also bring about the oversmoothness issue where the performance demonstrates a peak variation with the increasing of the number of propagation layers.

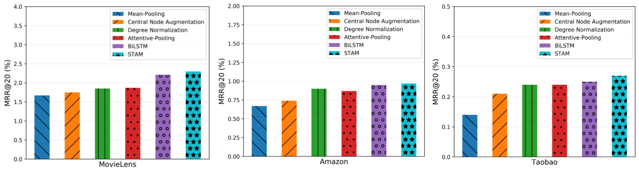

Figure 3: Results between STAM and previous aggregations.

Table 4: Results comparison of STAM at different layers.

# Layers	MovieLens			Amazon			Taobao		
	MRR	NDCG	HR	MRR	NDCG	HR	MRR	NDCG	HR
1 Layer	2.17	13.59	70.73	0.89	2.28	12.46	0.23	1.47	22.29
2 Layer	2.18	13.61	72.14	0.93	2.34	12.95	0.25	1.62	22.47
3 Layer	2.30	13.86	71.25	0.97	2.46	13.17	0.27	1.65	22.58
4 Layer	2.15	13.45	70.68	0.90	2.28	13.01	0.25	1.61	22.37

4.3.3 ***Impact of Input Lengths.*** We conduct an experiment to analyze the impact of input length S and represent the experimental results on *MRR*@20 in Figure 4. Specifically, we search S in the range of $\{5, 20, 50, 100, 150, 200\}$, and keep the best setting (S = 200 for MovieLens and S = 50 for Amazon and S = 100 Taobao datasets, respectively) in our experiments. From Figure 4, it is clearly observed that the performance of STAM fluctuates slightly with the number of input lengths. This indicates that spatiotemporal neighbor embeddings generated by STAM are slightly affected by the temporal length of one-hop neighbors. Moreover, we observe that the proper input length S is highly dependent on the average length of one-hop neighbors of the dataset, which enables us to set the optimal input length from the property of the dataset.

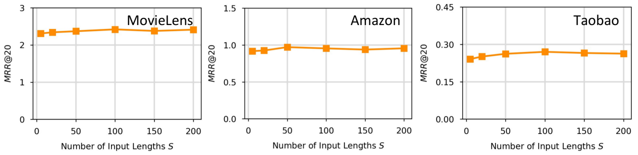

Figure 4: Analyzing the impact of input lengths S on GNN-based recommendation plugged with STAM.

4.4 Parameter Sensitivity

To test the robustness of STAM, we visualize the $MRR@20$ curves by varying the two most important hyperparameters: the number of heads k for multi-head attention and the hidden dimensionality d. Figure 5 illustrates the performance of GNN-based recommendation utilized with STAM on the MovieLens dataset.

4.4.1 ***Multi-head attention.*** To analyze the benefits of multi-head attention, we vary the number of attention heads k independently in the range of $\{1, 2, 4, 8, 16\}$. In general, STAM benefits from multi-head attention for spatiotemporal neighbor embedding learning. The optimal performance stabilizes with 4 attention heads, which implies that STAM can sufficiently capture temporal order of one-hop neighbors from different latent subspaces. Overall, multi-head attention plays a beneficial role in the expressivity of STAM.

4.4.2 ***Hidden Dimensionality*** *d.* We now study the impact of hidden dimensionality on GNN-based recommendation plugged with STAM. We vary the hidden dimensionality from 16 to 256, while keeping optimal hyperparameters fixed for fairness. The obvious observation is that the performance tends to converge with the increase of dimensionality. Intuitively, a larger dimensionality will result in better performance but a larger dimensionality leads to a longer training time. Therefore, we need to find a proper dimensionality to balance the trade-off between performance and time consumption. In our experiments, we set the hidden dimensionality as 64 for STAM and a variety of baselines.

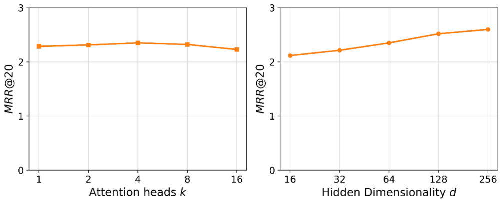

Figure 5: The performance of STAM on the MovieLens dataset by varying attention heads and hidden dimensions.

5 RELATED WORK

5.1 GNN-based Recommendation

Recent years have witnessed the tremendous success of graph neural networks (GNNs) in recommendation systems, showing significant performance improvements and boosting web-scale applications. Graph neural network is able to capture the higher-order interaction in the user-item graph through iterative propagation. GC-MC [1] applies the graph convolution network (GCN) [24] on the user-item graph, which models the direct connections between users and items via one convolutional layer and doesn't fully exploit the message passing through the graph structure. STAR-GCN [57] stacks identical GCN blocks (i.e. GC-MC) instead of directly staking multi-layer GCNs, which alleviates the over-smoothness issue and results in better performance compared to GC-MC. NGCF [48] exploits the user-item graph structure by propagating embeddings on it, which leads to the expressive modeling of high-order connectivity in the user-item graph. Recently, SGCN [49] argues the unnecessary complexity of GCN, which simplifies GCN by removing nonlinearities and collapsing multiple weight matrices to one weight matrix. Inspired by the design of SGCN, LightGCN [17] simplifies the design of GCN to make it more concise and appropriate for recommendation, which learns user and item embeddings by linearly propagating them on the user-item interaction graph and uses the weighted sum of the embeddings learned at all layers as the final embedding. The above methods apply GNN over the full graph without neighborhood sampling, which preserves the original graph structure but prevents web-scale applications due to low generalization power. PinSage [56] combines random-walk based sampling strategy and graph convolutions to learn the embeddings on an item-item graph for Pinterest image recommendation.

5.2 Sequential Recommendation

Sequential recommendation captures sequential patterns among sucessive items and recommends what the users might click next [4, 20, 29, 35, 40, 47]. Some works adopt Markov Chain (MC) [15, 35] to capture the item-to-item transition based on the assumption that the most recent clicked item reflects the user's dynamic preference. For instance, FPMC [35] fuses an MF term and an item-item transition term to capture long-term preferences and short-term transitions respectively. Owing to the advantage of Recurrent Neural Network (RNN) in sequence modeling, several works employ RNN to model user sequence patterns [19, 20, 41]. GRU4Rec [20] utilizes Gated Recurrent Units (GRU) [5] to model click sequences for session-based recommendation. Besides, Caser [42], a CNN-based method, uses convolutional operations on the embedding matrix of L most recent items to capture high-order Markov chains. Recently, a new sequential model Transformer achieves state-of-the-art performance on NLP tasks [43], and its proposed attention mechanism provides a new method for sequential recommendation [26, 28]. SASRec [22] employs self-attention technique to model sequential patterns and captures long-term semantics. BERT4Rec [40] uses a deep bidirectional sequential model for sequential recommendation. Besides, some studies have attempted to infer the dynamic user preferences with sequential user interactions [2, 30, 33, 45]. With the emerging of GNNs, some works transform sequence data into the sequence graphs and conduct message propagation on such graphs to model dynamic user preference [3, 51, 53].

6 CONCLUSION

In this paper, we propose a universal spatiotemporal aggregation method named STAM to learn spatiotemporal neighbor embeddings for neighbor embedding learning. Specifically, STAM utilizes the Scaled Dot-Product Attention to capture temporal orders of one-hop neighbors and employs multi-head attention to perform joint attention over different latent subspaces. We stack multiple STAM for GNN-based recommendation to learn users and items embeddings, in which STAM does not alter the framework of GNN-based recommendation. Experimental results on three widely-used datasets demonstrate that STAM brings significant improvements on GNN-based recommendation compared with spatial-based aggregation methods, which can be plugged into GNN-based recommendation.

ACKNOWLEDGEMENTS

The work is supported by the NSFC for Distinguished Young Scholar (61825602), NSFC (61836013), and a research fund supported by Alibaba Group.

REFERENCES

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[2] Yukuo Cen, Jianwei Zhang, Xu Zou, Chang Zhou, Hongxia Yang, and Jie Tang. 2020. Controllable Multi-Interest Framework for Recommendation. In *KDD'20*. 2942–2951.

[3] Tianwen Chen and Raymond Chi-Wing Wong. 2020. Handling Information Loss of Graph Neural Networks for Session-based Recommendation. In *KDD'20*. 1172–1180.

[4] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *WSDM'18*. 108–116.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *RecSys'16*. 191–198.

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

[8] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative memory network for recommendation systems. In *SIGIR'18*. 515–524.

[9] Cédric Févotte and Jérôme Idier. 2011. Algorithms for nonnegative matrix factorization with the β-divergence. *Neural computation* 23, 9 (2011), 2421–2456.

[10] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*. PMLR, 1243–1252.

[11] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 249–256.

[12] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 6645–6649.

[13] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks* 18, 5-6 (2005), 602–610.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS'17*. 1024–1034.

[15] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM'16*. IEEE, 191–200.

[16] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW'16*. 507–517.

[17] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *SIGIR'20* (2020).

[18] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW'17*. 173–182.

[19] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *CIKM'18*. 843–852.

[20] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[21] Santosh Kabbur, Xia Ning, and George Karypis. 2013. Fism: factored item similarity models for top-n recommender systems. In *KDD'13*. 659–667.

[22] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM'18*. IEEE, 197–206.

[23] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[24] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[26] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *CIKM'17*. 1419–1428.

[27] Jian Li, Zhaopeng Tu, Baosong Yang, Michael R Lyu, and Tong Zhang. 2018. Multi-head attention with disagreement regularization. *arXiv preprint arXiv:1810.10183* (2018).

[28] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: short-term attention/memory priority model for session-based recommendation. In *KDD'18*. 1831–1839.

[29] Fuyu Lv, Taiwei Jin, Changlong Yu, Fei Sun, Quan Lin, Keping Yang, and Wilfred Ng. 2019. SDM: Sequential deep matching model for online large-scale recommender system. In *CIKM'19*. 2635–2643.

[30] Chen Ma, Peng Kang, and Xue Liu. 2019. Hierarchical gating networks for sequential recommendation. In *KDD'19*. 825–833.

[31] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR'15*. 43–52.

[32] Andriy Mnih and Russ R Salakhutdinov. 2007. Probabilistic matrix factorization. *NIPS'07* 20 (2007), 1257–1264.

[33] Ruihong Qiu, Jingjing Li, Zi Huang, and Hongzhi Yin. 2019. Rethinking the item order in session-based recommendation with graph neural networks. In *CIKM'19*. 579–588.

[34] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[35] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW'20*. 811–820.

[36] 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.

[37] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW'01*. 285–295.

[38] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 291–324.

[39] Jürgen Schmidhuber and Sepp Hochreiter. 1997. Long short-term memory. *Neural Comput* 9, 8 (1997), 1735–1780.

[40] 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'19*. 1441–1450.

[41] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *DLRS'16*. 17–22.

[42] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM'18*. 565–573.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS'17* (2017).

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *ICLR'18* (2017).

[45] Jianling Wang, Kaize Ding, Liangjie Hong, Huan Liu, and James Caverlee. 2020. Next-item recommendation with sequential hypergraphs. In *SIGIR'20*. 1101–1110.

[46] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR'17*. 515–524.

[47] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning hierarchical representation model for nextbasket recommendation. In *SIGIR'15*. 403–412.

[48] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR'19*. 165–174.

[49] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153* (2019).

[50] Le Wu, Yonghui Yang, Lei Chen, Defu Lian, Richang Hong, and Meng Wang. 2020. Learning to Transfer Graph Embeddings for Inductive Graph based Recommendation. In *SIGIR'20*. 1211–1220.

[51] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *AAAI'19*, Vol. 33. 346–353.

[52] Shiwen Wu, Wentao Zhang, Fei Sun, and Bin Cui. 2020. Graph Neural Networks in Recommender Systems: A Survey. *arXiv preprint arXiv:2011.02260* (2020).

[53] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph Contextualized Self-Attention Network for Session-based Recommendation.. In *IJCAI'19*. 3940–3946.

[54] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: high-order proximity for implicit recommendation. In *RecSys'18*. 140–144.

[55] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. 2020. Understanding Negative Sampling in Graph Representation Learning. *KDD'20* (2020).

[56] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD'18*. 974–983.

[57] Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. 2019. Star-gcn: Stacked and reconstructed graph convolutional networks for recommender systems. *arXiv preprint arXiv:1905.13129* (2019).

[58] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR'13*. 785–788.

[59] Chuanchuan Zhao, Jinguo You, Xinxian Wen, and Xiaowu Li. 2020. Deep Bi-LSTM Networks for Sequential Recommendation. *Entropy* 22, 8 (2020), 870.

[60] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *KDD'18*. 1079–1088.

A APPENDIX

In the appendix, we first report the implementation notes of STAM. Then, we present datasets, dataset splitting, and evaluation metrics in detail. Next, a detailed description of each baseline is provided. Finally, we introduce five previous representative aggregation methods, including five spatial-based aggregators and a BiLSTM aggregator.

Implementation Details. The parameters are optimized via extensive grid search on all the datasets, and early stopping is applied to select the best models by the MRR@20 score on the validation set. For all models, We use Adam optimizer [23] with learning rate $lr = 0.001$ for training. The user and item embedding dimension is set as 64 to achieve the trade-off between the performance and time consumption, which is determined by grid search in the range of $\{16, 32, 64, 128, 256\}$. The batch size for the MoiveLens and Amazon datasets and the Taobao dataset is set to 1024 and 2048, respectively. The number of hidden units for BiLSTM is set to 32. For STAM, we set the number of attention heads as $k = 4$. The coefficient of L2 regularization is searched in $\{10^{-5}, 10^{-4}, \cdots, 10^{-1}\}$. We use the Xavier initializer [11] to initialize the model parameters. The two aggregation layers with a fixed number of neighbors are applied for PinSage and the number of propagation layers is set to 3 for NGCF and LightGCN. In PinSage, the number of neighbors for two propagation layers is set to 25 and 10, respectively. The layer combination coefficient for LightGCN is uniformly set to $\frac{1}{1+L}$ where L is the number of propagation layers.

A.1 Datasets

We use three publicly available datasets to evaluate our proposed STAM, and the detailed descriptions are listed as follows.

- **MovieLens** is a widely used public movie rating dataset for both general and sequential recommendation. We use the ML-1M version of the MovieLens dataset and discard users and items with less than 5 related interactions.
- **Amazon**[*] is a collection of products with reviews and product metadata from Amazon [16, 31]. We conduct experiments on a subset named Books and filter these users and items that the number of interactions is shorter than 10.
- **Taobao**[†] is a dataset of user behaviors from the commercial platform of Taobao [60]. The dataset contains several types of user behaviors, including click, purchase, add-to-cart, and item favoring. In our experiment, we only use click behaviors.

A.2 Dataset Splitting

To incorporate temporal information into aggregation methods for GNN-based recommendation, we collect the temporal information from user's behavior history and build user's entire temporal sequence $\mathcal{T}_u$. Thereafter, we split those temporal sequences $\mathcal{T}_{u(s)}$ to the training, validation, and test dataset according to the timestamps. We hold the first 70% of interactions in each user's temporal sequence $\mathcal{T}_u$ as the training set and use the next 10% of interactions as the validation set to search the optimal hyperparameter settings for all methods. The remaining 20% interactions are used as the test set to evaluate the recommendation performance.

Thereafter, we introduce the establishment of user's and item's temporal order $T_u = \{v_1, \cdots, v_S\}/T_v = \{u_1, \cdots, u_S\}$ for training, where S is the number of one-hop neighbors. If the length of temporal sequence $\mathcal{T}_u$ is less than S, we repeatedly add a "padding" item to the left until the length is S. Otherwise, we truncate the temporal sequence $\mathcal{T}_u$ to the last S items and build a user's temporal order $T_u = \{v_1, ..., v_S\}$. Similarly, an item's temporal order $T_v = \{u_1, ..., u_S\}$ can also be constructed by the abovementioned operation.

A.3 Evaluation Protocol

A.3.1 *Evaluation Metrics.*

We define the recommended list for user u as $R_u = \{r_u^1, r_u^2, \cdots, r_u^K\}$, where r_u^i represents the ranked at the i-th position in R_u based on the predicted score. T_u is the set of u's interacted items in the test data.

- **MRR@K:** Mean Reciprocal Rank (MRR) is the average of reciprocal ranks of the correctly-recommended item t. The MRR metric considers the order of recommendation ranking, where a large MRR value indicates the correct recommendation at the top of the recommended list. The MRR is set to 0 when the rank of the correctly recommended item exceeds K.

$$MRR@K = \frac{1}{N} \sum_{t \in R_u} \frac{1}{Rank(t)} \tag{15}$$

  where N denotes the number of users in test data.

- **NDCG@K:** Normalized Discounted Cumulative Gain (NDCG) is a position-aware metric that assigns larger weights on higher positions.

$$NDCG@K = \frac{1}{Z} DCG@K = \frac{1}{Z} \sum_{j=1}^{K} \frac{2^{I(|r_u^j \cap T_u|)} - 1}{\log_2(j+1)} \tag{16}$$

  where $I(x)$ is the indicator function, and Z is a normalization constant denoting the ideal discounted cumulative gain (IDCG@K), which is the maximum possible value of DCG@K.

- **HR@K:** Hit Ratio gives the percentage of users that can receive at least one correct recommendation.

$$HR@K = \frac{1}{N} \sum_{u} I(|R_u \cap T_u|) \tag{17}$$

  where $I(x)$ is the indicator function.

A.3.2 *Evaluation Setup.*

For sequential models, we take test sequences as inputs and obtain the embeddings of test users based on the current model. Next, the item embeddings can be represented by the current model. Finally, we compute the metrics by ranking all items that are not interacted by a test user. For STAM, we obtain user and item embeddings by leveraging STAM to GNN-based recommendation, and then compute the metrics by ranking all items that are not interacted by a test user.

A.4 Baselines

We collect a variety of baselines from recommendation models, including traditional (MostPopular, BPRMF), Neural Network-based (NeuMF), Graph Neural Networks-based (GC-MC, PinSage, NGCF, and LightGCN), and sequential-based (GRU4Rec, Caser, SASRec, and BERT4Rec).

[*] http://jmcauley.ucsd.edu/data/amazon/

[†] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649&userId=1

- **MostPopular** is a non-personalized static method that recommends the top rank items based on popularity.
- **BPRMF** [34] is the classical matrix factorization method for item recommendation on implicit feedback data, which optimizes matrix factorization via a pairwise bayesian personalized ranking (BPR) loss.
- **NeuMF** [18] is a neural CF model, which uses multiple hidden layers to capture user-item nonlinear feature interactions.
- **GC-MC** [1] only takes one-hop neighbors into consideration, which models the first-order user-item interactions and ignores the original user and item representation itself.
- **PinSage** [56] utilizes a random-walk based sampling strategy to sample the fixed size of neighborhoods and propagates information via graph convolutions, which is scalable to a web-scale recommendation.
- **NGCF** [48] models multi-order connectivity in the user-item graph via message propagation and utilizes the residual network to get final node embedding from different layers.
- **LightGCN** [17] simplifies the GCN structure by removing the feature transformation and nonlinear activation, which only uses linear neighborhood aggregation and weighted sum of the embeddings at all layers as the final embedding.
- **GRU4Rec** [20] is the first work using RNNs to model sequential user behaviors for the session-based recommendation.
- **Caser** [42] proposes convolutional neural network based method to capture sequential structure of the L most recent items, and achieves better sequential recommendation performance.
- **SASRec** [22] uses self-attention mechanisms to identify the "relevant" items from a user's action history to predict the next item, where a set of trainable position embeddings is applied to encode the order of the items in a sequence.
- **BERT4Rec** [40] employs the deep bidirectional self-attention to model user behavior sequences and makes the recommendations to learn a bidirectional representation from both left and right sides. Besides, the Cloze task that predicts the masked items using both left and right context is used for model training.

A.5 Aggregation Methods

We collect five representative aggregation methods from previous GNN-based recommendation models and graph representation learning. There are two types of aggregation methods, consisting of spatial-based aggregation methods and temporal-based aggregation methods. We firstly review the spatial-based methods, including "mean pooling", "attentive pooling", "degree normalization", and "central node augmentation". We then introduce the BiLSTM aggregator utilized for our ablation study. Here, we take the central user as an example and learn the aggregated neighbor embedding $\mathbf{n}_u$.

- **Mean Pooling Aggregator.** As illustrated in many GNN-based recommendation models, mean-pooling aggregator treats the neighbors equally to reflect the user preference. The mean-pooling aggregator can be formulated as:

$$\mathbf{n}_u = \sigma(\sum_{v \in \mathcal{N}_u} \frac{1}{|\mathcal{N}_u|} \mathbf{e}_v) \tag{18}$$

  where $\sigma(\cdot)$ is the non-linear activation function, $\mathcal{N}_u$ is the neighbors of a given user u.

- **Attentive Pooling Aggregator.** As proposed in GAT [44], attentive pooling aggregator differentiates the importance of neighbors with attention mechanism and updates the embedding of each node (user and item) by attending over its neighbors. The attentive pooling aggregator is defined as:

$$\mathbf{n}_u = \sigma(\sum_{v \in \mathcal{N}_u} \alpha_{uv} \mathbf{W} \cdot \mathbf{e}_v) \tag{19}$$

  where α_{uv} is attention weights that be formulated as:

$$\alpha_{uv} = \frac{\exp(\text{LeakyReLU}(\mathbf{a} \cdot [\mathbf{W} \cdot \mathbf{e}_u || \mathbf{W} \cdot \mathbf{e}_v]))}{\sum_{i \in \mathcal{N}_u} \exp(\text{LeakyReLU}(\mathbf{a} \cdot [\mathbf{W} \cdot \mathbf{e}_u || \mathbf{W} \cdot \mathbf{e}_i)])} \tag{20}$$

  where $\mathbf{a}$ and $\mathbf{W}$ are trainable parameters.

- **Degree Normalization Aggregator.** Degree normalization aggregator assigns weights to nodes based on the graph structure. As demonstrated in LightGCN, it omits non-linear transformation where the aggregated neighbor embedding and assigns weights based on the graph structure that can be defined as:

$$\mathbf{n}_u = \sum_{v \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_v|}} \mathbf{e}_v \tag{21}$$

- **Central Node Augmentation Aggregator.** As shown in NGCF, it decides the aggregated neighbor embedding on the affinity between the central node. Specifically, NGCF employs element-wise product to augment the items' features the user cares about. Take the central user as an example:

$$\mathbf{n}_u = \sum_{v \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_v|}} (\mathbf{W}_1 \cdot \mathbf{e}_v + \mathbf{W}_2 \cdot (\mathbf{e}_u \odot \mathbf{e}_v)) \tag{22}$$

  where $\odot$ denotes element-wise multiplication operation.

- **BiLSTM Aggregator.** As proposed in GraphSAGE, LSTM aggregator is utilized to model temporal order of one-hop neighbors. However, vanilla LSTM only exploits the preceding or past information. Some works [40, 59] demonstrate that unidirectional models are sub-optimal and restrict the power of hidden representations, in which each item can only encode the information from previous items. Bidirectional LSTM (BiLSTM) [12] is an advancement of vanilla LSTM in which the forward hidden layer is combined with backward hidden layer, that can access both the preceding and past information. Thus, we employ BiLSTM to capture temporal order from the view of preceding and subsequent. BiLSTM takes a user's temporal order $T_u = \{v_1, v_2, \cdots, v_S\}$ as an input and computes the hidden state vector for each item:

$$\begin{aligned} \overrightarrow{\mathbf{h}_i} &= \overrightarrow{LSTM}(\vec{\mathbf{h}}_{i-1}, \mathbf{e}_{v_i}) \\ \overleftarrow{\mathbf{h}_i} &= \overleftarrow{LSTM}(\vec{\mathbf{h}}_{i+1}, \mathbf{e}_{v_i}) \end{aligned} \tag{23}$$

  We obtain the final hidden representation of i-th item by concatenating the hidden states from both directions,

$$\mathbf{h}_i = \overrightarrow{\mathbf{h}}_i || \overleftarrow{\mathbf{h}}_i \tag{24}$$

  Let $\mathbf{H}$ be a matrix consisting of output vectors $[\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_S]$ that the BiLSTM layer produced, where S is the input sequence length. The neighbor embedding can be formulated as:

$$\mathbf{n}_u = w^\top tanh(\mathbf{H}) \tag{25}$$

  where w is a trainable parameter vector.