Mathematical Biosciences and Engineering

MBE, 21(3): 3563–3593.
DOI: 10.3934/mbe.2024157
Received: 01 October 2023
Revised: 17 January 2024
Accepted: 29 January 2024
Published: 05 February 2024

http://www.aimspress.com/journal/MBE

Research article

DyCARS: A dynamic context-aware recommendation system

Zhiwen Hou, Fanliang Bu*, Yuchen Zhou, Lingbin Bu, Qiming Ma, Yifan Wang, Hanming Zhai and Zhuxuan Han

School of Information Network Security, People's Public Security University of China, Beijing 100038, China

* **Correspondence:** Email: bufanliang@sina.com.

Abstract: Dynamic recommendation systems aim to achieve real-time updates and dynamic migration of user interests, primarily utilizing user-item interaction sequences with timestamps to capture the dynamic changes in user interests and item attributes. Recent research has mainly centered on two aspects. First, it involves modeling the dynamic interaction relationships between users and items using dynamic graphs. Second, it focuses on mining their long-term and short-term interaction patterns. This is achieved through the joint learning of static and dynamic embeddings for both users and items. Although most existing methods have achieved some success in modeling the historical interaction sequences between users and items, there is still room for improvement, particularly in terms of modeling the long-term dependency structures of dynamic interaction histories and extracting the most relevant delayed interaction patterns. To address this issue, we proposed a Dynamic Context-Aware Recommendation System for dynamic recommendation. Specifically, our model is built on a dynamic graph and utilizes the static embeddings of recent user-item interactions as dynamic context. Additionally, we constructed a Gated Multi-Layer Perceptron encoder to capture the long-term dependency structure in the dynamic interaction history and extract high-level features. Then, we introduced an Attention Pooling network to learn similarity scores between high-level features in the user-item dynamic interaction history. By calculating bidirectional attention weights, we extracted the most relevant delayed interaction patterns from the historical sequence to predict the dynamic embeddings of users and items. Additionally, we proposed a loss function called the Pairwise Cosine Similarity loss for dynamic recommendation to jointly optimize the static and dynamic embeddings of two types of nodes. Finally, extensive experiments on two real-world datasets, LastFM, and the Global Terrorism Database showed that our model achieves consistent improvements over state-of-the-art baselines.

Keywords: recommendation systems; dynamic graph; context-aware recommendation

1. Introduction

With the rapid development of the Internet and big data technology, recommendation systems have become an integral part of technical tools and application solutions in multiple industries and fields, particularly in e-commerce [1], social media [2], and online education [3]. These systems demonstrate significant application value in these areas. Moreover, they also play an important role in public safety warning [4–6], environmental monitoring [7], and other domains of social welfare. Traditional recommendation systems are mostly built on static models, which means they typically process historical interaction data using one-time or batch updates, making it difficult to capture the dynamic changes in user interests and item attributes [8–12]. In contrast, dynamic recommendation systems, as a subset within the context-aware recommendation system domain [13,14], focus on considering time-related contextual information, such as the interaction history of users and items. They employ more complex models, like Recurrent Neural Networks (RNNs), to capture the dynamic interaction patterns between users and items [15,16]. Due to their ability to accurately reflect and adapt to the dynamic changes in users and items, dynamic recommendation systems are emerging as a cutting-edge direction in the research and application of recommendation algorithms [17,18].

In this paper, the two types of entities, users and items, can cover a variety of notions, e.g., users could be customers in an e-commerce system, or accounts on social media; items could be products, posts, or media produced or consumed by users. To more comprehensively learn the dynamic changes in user interests and item attributes, and thereby provide precise personalized recommendations, we propose a novel dynamic recommendation approach. This method captures the dynamic relationships between users and items by mining the most relevant delayed interaction patterns within the dynamic interaction history, aiming to predict the items with which users are likely to interact in the future [6,18]. Dynamic graphs naturally handle graph data structures with timestamps, where each edge represents a user-item interaction at a specific time point [19]. Therefore, we construct our model based on dynamic graphs, enabling it to effectively capture the evolving interaction patterns between users and items over time.

Currently, several methods have been proposed to predict which items users are likely to interact with in the future [16,20–25]. However, these methods often suffer from a common limitation: they typically generate corresponding embeddings only when users perform specific actions, thereby overlooking the rich information contained in the historical interaction sequences [8]. Therefore, RNNs and other models optimized for sequential data [15,26–28] have been employed to model the historical interaction sequences and capture the long-term dependency relationships among items in the sequence. Recent studies have demonstrated significant improvements over traditional methods by jointly modeling users and items using dynamic graphs [16,21]. One key challenge lies in accurately capturing the evolution of user interests and item attributes over time. This is because users may interact with different items in a sequential manner, and their interests can change within a certain time frame. Similarly, item attributes also undergo continuous changes and are heavily influenced by user behavior. In contrast to previous research, JODIE [16], Dynamic Graph Collaborative Filtering (DGCF) [17], and several other dynamic evolution models [29–31] employ mutually recursive RNNs to model the dynamic interaction between users and items. However, these methods heavily rely on the embeddings at time t-1 to recursively compute the dynamic embeddings at time t. As the network scale increases,

this recursive training approach has been shown to be a performance bottleneck. Although recent studies have attempted to alleviate this burden by introducing a batch processing algorithm called t-Batch, the computational overhead has not been fundamentally reduced [16]. In contrast, DeePRed proposes a simple and efficient approach to model short-term interaction behavior, effectively eliminating the aforementioned recursive dependency issue and achieving a more efficient training process [18].

Although there is great potential in constructing dynamic embeddings by capturing the dynamic interaction relationships between users and items for dynamic recommendation, there is still room for further improvement in modeling the long-term dependency structures of dynamic interaction histories and extracting the most relevant delayed interaction patterns. In dynamic recommendation systems, delayed interaction patterns refer to the interactions between users and items that occurred during a certain period in the past. Although these interactions happened at earlier points in time, they still significantly influence the user's current interests and preferences. The key to mining delayed interaction patterns lies in identifying and learning those historical interactions that continue to have a significant impact on current recommendation decisions. For instance, a user might have frequently watched a certain genre of movies or purchased specific types of products over a past period. Although such behavior may have decreased recently, these long-term interests and preferences still influence their current choices. Therefore, effectively mining and utilizing delayed interaction patterns can help recommendation systems to understand the evolution of user interests more comprehensively, thereby providing more accurate and personalized recommendations. To this end, we propose a Dynamic Context-Aware Recommendation System (DyCARS) for dynamic recommendation. By treating the dynamic interaction history of users and items as the dynamic context, this method utilizes a Gated Multi-Layer Perceptron (GaMLP) encoder to learn the long-term dependency structure of user-item interaction sequences. Moreover, it employs an Attention Pooling (AP) network to extract the most relevant delayed interaction patterns from the dynamic interaction history. In summary, our main contributions are as follows:

1) We propose a novel non-recursive dynamic recommendation model, DyCARS, which effectively captures the long-term dependency structure in the dynamic interaction history using the GaMLP encoder introduced in this paper and further extracts high-level features. Additionally, the AP network is introduced to learn similarity scores between high-level features of user and item dynamic interaction histories. By computing bidirectional attention weights, the model is able to identify the most relevant delayed interaction patterns from the historical interaction sequence. As a result, the model can effectively predict the dynamic embeddings of users and items.

2) We emphasize the importance of the loss function in dynamic recommendation and accordingly propose the Pairwise Cosine Similarity (D-PCS) loss function to jointly optimize the static and dynamic embeddings of two types of nodes. In the L_{PCS} term of the D-PCS loss function, we utilize cosine similarity to calculate the similarity between user-item pairs and allow the model to emphasize the learning of hard negative samples, which aids in learning more discriminative features and enhances the model's ability to extract useful information.

3) Extensive experimental analyses were conducted on two public datasets, LastFM and the Global Terrorism Database (GTD). Compared to the state-of-the-art baseline models, the experimental results confirm that our proposed DyCARS achieves the best performance.

The remaining parts of this paper are organized as follows: Section 2 formally describes the dynamic recommendation problem addressed in this paper. Section 3 provides a brief overview of the related research in the field of dynamic recommendation systems. Section 4 elaborates on the overall architecture and individual components of the proposed model. In Section 5, experiments are

conducted on two real-world datasets to validate the effectiveness of the proposed model. Finally, Section 6 concludes the paper and outlines future directions.

2. Problem definition

The purpose of this study is to learn representations of users and items based on their current interaction behavior and historical records, and use these representations to predict the items that users may be interested in the future. This "interest" is the relationship between the user and the recommended item. Dynamic graph structures provide a natural and effective way to depict the evolving dynamic interaction relationship between users and items over time [17,32]. Therefore, we utilize dynamic graph structures to model the problem of dynamic recommendation.

Definition 1: Dynamic graph. Essentially, the dynamic graph in this context is a bipartite graph, where users and items are considered as nodes, and all interaction behaviors occur between user nodes and item nodes. If at time $t \in \mathbb{R}^+, \forall t \in [0,T]$, an interaction occurs between a user and an item, a dynamic graph $G_t = (U \cup V, E_t)$ is formed. Here, U and V represent the sets of users and items, respectively, and $|U|$ and $|V|$ denote the number of nodes in the user and item sets. $U \cup V$ represents a finite set containing all user and item nodes, satisfying $U \cap V = \emptyset$. E_t represents a finite set of all interactions in G_t, which essentially refers to the collection of all interactions between users and items up until time t. At the initial time t_0, the dynamic graph $G_{t_0} = (U \cup V, E_{t_0})$ consists of isolated nodes or snapshots of the dynamic graph, where the initial embeddings of users and items are initialized as either initial feature vectors or random vectors.

Definition 2: User-item interaction events. In the context of dynamic recommendation, the interaction between a user and an item can be represented as a triplet $i = (u, v, t)$, where at a specific time t, the user node $u \in U$ interacts with the item node $v \in V$, denoted as $(u, v) \in E_t$. From this perspective, a dynamic graph can be understood as a sequential arrangement of user-item interaction events $I = \{(u, v, t)_n : n = 1,2, \cdots, |E_T|\}$. Let I_u denote the set of all interaction events for user u, which can be expressed as $I_u = \{i_1^u, i_2^u, \cdots, i_N^u\}$. Similarly, I_v represents the set of all interaction events for item v, written as $I_v = \{i_1^v, i_2^v, \cdots, i_M^v\}$. For a specific user u and item v, I_u and I_v respectively capture their complete interaction history.

Definition 3: Static and dynamic embeddings. When predicting future interactions between users and items, long-term and short-term interaction patterns are often considered [21,33,34]. We adopt this assumption and model the users and items from both long-term and short-term perspectives. From the user's point of view, although it is commonly believed that user interests evolve over time [15], we assume the existence of a stable long-term interest that remains unchanged. Additionally, the user's interest may also exhibit a corresponding short-term trend based on recent behavior changes. For example, in music recommendation systems, users may have a tendency towards a specific genre of music at any given moment. However, individuals may also be influenced by emotions or life events, leading to an interest in different genres during specific time periods. To encode the long-term static attributes and short-term dynamic attributes of users and items, we allocate static and dynamic embeddings to all users and items.

Static embeddings. We define the static embeddings of user u and item v at time t as $\bar{u} \in \mathbb{R}^d$ and $\bar{v} \in \mathbb{R}^d$, respectively. These embeddings do not change over time. In other words, we use an embedding matrix $E \in \mathbb{R}^{\mathrm{d} \times (|U|+|V|)}$ as the static embedding matrix, which is trained when observing user-item interactions.

Dynamic embeddings. We assign dynamic embeddings to each user u and item v at time t, represented as $u(t)|I_u(t^<, n) \in \mathbb{R}^k$ and $v(t)|I_v(t^<, n) \in \mathbb{R}^k$, respectively. These embeddings

effectively capture the evolving user interests and item attributes over time. $I_u(t^<, n)$ and $I_v(t^<, n)$ denote the n most recent interaction events for each user u and item v before time t:

$$I_u(t^<, n) = \{v_i, \Delta_i : (u, v_i, t_i) \in I_u, t_i < t, i = p - n, \cdots, p\} \quad (1)$$

$$I_v(t^<, n) = \{u_i, \Delta_i : (u_i, v, t_i) \in I_v, t_i < t, i = p - n, \cdots, p\} \quad (2)$$

where $\Delta_i = t - t_i$ represents the time interval between the current moment t and the previous interaction time t_i. It effectively captures the temporal decay effect of recent interaction events on the current interaction event.

Table 1. Mathematical notation used in this paper.

Symbols	Definitions and Descriptions
U	User set
V	Item set
G_t	The dynamic graph at time t
$\|U\|$	The number of user nodes
$\|V\|$	The number of item nodes
E_t	The finite set of all interactions in G_t
I	Sequential arrangement of user-item interaction events over time
I_u	The set of all interaction events for user u
I_v	The set of all interaction events for item v
$\bar{u}$	The static embedding of user u at time t
$\bar{v}$	The static embedding of item v at time t
E	Static embedding matrix
$I_u(t^<, n)$	The n most recent interaction events of user u before time t
$I_v(t^<, n)$	The n most recent interaction events of item v before time t
$u(t)\|I_u(t^<, n)$	The dynamic embedding of user u at time t
$v(t)\|I_v(t^<, n)$	The dynamic embedding of item v at time t
Δ_i	The time interval between the current moment t and the previous interaction time t_i
$X_u(t)$	The embedding matrix of user u's n most recent interaction history at time t
$X_v(t)$	The embedding matrix of item v's n most recent interaction history at time t
$Z_u(t)$	The high-level feature matrix of user u's most recent n interaction histories at time t
$Z_v(t)$	The high-level feature matrix of item v's most recent n interaction histories at time t
$S(t)$	The similarity score matrix of high-level features among the most recent n interaction histories of users and items
$Att_u(t)$	The attention vector of user u at time t
$Att_v(t)$	The attention vector of item v at time t
$N(u)$	The set of negatively sampled items for user u based on random probability sampling

We adopt the approach consistent with previous studies [33,35] to capture the short-term dynamic characteristics of users or items by incorporating the n most recent interaction events. However, unlike the research in [20,24,26,36], we consider that the n recent interaction events of both users and items have an impact on the next user-item interaction. In previous studies, the dynamic embeddings of users and items often recursively depend on their respective previous dynamic embeddings [16,21]. The recursive nature of these algorithms increases computational costs, necessitating the use of specialized batch processing

algorithms to avoid sequential processing. Table 1 summarizes the mathematical notation used in this paper.

Research Problem. The main problem addressed in this study is: Given a known ordered sequence of user-item interaction events I, can we design an algorithm to effectively predict future interaction behavior in the dynamic graph?

3. Related work

The main task of dynamic recommendation models is to capture the dynamic changes of users and items from historical and current interactions, so as to accurately predict the embedding trajectory of users or items over time to recommend items that users may interact with in the future. In the field of dynamic recommendation research, conventional methods typically employ standard deep neural network (DNN) architectures that integrate manually designed features with learned features into the recommendation model. For instance, Covington et al. [20] divided YouTube's recommendation system into two separate stages: candidate generation and ranking. Both stages utilized structurally similar deep neural networks, resulting in significant improvements in recommendation performance.

To accurately capture the repetitive patterns in user-item interaction sequences, Recurrent Neural Networks (RNNs) and their variants have been widely applied in dynamic recommendation. Specifically, Hidasi et al. [26] were the first to apply RNNs to session-based recommendation, particularly in scenarios that rely on short-term session data for recommendations. Similarly, Wu et al. [15] employed Long Short-Term Memory (LSTM) networks to model the dynamic changes in users and movies, thereby predicting future behavioral trajectories. Beutel et al. [33] effectively incorporated various types of contextual data into a Gated Recurrent Unit (GRU)-based recommendation system by performing element-wise multiplication on the model's hidden states.

Activities closer in time to a specific event are often more likely to trigger that event. Therefore, modeling the time differences in historical behavior sequences has shown significant advantages in improving recommendation system performance. Although standard RNNs and their variants can capture repetitive patterns in sequences, they lack the ability to encode the time intervals between historical events. To address this limitation, extensive research has been conducted to enhance RNNs and their variants by fully considering the factor of time intervals [34,35]. For instance, Zhu et al. [34] designed dedicated temporal gates based on LSTM, enabling the model to more effectively capture the time interval factors in user behavior sequences.

Although the aforementioned methods have successfully simulated the evolution of user interests through user-item interaction sequences, they fall short in fully consider the changing trends of items themselves. To alleviate this limitation, several studies have attempted to combine point process models with RNNs to jointly learn dynamic embeddings of users and items for next-item prediction. Specifically, Dai et al. [21] employed RNNs to define the intensity function of the point process, enabling the capture of complex interdependencies and the temporal evolution of features between users and items. Kumar et al. [16] proposed a coupled recursive neural network recommendation model that included two steps of embedding update operations and an embedding projection function to predict the dynamic embeddings of users and items. Li et al. [17] first applied dynamic graphs to dynamic recommendation by integrating second-order neighborhood information from dynamic interaction networks to extend JODIE. Hou et al. [6] simultaneously introduced Hawkes processes and dynamic graphs into dynamic recommendation scenarios to model the dynamic interactions and evolution between users and items.

Table 2. Comparison of DyCARS with other works.

Dynamic recommendation system methods	Key features	Contribution	Comparison of DyCARS with other works
Traditional DNN Architecture (e.g., Covington et al. [20])	These methods utilize deep neural networks to combine manually designed features with learned features.	These methods improve system performance but may not fully capture temporal dynamics.	DyCARS leverages dynamic interaction history as context to better capture the temporal dynamics and evolution of user behavior.
RNN and its variants (such as Hidasi et al. [26], Wu et al. [15])	These methods are used to capture repetitive patterns in sequences, especially short-term session data.	These methods are effective in session-based recommendation but are less capable in modeling long-term dependencies.	DyCARS learns the sequential dependency relationships between elements in historical sequential data, providing a more effective modeling of the long-term dependency structure in the user-item dynamic interaction history.
Time-aware RNN and its variants (e.g., Zhu et al. [34])	These models are capable of effectively capturing the time interval factors in user behavior sequences.	These models can simulate the evolution of user interests but do not adequately consider the trends in items.	DyCARS emphasizes the dual aspects of dynamic interactions, namely the dynamic changes in both users and items.
Recursive dynamic recommendation models (e.g., Dai et al. [21], Kumar et al. [16], Li et al. [17])	These models combine point processes with dynamic graphs to capture the mutual influence between users and items.	These models effectively capture the dynamic interactions and evolution between users and items, but as the network scales up, they incur significant computational costs.	DyCARS employs a non-recursive approach, reducing computational costs while effectively modeling dynamic interactions.
Non-recursive dynamic recommendation models (such as Kefato et al. [18])	These models utilize static embedding proxies to model dynamic embeddings.	These models effectively avoid recursion and high computational costs, but there is still room for improvement in capturing the long-term dependency structure and delayed interaction patterns in dynamic interaction history.	DyCARS combines GaMLP encoder and AP network, enabling more effective modeling of long-term dependency structures and delayed interaction patterns.

However, these methods primarily rely on recursively calculating the dynamic embeddings at time t based on the embeddings at time t-1. As the network scale increases, this recursive training approach has been proven to be a performance bottleneck. Although recent research attempts to alleviate this issue by introducing a batch processing algorithm called t-Batch, the computational cost has not been fundamentally reduced. To address this problem, Kefato et al. [18] successfully avoid the recursion and high computational costs between consecutive dynamic embeddings by using static embeddings as proxies when modeling dynamic embeddings. Although many existing methods have achieved success in modeling the dynamic interaction between users and items, they still perform

poorly in capturing long-term dependency structures in the historical interaction sequences between users and items. It is challenging to accurately mine and model the most relevant delayed interaction patterns from the dynamic interaction history. To address this limitation, we propose a novel non-recurrent dynamic recommendation model called DyCARS. This model considers the dynamic interaction history between users and items as a dynamic context and effectively addresses the aforementioned shortcomings by utilizing the GaMLP encoder and the AP network. Table 2 summarizes the comparison of DyCARS with other dynamic recommendation system methods.

4. Research method

In this section, we will detail the design and implementation of DyCARS. Initially, an overview of the model architecture is presented, elucidating the overall framework of DyCARS and its core components. Subsequently, we delve into a key component of DyCARS: the GaMLP encoder. This part will explain in detail the working mechanism of the GaMLP encoder, including its handling of the embedding matrix of recent interaction history and the design and functionality of the spatial gating module. These components collectively capture the long-term dependencies in the dynamic interaction history and extract high-level features. Following this, the section introduces the design and application of the AP network. The AP network plays a pivotal role in DyCARS, learning the similarities and bidirectional attention weights among high-level features in the dynamic interaction history of users and items. It identifies the most relevant delayed interaction patterns from the historical interaction sequences, further enhancing the model's recommendation capabilities. Lastly, we discuss the D-PCS loss function and model optimization strategies. This part will demonstrate how the specially designed loss function effectively co-optimizes static and dynamic embeddings, achieving more efficient and accurate recommendations.

4.1. Model architecture

Figure 1 illustrates the overall architecture of DyCARS. The interactions between users and items form a dynamic graph over time. The input of DyCARS is determined by the observed user-item interaction events and the hyper-parameter *n* of the model. Whenever a new user-item interaction event is observed, the model parameters are updated accordingly. DyCARS is built on dynamic graphs and considers the *n* most recent interaction events of users and items (i.e., $I_u(t^<, n)$ and $I_v(t^<, n)$) prior to time *t* as dynamic contexts. These contexts are captured through static embeddings ($\bar{u}_i$ or $\bar{v}_i, i = 1, \cdots n$). Furthermore, we employ the proposed GaMLP encoder to model long-term dependency structures in the *n* recent interaction histories and extract high-level features. Subsequently, an AP network is utilized to compute similarity scores between the high-level features of user-item dynamic interaction histories. By calculating bidirectional attention weights, we are able to discover the most relevant delayed interaction patterns from the historical sequences, thus predicting the dynamic embeddings ($u(t)$ and $v(t)$) of users and items at time *t*. Finally, we jointly optimize the static and dynamic embeddings of users and items using the D-PCS loss function to make more accurate predictions for the next item recommendation.

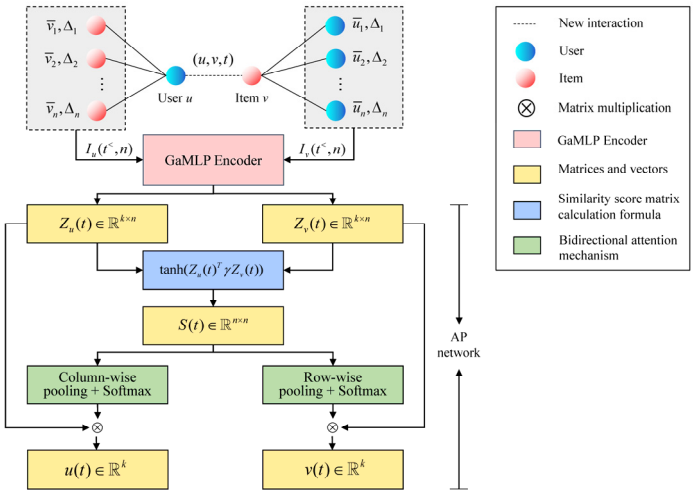

Figure 1. Overall architecture of DyCARS.

4.2. The GaMLP encoder

To model the long-term dependency structure in the user-item dynamic interaction history, we propose the GaMLP encoder. In DyCARS, the user encoder and item encoder share the same architecture. Therefore, we primarily focus on the user side to illustrate the structural design of the GaMLP encoder.

As illustrated in Figure 2, the GaMLP encoder consists of L modules with the same size and structure. Let $X_u(t)$ denote the embedding matrix of the recent n interaction histories for user u at time t (see Section 4.2.1 for details). Each module is defined as follows:

$$Y_u(t) = \sigma(X_u(t)W), \tilde{Y}_u(t) = \mathrm{SG}(Y_u(t)), Z_u(t) = \tilde{Y}_u(t)V \tag{3}$$

where σ is the activation function, e.g., Gaussian error linear unit (GeLU) [37]. W and V define linear projections along the channel dimension. For brevity, the above equations omit residual connections, normalization operations, and bias terms.

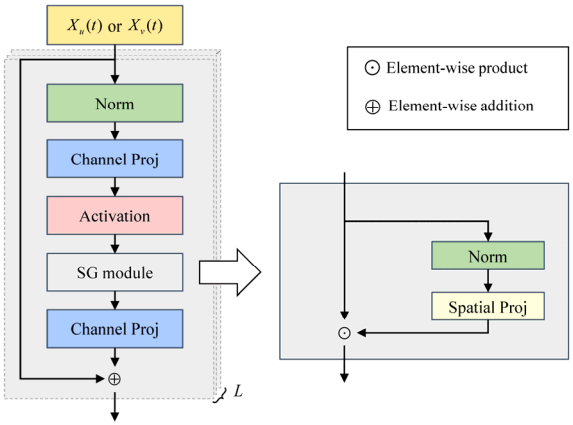

Figure 2. Architecture of the GaMLP encoder.

One key function in Eq (3) is $\mathrm{SG}(\cdot)$, which aims to capture the mutual interactions between dynamic interaction histories (see Section 4.2.2 for details). When $\mathrm{SG}(\cdot)$ behaves as an identity mapping, the $\mathrm{SG}(\cdot)$ transformation degenerates into a regular Feedforward Neural Network (FFN), where individual interaction history embeddings are processed independently without any cross interaction history embeddings communication. Therefore, one of our focuses is to design an $\mathrm{SG}(\cdot)$ that can effectively capture complex interactions between dynamic interaction histories. The overall architectural inspiration comes from inverted bottlenecks [38], where $\mathrm{SG}(\cdot)$ is defined as spatial depthwise convolutions.

Finally, the output of the GaMLP encoder, $Z_u(t) \in \mathbb{R}^{k\times n}$, represents the high-level feature matrix of the recent *n* interaction histories for user *u* at time *t*. It captures the interactions between dynamic interaction histories, effectively modeling the long-term dependency structure in the dynamic interaction history. Similarly, the encoding process for the recent interaction histories of items is identical to that of users. The output, $Z_v(t) \in \mathbb{R}^{k\times n}$, represents the high-level feature matrix of the recent *n* interaction histories for item *v* at time *t*. For instance, in a movie recommendation system, the GaMLP encoder can be used to capture the dynamic changes and long-term dependencies in users' movie-watching history and the history of movies being watched. Specifically, we take the most recent *n* movie-watching records of user *u* as input. The GaMLP encoder transforms these viewing records into a high-level feature matrix $Z_u(t) \in \mathbb{R}^{k\times n}$, where each column represents a specific viewing event, encompassing high-level information related to that event. By analyzing these high-level features, the GaMLP encoder can understand the dynamic changes in user viewing preferences, such as whether a user is increasingly favoring a certain genre of movies. Similarly, for a specific movie, we can also employ the GaMLP encoder to analyze its viewing history. In this scenario, the encoder's output $Z_v(t) \in \mathbb{R}^{k\times n}$ represents the high-level feature matrix of the most recent *n* viewings of movie *v* at time *t*. Through this approach, the GaMLP encoder can effectively model the long-term dynamic interaction history of users and items, providing essential input information for the AP network.

4.2.1. Embedding matrix of recent interaction history

A simple yet expressive technique employed by the GaMLP encoder is to utilize the embedding matrix $X_u(t) \in \mathbb{R}^{\mathrm{n}\times(\mathrm{d}+1)}$ (or $X_v(t) \in \mathbb{R}^{\mathrm{n}\times(\mathrm{d}+1)}$) of the user's (or item's) recent *n* interaction histories as input to the model, which is defined as follows:

$$X_u(t) = g\big(I_u(t^<, n)\big) = [[\bar{v}_i; \Delta_i] : (v_i, \Delta_i) \in I_u(t^<, n)] \tag{4}$$

$$X_v(t) = g\big(I_v(t^<, n)\big) = [[\bar{u}_i; \Delta_i] : (u_i, \Delta_i) \in I_v(t^<, n)] \tag{5}$$

In Eq (4), $X_u(t)$ is dependent on the static embeddings of the *n* most recent interacted with items by user *u*. Similarly, in Eq (5), the static embeddings of the *n* users who have recently interacted with item *i* are used to compute $X_v(t)$. The key assumption here is that the static embeddings of multiple recently interacted with items capture strong signals of a user's recent interests, as each static embedding $\bar{v}_i \in X_u(t)$ captures the intrinsic attributes or context of item v_i. It is important to note that the embedding matrix at time *t* only contains information from the past (i.e., before time *t*), as our objective is to predict the current (i.e., at time *t*) state.

Furthermore, research has also indicated that the time difference between user-item interactions

plays a crucial role in predicting future interactions. Therefore, in the embedding matrix of recent interaction history, each static embedding is concatenated with the time difference Δ to capture the temporal decay effect of recent interaction events on the current interaction event. It is important to note that some studies use decay functions instead of the time difference Δ, such as $f(\Delta) = 1/\log(e + \Delta)$ [33–35]. However, through our experimental testing, we have found no significant difference in these methods, and thus we opted to directly utilize Δ.

4.2.2. Spatial gating module

As shown in Figure 2, to capture the interactions between the user-item dynamic interaction history, the Spatial Gating (SG) module incorporates projection operations on the spatial dimension. Taking the user side as an example, we denote $\mathrm{SG}(\cdot)$ as:

$$\mathrm{SG}(Y_u(t)) = Y_u(t) \odot h_{U,b}(Y_u(t)) \tag{6}$$

$$h_{U,b}(Y_u(t)) = UY_u(t) + b \tag{7}$$

where $\odot$ denotes the element-wise multiplication operation. $U \in \mathbb{R}^{n \times n}$ represents a spatial projection matrix of the same size as the length n of the recent interaction history, and b represents the bias term. To ensure training stability, we have found it crucial to initialize U with values close to zero and initialize b as 1. This implies that, at the beginning of the training process, $h_{U,b}(Y_u(t))$ is approximately equal to 1, resulting in $\mathrm{SG}(Y_u(t))$ being close to $Y_u(t)$. This initialization method ensures that each GaMLP module behaves similarly to a conventional FFN during the initial stages of training. In this setup, each interaction history embedding is independently processed, gradually incorporating spatial information across the interaction history embeddings during the course of learning.

Furthermore, we have also found that introducing a residual mechanism (Eq (8)) or splitting $Y_u(t)$ into two separate parts along the channel dimension for gating functions and multiplication bypasses (Eq (9)) is effective:

$$\mathrm{SG}(Y_u(t)) = Y_u(t) + h_{U,b}(Y_u(t)) \tag{8}$$

$$\mathrm{SG}(Y_u(t)) = Y_u^1(t) \odot h_{U,b}(Y_u^2(t)),\ Y_u(t) = Y_u^1(t) \| Y_u^2(t) \tag{9}$$

In addition, we also normalized the input of $h_{U,b}(\cdot)$, a strategy that has been proven in practice to enhance the stability of the model.

Relation to Existing Networks. The overall formulation of the SG module bears resemblance to Gated Linear Units (GLU) [39] as well as early works such as Highway Networks [40] and LSTM [41]. However, a key distinction lies in the fact that our SG module operates on the spatial dimension for computations, rather than the channel dimension. In terms of element-wise multiplication operations, there is a connection between the SG module and SE blocks [42]. However, the SG module does not include cross-channel projections nor does it enforce permutation invariance. The spatial projection in the SG module can theoretically learn to express superficial depthwise convolutions; unlike typical depthwise convolutions with channel-specific filters, the SG module learns a single transformation shared across channels.

4.3. The AP network

Static embeddings of users and items are both located in the same space, from which the derived high-level features $Z_u(t)$ and $Z_v(t)$ are based. The AP network enables the direct influence of information from these two high-level features on each other's computations. The main idea behind the AP network is to calculate similarity scores between high-level features based on the dynamic interaction history of users and items, and determine bidirectional attention weights according to these similarity scores of interaction histories [43].

Figure 1 illustrates how the AP network is applied to the output of the GaMLP encoder to predict the dynamic embeddings of users and items ($u(t)$ and $v(t)$) at time *t*. Considering the input pair $(Z_u(t), Z_v(t))$, the similarity score matrix $S(t) \in \mathbb{R}^{n\times n}$ between high-level features in the most recent *n* interaction histories of users and items is represented as:

$$S(t) = \tanh(Z_u(t)^T \gamma Z_v(t)) \tag{10}$$

where γ is the parameter matrix. However, based on our empirical observations, setting γ as the identity matrix yields superior results. Therefore, in DyCARS, the only parameters that need to be determined are the static embedding matrix *E* and the parameters associated with the GaMLP encoder, including *W*, *V*, *U*, and bias terms.

Next, we apply max pooling (or average pooling) along the columns and rows of matrix $S(t)$, resulting in the generation of vectors $\tilde{u}(t) \in \mathbb{R}^n$ and $\tilde{v}(t) \in \mathbb{R}^n$, respectively. Formally, the computation of the *i*-th element of vectors $\tilde{u}(t)$ and $\tilde{v}(t)$ is as follows (using max pooling as an example):

$$[\tilde{u}(t)]_i = max_{1\le j\le n}[S(t)_{i,j}] \tag{11}$$

$$[\tilde{v}(t)]_i = max_{1\le m\le n}[S(t)_{m,i}] \tag{12}$$

The *i*-th element of vector $\tilde{u}(t)$ can be interpreted as the importance score of the *i*-th interaction history of user *u* relative to the most recent *n* interaction histories of item *v*. Similarly, the *i*-th element of vector $\tilde{v}(t)$ can be interpreted as the importance score of the *i*-th interaction history of item *v* relative to the most recent *n* interaction histories of user *u*.

Next, we apply the Softmax function to vectors $\tilde{u}(t)$ and $\tilde{v}(t)$ to construct attention vectors $Att_u(t)$ and $Att_v(t)$. Taking the *i*-th element of $Att_u(t)$ as an example, its calculation formula is as follows:

$$[Att_u(t)]_i = \frac{e^{[\tilde{u}(t)]_i}}{\sum_{1\le j\le n} e^{[\tilde{u}(t)]_j}} \tag{13}$$

Finally, the dynamic embeddings of the user and item at time *t* ($u(t)$ and $v(t)$) are calculated as follows:

$$u(t) = Z_u(t)Att_u(t) \tag{14}$$

$$v(t) = Z_v(t)Att_v(t) \tag{15}$$

Both of the above equations can be seen as feature selectors based on bidirectional attention mechanisms. Specifically, the Softmax function provides a weight distribution for different events in the dynamic interaction history of user *u* and item *i*, allowing the recent interaction events with larger attention weights to dominate the projection operation. Therefore, DyCARS, by learning the allocation

of attention weights, is able to extract more accurately the most relevant delayed interaction patterns from the historical sequence, thus predicting the dynamic embeddings of users and items.

4.4. D-PCS loss and model optimization

Learning the parameters of DyCARS requires defining an objective function for model optimization. In this study, we propose a novel loss function, D-PCS, suitable for dynamic recommendation tasks. The D-PCS loss function is defined as:

$$L = min\frac{1}{B}\sum_{(u,v,t)\in I_{train}}\|u(t) - v(t)\|_2^2 + L_{REG} + L_{PCS}(u,v) \tag{16}$$

where B represents batch size, and I_{train} represents the observed interaction events in the training set. In the D-PCS loss function, the objective of the first term of the loss is to jointly train dynamic embeddings and static embeddings to make the projections of frequently interacted items as close as possible. Therefore, we minimize the L_2 distance between the dynamic embeddings of users and items.

To avoid the trivial solution of the D-PCS loss function, we introduce a regularization loss, L_{REG}. This strategy is inspired by Laplacian eigenmaps, which avoids collapse by adding the constraint $u(t)^T v(t) = 1$. Therefore, we define L_{REG} as:

$$L_{REG} = \alpha\left\|[u(t); v(t)]^T[u(t); v(t)] - I_{identity}\right\|_F^2 \tag{17}$$

where α represents the regularization coefficient, and $I_{identity}$ denotes the identity matrix.

For the third term in the D-PCS loss function, given a positive user-item pair (u, v) and a set of negative sample items $N(u)$ randomly sampled for user u, L_{PCS} is defined as:

$$L_{PCS}(u,v) = (1 - r_{uv}) + \frac{\tau}{|N(u)|}\sum_{m\in N(u)} Relu(r_{um} - M) \tag{18}$$

$$r_{uv} = Cosine(\bar{u}, \bar{v}) \tag{19}$$

where $|N(u)|$ represents the number of negative samples; M is the threshold for filtering negative samples, typically ranging from 0.4 to 0.6 and may vary depending on the specific experiment; τ is used to control the relative weight of positive and negative sample losses, typically taking values of 1, 2, or 4; r_{uv} represents the preference matching score between user u and item v; and $Cosine(\cdot)$ denotes the cosine similarity between the static embeddings of users and items, defined as follows:

$$Cosine(A,B) = \frac{A\cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{d} A_i \times B_i}{\sqrt{\sum_{i=1}^{d}(A_i)^2}\times\sqrt{\sum_{i=1}^{d}(B_i)^2}} \tag{20}$$

The objective of L_{PCS} is to maximize the preference matching score between positives and minimize the preference matching score between negatives. In random probability sampling, we sample based on the occurrence frequency of each item among all negative samples. The mathematical expression of this can be formulated as:

$$P(v) = \frac{f(v)^{\frac{3}{4}}}{\sum_{m \in N(u)} \left(f(m)^{\frac{3}{4}} \right)} \tag{21}$$

where $f(v)$ represents the frequency of item v appearing in all negative samples, and $P(v)$ denotes the sampling probability of item v. This frequency-based sampling method balances the occurrence probabilities of different items during the sampling process by utilizing probability weights. Consequently, it increases the chances of selecting low-frequency items while reducing the sampling probability of high-frequency items. This strategy effectively alleviates biases caused by significant frequency differences, thereby enhancing the effectiveness and diversity of the sampling process.

The design inspiration of L_{PCS} originates from the widely used contrastive loss in computer vision tasks [44,45]. In contrast to common loss functions in the field of recommendation systems, we made several choices in the design of L_{PCS} to facilitate model training and achieve better performance.

First, we chose to use cosine similarity to compute the similarity (or preference matching degree) between user-item pairs, as opposed to using the dot product (e.g., LightGCN [46]), Euclidean distance (e.g., Collaborative metric learning [47]), or multi-layer perceptron (e.g., Dual sIde deepContext-awarEmodulation for socialRecommendation [48]). The reason for this choice is that, similar to the calculation of word similarity in word2vecv [49], cosine similarity effectively measures the similarity between vectors and avoids the issue of large value ranges that may arise with the dot product or other methods. Additionally, cosine similarity can handle cases where the vector lengths of users and items differ, making it more suitable for feature vectors of users and items in recommendation systems. For example, in recommendation tasks, the embedding size of users or items may be influenced by their popularity, thus making the advantages of cosine similarity particularly relevant in such scenarios.

Second, we consider the issues caused by the increase in the number of negative samples. Typically, the increased negative samples contain a large amount of redundant and ineffective data, which may interfere with the model's training process, leading to decreased model performance and slower training speed. To address this problem, L_{PCS} introduces a threshold M to filter out these useless negative samples. When the cosine similarity of a negative sample is too small and falls below M, they will be assigned a loss of zero. Therefore, L_{PCS} can automatically identify the hard negative samples with cosine similarity greater than M and optimize the model training based on them. This strategy is similar to the widely used hard example mining technique in the field of computer vision [50].

Finally, for dynamic recommendation systems, there is often a severe imbalance between positive and negative samples. Summing or averaging the loss of all negative samples directly may have adverse effects on the model's performance. Therefore, we introduce a weight factor τ to adjust the balance between the loss of positive and negative samples. This helps improve the model's performance, especially when there is a large number of negative samples. Moreover, this is a common approach to addressing class imbalance issues.

To optimize the objective function, we choose to use the Adaptive Moment Estimation (Adam) as the optimizer. Adam is a gradient-based optimization algorithm that automatically adjusts the learning rate during the training phase and performs well in handling sparse gradients and non-stationary objective functions. Additionally, to mitigate the overfitting issue, we employ the Dropout strategy [51]. Dropout is a widely accepted regularization technique that randomly sets a portion of the neuron outputs to zero during the training phase, thereby preventing the model from overfitting to the training data excessively. By combining the Dropout optimization strategies, our model can

effectively handle the training data and enhance recommendation performance.

5. Experiment

A series of experiments were conducted for a comprehensive evaluation of DyCARS, aiming to address the following research questions:

Q1: How does DyCARS perform compared to state-of-the-art dynamic recommendation methods?

Q2: Are the core components of the GaMLP encoder, gating mechanism, bidirectional attention mechanism, and L_{PCS} loss essential for improving the recommendation performance of DyCARS?

Q3: What is the operational efficiency of DyCARS?

Q4: How diverse are the recommendations provided by DyCARS?

Q5: How do the hyper-parameters in DyCARS affect the recommendation performance?

5.1. Experiment setup

In this section, we will provide a detailed introduction to the experimental setup, including the selection and preprocessing of datasets, the determination of baseline methods and evaluation metrics, as well as the configuration of experimental parameters and the description of the experimental environment.

5.1.1. Datasets and preprocessing

In order to comprehensively evaluate the performance of the model, a series of experiments were conducted on two real-world datasets. The details of the datasets are shown in Table 3.

Table 3. The details of the datasets.

Dataset	users	items	Interactions	Action Repetition
LastFM	1000	1000	1293,103	8.6%
GTD	566	744	99,043	22.7%

1) LastFM: This dataset is widely adopted and consists of music play records within a month [52]. We selected a total of 1000 users and the top 1000 most played songs for our study, resulting in 1,293,103 interactions. Note that users only listened to the same song continuously in 8.6% of the interactions. For the experiments, we divided the data into training, validation, and testing sets in proportions of 80, 10, and 10%, respectively.

2) GTD: The GTD is widely recognized as the most comprehensive database covering global terrorist incidents, encompassing over 200,000 terrorist activities carried out by nearly 3000 terrorist organizations worldwide since 1970. This study aims to achieve early warning of terrorist attacks by using DyCARS to predict the next province or state where a terrorist organization is likely to strike at a specific time. During the experiment, we excluded events with unclear dates, unknown perpetrating organizations and attack locations, as well as events involving terrorist organizations and attack locations with fewer than 10 occurrences. Finally, a total of 99,043 events were screened, of which a terrorist organization interacts with the same location of attack consecutively in 22.7% interactions. The data was partitioned based on time, with 70% of the interactions used for model training, 10% for validation, and the remaining interactions for testing purposes.

5.1.2. Baselines and evaluation metrics

To evaluate the performance of DyCARS, we compare it with the following seven baselines:

1) LSTM [41]: This model is a special type of RNN that can capture long-term dependencies.

2) Time-LSTM [34]: This model extends the standard LSTM by adding two time gates to model the influence of time intervals on both the user's current and long-term behaviors.

3) RRN [15]: This model is based on the concept of matrix factorization and utilizes an RNN to learn dynamic embeddings of users and items.

4) DeepCovolve [21]: It is based on co-evolutionary point process algorithms. According to [16], 10 negative samples are used in each interaction.

5) JODIE [16]: This model is a coupled RNN model used to learn dynamic feature representations of users and items.

6) DGCF [17]: This model is a novel framework that captures both the collaborative relationships and historical sequence relationships between users and items using dynamic graphs.

7) DeePRED [18]: This model employs a non-recurrent approach to model the short-term preferences of users and items based on their recent interaction history, which represents the state-of-the-art model in dynamic recommendation problems.

We use the following two evaluation metrics for experiments:

1) Mean Reciprocal Rank (MRR) is the average of the reciprocal rank of the first positive example in all user recommendation lists. The performance of a model on the item ranking list can be evaluated using this metric, where a higher score indicates that the ground truth items are ranked higher in the recommendation list. In certain cases, such as search result ranking for specific users, improving MRR can significantly enhance user experience, especially for items with higher importance in the ranking results. The calculation formula is as follows:

$$MRR = \frac{1}{|I|}\sum_{i \in I}\frac{1}{rank_i} \tag{22}$$

where $i \in I$ represents traversing all interactions and $rank_i$ represents the position of the first ground truth item in the recommendation list for the i-th interaction.

2) Recall@k is the fraction of interactions in which the ground truth item is ranked in the top k. This metric focuses on evaluating the comprehensiveness and diversity of recommendation results, making it highly applicable in recommendation systems. This metric is used because the recommendation system aims to cover a wide range of items within the user's interests, ultimately enhancing user satisfaction and participation. In certain application environments, such as e-commerce and video recommendations, improving Recall@k can significantly optimize the practical effectiveness of recommendation systems, thereby increasing sales or user stickiness. Formally, Recall@k is defined as:

$$Recall@k = \frac{n_{hit}}{|I|} \tag{23}$$

where n_{hit} is the number of ground truth items that are among the top-k recommendation list, and $|I|$ denotes the total number of interactions.

5.1.3. Experiments details

The proposed DyCARS in this paper is implemented based on Pytorch and employs Xavier initialization [53] to initialize the model parameters. For all algorithms, we adopt 128-dimensional dynamic embeddings and randomly initialize the embeddings of users and items from a Gaussian distribution with mean 0 and variance 1. During the training process, we utilize the validation set to tune the hyper-parameters of DyCARS through Bayesian optimization.

For the LastFM dataset, we set the learning rate of the Adam optimizer to 1×10^{-4}, dropout rate to 0.8, and the length n of the user-item recent interaction history to 200. The number of layers L for the GaMLP encoder is set to 3, and we select max pooling as the pooling strategy for the AP network. The regularization coefficient α in the D-PCS loss function is set to 0.7, the number of negative samples is set to 10, the threshold M for filtering negative samples is set to 0.4, and the hyper-parameter τ is set to 1 to control the relative weight between positive and negative sample losses. For the GTD dataset, we set the dropout rate to 0.7 and the number of layers L for the GaMLP encoder to 1. Additionally, we choose average pooling as the pooling strategy for the AP network, and set the regularization coefficient α in the D-PCS loss function to 0.5. It is worth noting that, except for the adjustments mentioned above, the settings of the remaining hyper-parameters are consistent with those for the LastFM dataset.

During the testing phase, for a given real user-item interaction (u, v, t), DyCARS predicts the top-k items that user u will interact with at time t based on the dynamic interaction history of the user and item, denoted as $I_u(t^<, n)$ and $I_v(t^<, n)$, respectively. It should be emphasized that the prediction target of DyCARS is dynamic embedding rather than interaction probability, allowing us to predict the top-k items in the recommendation list through an effective nearest neighbor search strategy.

We carefully tune the parameters to ensure that all baseline models achieve optimal performance. All experiments are run independently in the same experimental environment with an Intel(R) Xeon(R) Platinum 8358P host and NVIDIA Tesla A40-48GB GPU.

5.2. Next item prediction experiment (Q1)

For next item prediction, we compared DyCARS with seven baselines on two datasets, and the results are presented in Table 4. The best and second-best results are highlighted in bold and underlined, respectively. "Improvement" denotes the performance improvement of DyCARS over the best baseline. Based on the experimental findings, we can draw the following conclusions:

1) On both datasets, DyCARS, DeePRED, and DGCF demonstrate superior performance compared to other baselines, indicating the crucial importance of modeling topological structure information for learning dynamic graphs.

2) DyCARS outperforms all baselines in terms of all metrics on both datasets. Particularly on the LastFM dataset, compared with the best baseline, the improvement of DyCARS on MRR is 39.95% and the improvement on Recall@10 is 28.88%. Furthermore, the performance of DyCARS gradually improves compared to state-of-the-art baselines on both the GTD and LastFM datasets, which aligns with the repetitive patterns observed in user behavior within the datasets. These substantial improvements can be attributed to two key factors. First, DyCARS leverages the recent interaction history of users and items as dynamic context, and models long-term dependency structures in the dynamic interaction history through the GaMLP encoder. Second, we introduce the AP network to

learn similarity scores between high-level features of user and item dynamic interaction histories, extracting the most relevant delayed interaction patterns from the historical sequences by calculating bidirectional attention weights. As a result, DyCARS is capable of effectively handling scenarios with low action repetition.

3) In DyCARS, there is a significant reduction in the gap between the MRR and Recall@10 metrics compared to other baselines. This indicates that DyCARS exhibits higher precision in ranking ground truth items, whereas other baseline models often only achieve high rankings within the top ten predicted items.

Table 4. Performance comparison of different methods on two datasets.

Method	LastFM		GTD	
	MRR	Recall@10	MRR	Recall@10
LSTM	0.062	0.119	0.124	0.211
Time-LSTM	0.068	0.137	0.267	0.452
RRN	0.089	0.182	0.402	0.643
DeepCovolve	0.019	0.039	0.051	0.085
JODIE	0.195	0.307	0.496	0.764
DGCF	0.321	<u>0.456</u>	0.509	<u>0.785</u>
DeePRED	<u>0.393</u>	0.416	<u>0.740</u>	0.756
DyCARS	0.550	0.588	0.810	0.822
Improvement	39.95%	28.88%	9.46%	4.71%

5.3. Detailed model analysis (Q2)

In this section, we will conduct a detailed analysis of the model to gain a deeper understanding of the impact of each core component of DyCARS on its overall performance. This analysis will focus on four key aspects: the influence of different encoders, the role of the gating mechanism, the contribution of the bidirectional attention mechanism, and the effectiveness of the L_{PCS} loss function. Through these analyses, our aim is to uncover the internal workings of DyCARS and how each component collaboratively enhances the performance of the recommendation system.

5.3.1. *Impact of different encoders*

Several variants of DyCARS were implemented, and comparative experiments on two datasets were conducted to demonstrate the effectiveness of the GaMLP encoder. The specific settings for these variants are as follows:

1) DyCARS-GRU: In this variant, the GaMLP encoder is replaced with a GRU [54]. GRU is a commonly used RNN variant for sequence modeling. It employs gate mechanisms that allow the network to selectively retain and update information when processing sequential data, enabling better capture of long-term dependencies.

2) DyCARS-Trm: In this variant, the GaMLP encoder is replaced with a Transformer model [55]. The Transformer model is a deep learning architecture based on self-attention mechanisms, specifically designed for handling sequential data. The self-attention mechanism allows the model to attend to different positions within the input sequence, capturing richer contextual information without relying

on complex recursive or convolutional structures. Additionally, the Transformer model supports parallel computation, greatly improving the efficiency of processing long sequential data.

According to the experimental results in Table 5, the following can be observed:

1) The performance of DyCARS-GRU is superior to DyCARS-Trm. This result may be due to several factors. First, models based on RNNs (such as GRU) often perform better on smaller datasets. Conversely, Transformer models typically require large-scale datasets to achieve optimal performance. Second, when dealing with dynamic recommendation problems, certain characteristics of GRUs may contribute to its more prominent performance. For instance, the adoption of gate mechanisms in GRUs allows for explicit storage and retrieval of user-item interaction history. Additionally, due to its recursive nature, GRUs can maintain the sequential order of input sequences. These two characteristics are crucial for addressing dynamic recommendation problems.

2) The performance of DyCARS surpasses that of variants using GRUs or Transformers, and several key factors can account for this. First, the GaMLP encoder utilizes an embedding matrix of the user's (or item's) recent *n* interaction histories as input. This embedding matrix concatenates the static embeddings of the most recent interacting items (or users) with the temporal differences between user-item interactions, thereby capturing user behavior patterns and variations in intrinsic item properties more comprehensively. It also effectively considers the time decay effect of recent interaction events on the current event. Second, the GaMLP encoder employs the SG module to capture the interactions between user-item dynamic interaction histories, enabling more effective exploration of correlations and patterns among interaction events, thus achieving modeling of long-term dependency structures in dynamic interaction histories. In contrast, the modeling capabilities of DyCARS-GRU and DyCARS-Trm may be constrained by their respective architectures, potentially limiting their ability to fully capture the complexity of dynamic interactions.

Table 5. Impact of different encoders.

Variants	LastFM		GTD	
	MRR	Recall@10	MRR	Recall@10
DyCARS-GRU	0.546	0.575	0.801	0.808
DyCARS-Trm	0.545	0.569	0.800	0.806
DyCARS	0.550	0.588	0.803	0.816

5.3.2. Impact of gating mechanisms

One notable feature of the GaMLP encoder is its adoption of the SG module to capture the interactions between user-item dynamic interaction history, enabling the modeling of long-term dependency structures within the dynamic interaction history. Table 6 demonstrates the impact of different gating mechanisms on model performance on two datasets. Taking the user side as an example, the mathematical representations of various gating mechanisms are as follows:

1) Variant *a* (Linear GaMLP): $\mathrm{SG}(Y_u(t)) = h_{U,b}(Y_u(t))$

2) Variant *b* (Additive GaMLP): $\mathrm{SG}(Y_u(t)) = Y_u(t) + h_{U,b}(Y_u(t))$

3) Variant *c* (Multiplicative GaMLP): $\mathrm{SG}(Y_u(t)) = Y_u(t) \odot h_{U,b}(Y_u(t))$

4) Variant *d* (Multiplicative, Split GaMLP): $\mathrm{SG}(Y_u(t)) = Y_u^1(t) \odot h_{U,b}(Y_u^2(t)), Y_u(t) = Y_u^1(t) \| Y_u^2(t)$

The model architectures of the four variants are illustrated in Figure 3.

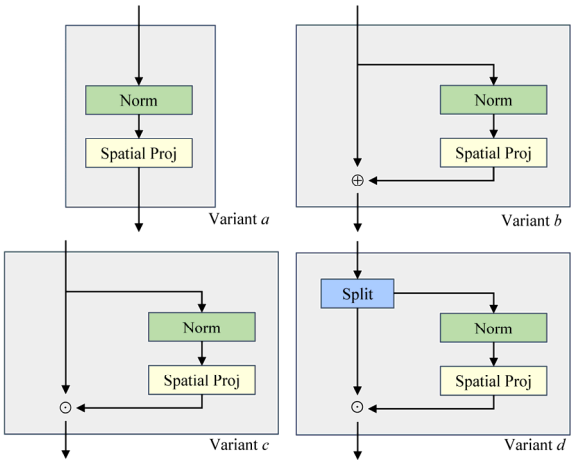

Figure 3. The model architectures of the four variants of SG modules.

Except for the variant *b* used on the GTD dataset in Section 5.2, variant *d* is employed throughout the rest of this paper. From Table 6, it can be observed that the impact of different gating mechanisms on model performance varies depending on the dataset. For the LastFM dataset, its low user action repetition and higher number of interaction events indicate that the dataset exhibits stronger dynamism and is more challenging to predict. In this scenario, variant *d* performs the best. This approach may have increased the complexity and flexibility of the model by splitting the input embeddings into two parts and processing them separately, enabling the model to better capture complex user behavior patterns and long-term dependencies. In contrast, for the GTD dataset, its higher user action repetition suggests a weaker level of dynamism, making the prediction relatively simpler. In this scenario, variant *b* performed the best, possibly due to the provision of moderate complexity offered by variant *b*.

Table 6. Impact of gating mechanisms.

Variants	LastFM		GTD	
	MRR	Recall@10	MRR	Recall@10
Linear GaMLP	0.546	0.575	0.803	0.817
Additive GaMLP	0.548	0.582	0.810	0.822
Multiplicative GaMLP	0.548	0.583	0.803	0.813
Multiplicative, Split GaMLP	0.550	0.588	0.803	0.816

5.3.3. Impact of bidirectional attention mechanism

DyCARS leverages the AP network to learn bidirectional attention weights for extracting the most relevant delayed interaction patterns from user-item interaction history, enabling the prediction of dynamic embeddings for users and items. To investigate the impact of the bidirectional attention mechanism on model performance, we conducted comparative experiments on two datasets. Specifically, "DyCARS w/o Att" means that we pool the high-level feature matrices of the recent interaction history of users and items by rows and columns, respectively, to directly generate dynamic

embeddings of users and items.

The results in Table 7 clearly demonstrate a significant decline in the performance of DyCARS across both metrics when lacking a bidirectional attention mechanism. This observation underscores the inadequacy of solely relying on pooling operations for extracting high-level feature matrices of users and items. Such a method fails to capture the complexity and dynamics in the user-item interaction history, potentially overlooking crucial delayed interaction patterns that have decisive impacts. Moreover, this finding underscores the critical role of bidirectional attention mechanisms in DyCARS.

Table 7. Impact of bidirectional attention mechanism.

Variants	LastFM		GTD	
	MRR	Recall@10	MRR	Recall@10
DyCARS w/o Att	0.545	0.570	0.796	0.808
DyCARS	0.550	0.588	0.803	0.816

5.3.4. Impact of L_{PCS} loss

Currently, most research on dynamic recommendation focuses primarily on constructing recommendation models, often overlooking the crucial role of the loss function in model learning. To investigate the impact of the L_{PCS} loss on model performance, we conducted comparative experiments on two datasets. Specifically, "DyCARS w/o L_{PCS}" denotes training DyCARS without the L_{PCS} loss by removing it from the D-PCS loss function.

Table 8 presents the experimental results. It should be noted that each model has been trained for sufficient iterations to achieve convergence, and the best results have been reported. The results demonstrate a significant improvement in both evaluation metrics when the L_{PCS} loss is included in the D-PCS loss function. This finding underscores the crucial role of the L_{PCS} loss in model parameter optimization and performance enhancement. By utilizing cosine similarity to compute the similarity between user-item pairs, the L_{PCS} loss avoids the issue of large value ranges that may arise in dot product or other methods. Additionally, the L_{PCS} loss allows the model to emphasize learning from hard negative samples, resulting in more discriminative features and enhancing the model's ability to extract useful information.

Table 8. Impact of $\boldsymbol{L_{PCS}}$ loss.

Variants	LastFM		GTD	
	MRR	Recall@10	MRR	Recall@10
DyCARS w/o L_{PCS}	0.549	0.582	0.800	0.812
DyCARS	0.550	0.588	0.803	0.816

5.4. Runtime experiment (Q3)

To empirically evaluate the efficiency of DyCARS, we conducted measurements to assess its runtime. In Figure 4, we present a comparison of the computational time (in seconds) required to complete an epoch using the GTD dataset. The following figure clearly demonstrates that DyCARS exhibits a significant advantage in terms of the computational time compared to recursive baseline models

such as JODIE and DGCF. Moreover, the computational time of DyCARS is comparable to that of the non-recursive baseline model, DeePRed. Recursive models typically incur additional computational costs as they rely on their previous states to predict dynamic embeddings. However, DyCARS circumvents this additional overhead by leveraging its non-recursive architecture that relies on interaction history rather than previous dynamic embeddings. Notably, DyCARS achieves both high efficiency and excellent predictive accuracy. This is primarily attributed to the powerful expressive capabilities of the GaMLP encoder, the AP network, and the optimized design of the D-PCS loss function.

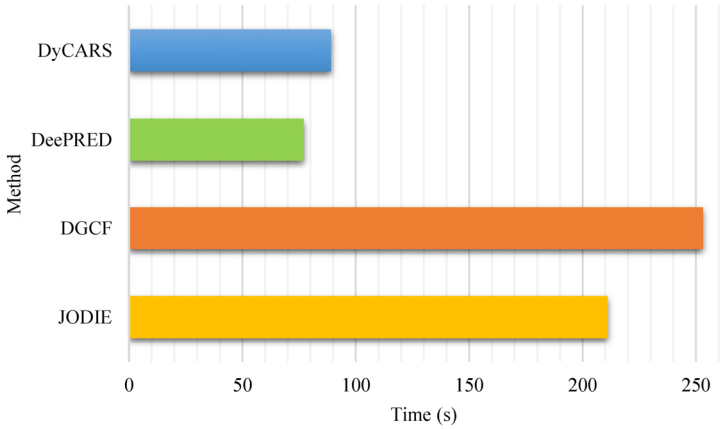

Figure 4. The computational time (in seconds) required to complete an epoch using the GTD dataset.

5.5. Diversity analysis in recommendations (Q4)

In practical applications, an enhancement in diversity within recommendation systems often accompanies a reduction in prediction accuracy. This phenomenon partially stems from the inherent uncertainty in user interests and the influence of certain unobservable factors in recommendation systems, which limit the real-time prediction capabilities for users' next actions. Moreover, most existing experimental simulations do not fully reflect the impact of real-time recommendation systems on user behavior. However, this does not imply an irreconcilable contradiction between diversity and accuracy. In fact, a key goal of recommendation systems is to guide user choices, rather than merely matching users' historical data. Increasing the exposure of different items in the recommendation list aims to provide users with a broader range of information, assisting them in discovering new items that may be of interest. Considering this, our study has adopted the widely used diversity metric in the field of recommendation systems, Diversity [56], to measure the degree of variation among items in the recommendation list. The formula for calculating this metric is as follows:

$$Div = \frac{2(K-1)}{K} \times \sum_{1 \le i \le j \le K} 1 - sim(i,j) \tag{24}$$

In this formula, K represents the length of the recommendation list, and $sim(i,j)$ denotes the cosine similarity between item i and item j.

Figure 5 displays the results of the diversity analysis of DyCARS on the GTD dataset. The experimental data indicate that, compared to the state-of-the-art baseline model DeePRED, DyCARS

not only maintains personalized recommendations in line with user interest patterns but also effectively enhances the diversity of the recommendations.

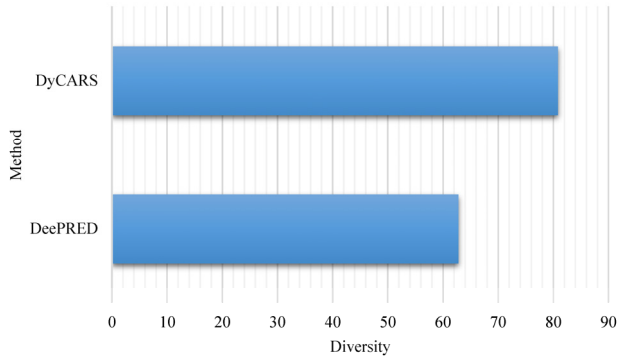

Figure 5. Analysis results of recommendation diversity on the GTD Dataset.

5.6. Parameter analysis (Q5)

In this section, we analyzed the impact of various hyper-parameters of DyCARS on the next item prediction. Throughout the subsequent experimental process, unless otherwise stated, we employ average pooling as the pooling strategy for the AP network on the LastFM dataset.

5.6.1. Impact of embedding size

Table 9 illustrates the impact of embedding size on model performance on two datasets. From the experimental results, it can be observed that for the LastFM dataset, increasing the embedding size yields improved performance. In contrast, for the GTD dataset, the model achieves optimal performance when the embedding size is set to 128. However, increasing the size of embeddings also results in increased computational complexity. Therefore, we need to choose a suitable embedding size to balance the model performance and efficiency.

Table 9. Impact of embedding size.

Embedding size	LastFM		GTD	
	MRR	Recall@10	MRR	Recall@10
32	0.544	0.569	0.800	0.813
64	0.544	0.570	0.803	0.814
128	0.546	0.572	0.803	0.816
256	0.548	0.577	0.799	0.816

5.6.2. Impact of the number of layers L in the GaMLP encoder

The experimental results in Figure 6 demonstrate the impact of the number of layers L in the GaMLP encoder on model performance, indicating that this effect is dependent on dataset characteristics. For the LastFM dataset, the model exhibits improved performance with an increase in

the number of layers L. This could be attributed to the fact that the LastFM dataset possesses more complex interaction dynamics and structures, requiring multiple layers of GaMLP encoders to capture deep-level interaction patterns and long-term dependencies. Each layer of the GaMLP encoder can extract different levels of information from the user-item interaction history, and the stacking of multiple layers aids the model in learning higher-order representations.

However, on the GTD dataset, the model achieves optimal performance when the number of layers is set to 1. This could be due to the lower complexity or smaller sample size of the GTD dataset. Therefore, increasing the model's complexity, such as adding more layers, may result in overfitting on the training data and a decrease in the generalization performance on the test data. Additionally, if the dataset has low dynamics, multiple layers of GaMLP may not bring additional performance improvement but instead increase computational costs and model complexity.

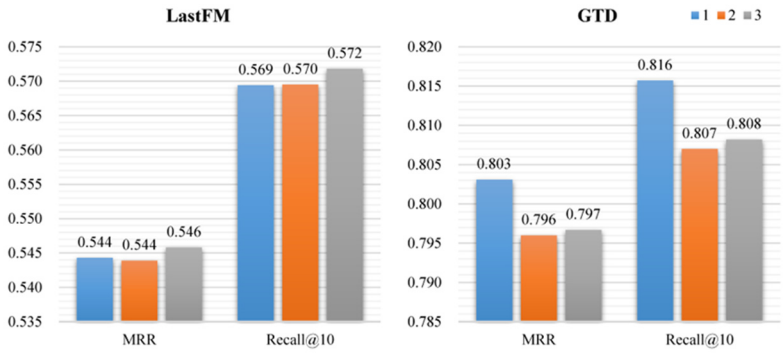

Figure 6. Impact of the number of layers L in the GaMLP encoder.

5.6.3. Impact of the length n of user-item recent interaction history

We conducted comparative experiments on two datasets to assess the effect of different lengths of recent user-item interaction history on model performance. As depicted in Figure 7, it can be observed that, as the length n of recent interaction history increases, the model's performance gradually improves on both datasets. This can be attributed to several potential reasons: First, appropriately increasing the length of recent interaction history allows the model to acquire richer contextual information. This helps the model better understand user behavior patterns and changes in interests, thereby making more accurate recommendations. Second, as the length n of recent interaction history increases, the model can encompass a wider time span, enabling it to effectively capture long-term dependency structures and better model the long-term trends in user behavior. Finally, in the short-term, user behavior may be influenced by certain transient factors such as emotions or seasonal activities, which can introduce noise. By increasing the history length, the model can consider user behavior across multiple time periods, reducing the impact of these transient factors and improving recommendation stability.

However, it is important to note that an excessively long length of recent interaction history may lead to decreased computational efficiency and introduce an abundance of irrelevant information, which can have a negative impact on the model's performance. Therefore, it is necessary to carefully select an appropriate length to strike a balance between model performance and efficiency.

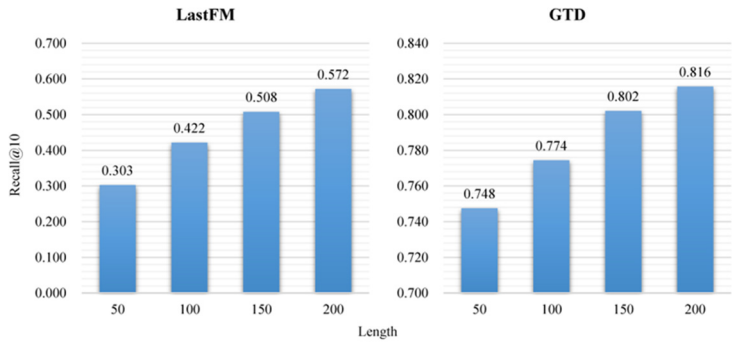

Figure 7. Impact of the length n of user-item recent interaction history.

5.6.4. Impact of different pooling strategies in the AP network

Comparative experiments were conducted on two datasets to assess the impact of various pooling strategies on model performance. As observed from Table 10, the max pooling strategy exhibits superior performance for the LastFM dataset. However, for the GTD dataset, the average pooling strategy demonstrates better performance. This indicates that the choice of pooling strategy directly influences the model's ability to capture and extract features. The effects of max pooling and average pooling may vary when dealing with datasets of different types.

Max pooling is a strategy that emphasizes the most salient features. The LastFM dataset features complex and diverse user-item interactions. Within this dataset, certain key features closely related to the prediction outcome play a significant role in predicting user behavior. Therefore, by selecting the maximum value, max pooling can capture these crucial features, thereby enhancing the performance of the model.

In contrast, average pooling takes into account the average effect of all features, enabling it to smooth the data and reduce the influence of noise and outliers. On the GTD dataset, there may not be significant key features, and the overall behavioral patterns are more crucial for prediction. In such cases, average pooling can capture the global behavioral patterns, leading to superior performance.

Table 10. Impact of different pooling strategies in the AP network.

Pooling strategy	LastFM		GTD	
	MRR	Recall@10	MRR	Recall@10
Max	0.550	0.588	0.797	0.810
Mean	0.546	0.572	0.803	0.816

6. Discussion

In this section, we first explore the potential impact of Session-Based Recommendation Systems (SBRSs) and the Dynamic Recommendation Systems (DRSs) used in this study on personalized recommendations, as well as their differences. Then, we discuss the prospects of applying recent large-scale time series language models in the field of DRSs. Additionally, we address the limitations of this study.

6.1. Session-based recommendation systems and dynamic recommendation systems

The potential impact and differences between SBRSs and DRSs on personalized recommendations are primarily evident in data processing and recommendation strategies. 1) Data processing: SBRSs typically handle session data, which consists of a series of interactions with clear start and end boundaries, focusing on intra-session interactions. DRSs, on the other hand, process continuous sequences of user behavior, emphasizing long-term user preferences and behavioral patterns. 2) Recommendation strategy: SBRSs concentrate on short-term user behavior and interests within the current session, which may lead to more immediate and current activity-related recommendations. DRSs, however, provide deeper levels of personalized recommendations by analyzing long-term data to understand a user's overall interests. Therefore, SBRSs focus more on short-term, immediate personalization, while DRSs emphasize the analysis of long-term user behavior and the depth of personalized recommendations.

In DyCARS, the following characteristics of the model design determine its potential impact on personalized recommendations: 1) Capturing dynamic interests: Through the dynamic graph and the GaMLP encoder, DyCARS can more accurately capture the evolving interests of users over time, thereby offering more tailored personalized recommendations that fit current needs. 2) Long-term dependency modeling: The model considers users' long-term interaction history, which aids in understanding and predicting their interests and preferences. 3) Real-time updates: As the model emphasizes dynamic interaction history, it can respond more swiftly to the latest user behaviors, providing real-time updated personalized recommendations. These features make DyCARS more precise and timely in delivering personalized recommendations, catering to the ever-changing needs of users.

6.2. Time series large language models and dynamic recommendation systems

Recent time series large language models, such as those based on the Transformer architecture, have provided significant insights for the field of DRSs. These models excel in processing long sequence data, particularly in capturing temporal dependencies and understanding complex contexts. In DRSs, these capabilities can be leveraged to analyze and predict users' behaviors and interests as they evolve over time more accurately.

Furthermore, the natural language processing capabilities of these models can also: 1) Analyze user-generated content, such as reviews or social media posts, to enrich user profiles and recommendation logic. 2) By analyzing the emotional tendencies in users' statements and interactions, DRSs can more accurately capture changes in user mood, thereby offering recommendations that better align with the user's current emotional state. 3) Language models can assist in identifying and analyzing trending topics in social media and online content, enabling recommendation systems to respond promptly to current hot topics and user interests. Therefore, combining research on time series large language models with DRSs could lead to new breakthroughs in personalized recommendations.

6.3. Limitations of this study

Despite the excellent performance of DyCARS in multiple aspects, we also recognize its limitations. 1) Dataset scale and complexity: The current DyCARS model has primarily been tested on datasets of limited scale. Its performance and efficiency may be challenged on larger and more

complex datasets. 2) Limited utilization of contextual information: Although the DyCARS model considers the dynamic interaction history of users and items, it does not fully utilize other potential contextual information, such as users' social networks, browsing behavior, and emotional states. 3) Challenges in real-time recommendations: In handling real-time data streams and rapidly changing user preferences, the model may require further optimization to improve response speed and accuracy.

7. Conclusions

This study proposes a novel non-recursive dynamic recommender model, called DyCARS, and successfully applies it to dynamic recommendation. DyCARS considers the dynamic interaction history of users and items as dynamic context. This approach utilizes the GaMLP encoder to capture long-term dependency structures in user-item interaction sequences. Additionally, we introduce the AP network to extract the most relevant delayed interaction patterns from the dynamic interaction history. Specifically, our model is built on dynamic graphs and leverages the static embeddings of recent user-item interactions as dynamic context. Additionally, we construct a GaMLP encoder to capture the long-term dependencies in the dynamic interaction history and extract high-level features. Furthermore, by applying the AP network, we learn similarity scores between the high-level features of users and items in the dynamic interaction history. By computing the similarity score matrix of high-level features, we obtain bidirectional attention weights and uncover the most relevant delayed interaction patterns from the historical sequence to predict the dynamic embeddings of users and items. Finally, we propose a D-PCS loss function suitable for dynamic recommendation, aiming to jointly optimize the static and dynamic embeddings of both types of nodes. We conducted extensive experiments on two real-world datasets, and the results demonstrate that DyCARS achieves state-of-the-art performance. In future research, we plan to integrate more fine-grained contextual factors into DyCARS, such as considering users' social network information, browsing behavior, and emotional states. The inclusion of these pieces of information may further enhance the accuracy and diversity of recommendations. Next, we aim to test and optimize the DyCARS model on larger and more diverse datasets to validate its performance and applicability in different scenarios. Moreover, in response to real-time data streams and rapidly changing user demands, we will explore new lightweight model architectures and real-time data preprocessing techniques to improve the response speed and accuracy of DyCARS.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by the Double First-Class Innovation Research Project for People's Public Security University of China (No.2023SYL08).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. Z. Lin, An empirical investigation of user and system recommendations in e-commerce, *Decis. Support Syst.*, **68** (2014), 111–124. https://doi.org/10.1016/j.dss.2014.10.003
2. T. Iba, K. Nemoto, B. Peters, P. A. Gloor, Analyzing the creative editing behavior of wikipedia editors: Through dynamic social network analysis, *Proc.-Soc. Behav. Sci.*, **2** (2010), 6441–6456. https://doi.org/10.1016/j.sbspro.2010.04.054
3. T. R. Liyanagunawardena, A. A. Adams, S. A. Williams, MOOCs: A systematic study of the published literature 2008–2012, *Int. Rev. Res. Open Distrib. Learn.*, **14** (2013), 202–227. https://doi.org/10.19173/irrodl.v14i3.1455
4. Q. Liu, S. Wu, L. Wang, T. Tan, Predicting the next location: A recurrent model with spatial and temporal contexts, in *Thirtieth AAAI Conference on Artificial Intelligence*, **30** (2016). https://doi.org/10.1609/aaai.v30i1.9971
5. S. Wu, Q. Liu, P. Bai, L. Wang, T. Tan, SAPE: A system for situation-aware public security evaluation, in *Thirtieth AAAI Conference on Artificial Intelligence*, **30** (2016). https://doi.org/10.1609/aaai.v30i1.9828
6. Z. Hou, X. Lv, Y. Zhou, L. Bu, Q. Ma, Y. Wang, A dynamic graph Hawkes process based on linear complexity self-attention for dynamic recommender systems, *PeerJ. Comput. Sci.*, **9** (2023), 1368. https://doi.org/10.7717/peerj-cs.1368
7. Y. Zheng, X. Yi, M. Li, R. Li, Z. Shan, E. Chang, et al., Forecasting fine-grained air quality based on big data, in *21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (2015), 2267–2276. https://doi.org/10.1145/2783258.2788573
8. X. Wang, X. He, M. Wang, F. Feng, T. Chua, Neural graph collaborative filtering, in *42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, (2019), 165–174. https://doi.org/10.1145/3331184.3331267
9. X. He, K. Deng, X. Wang, Yan Li, Y. Zhang, M. Wang, LightGCN: Simplifying and powering graph convolution network for recommendation, in *43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, (2020), 639–648. https://doi.org/10.1145/3397271.3401063
10. Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, *Computer*, **42** (2009), 30–37. https://doi.org/10.1109/MC.2009.263
11. D. Lee, H. S. Seung, Algorithms for non-negative matrix factorization, in *13th International Conference on Neural Information Processing Systems*, (2000), 535–541.
12. R. Salakhutdinov, A. Mnih, Probabilistic matrix factorization, in *20th International Conference on Neural Information Processing Systems*, (2007), 1257–1264.
13. S. Raza, C. Ding, Progress in context-aware recommender systems-An overview, *Comput. Sci. Rev.*, **31** (2019), 84–97. https://doi.org/10.1016/j.cosrev.2019.01.001
14. G. Adomavicius, K. Bauman, A. Tuzhilin, M. Unger, Context-aware recommender systems: From foundations to recent developments, in *Recommender Systems Handbook*, (2022), 211–250. https://doi.org/10.1007/978-1-0716-2197-4_6
15. C. Wu, A. Ahmed, A. Beutel, A. J. Smola, H. Jing, Recurrent recommender networks, in *Tenth ACM International Conference on Web Search and Data Mining*, (2017), 495–503. https://doi.org/10.1145/3018661.3018689

16. S. Kumar, X. Zhang, J. Leskovec, Predicting dynamic embedding trajectory in temporal interaction networks, in *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, (2019), 1269–1278. https://doi.org/10.1145/3292500.3330895
17. X. Li, M. Zhang, S. Wu, Z. Liu, L. Wang, P. S. Yu, Dynamic graph collaborative filtering, in *2020 IEEE International Conference on Data Mining*, (2020), 322–331. https://doi.org/10.1109/ICDM50108.2020.00041
18. Z. Kefato, S. Girdzijauskas, N. Sheikh, A. Montresor, Dynamic embeddings for interaction prediction, in *Web Conference 2021*, (2021), 1609–1618. https://doi.org/10.1145/3442381.3450020
19. S. Wu, F. Sun, W. Zhang, X. Xie, B. Cui, Graph neural networks in recommender systems: A survey, *ACM Comput. Surv.*, **55** (2022), 1–37. https://doi.org/10.1145/3535101
20. P. Covington, J. Adams, E. Sargin, Deep neural networks for YouTube recommendations, in *10th ACM Conference on Recommender Systems*, (2016), 191–198. https://doi.org/10.1145/2959100.2959190
21. H. Dai, Y. Wang, R. Trivedi, L. Song, Recurrent coevolutionary latent feature processes for continuous-time recommendation, in *1st Workshop on Deep Learning for Recommender Systems*, (2016), 29–34. https://doi.org/10.1145/2988450.2988451
22. Y. K. Tan, X. Xu, Y. Liu, Improved recurrent neural networks for session-based recommendations, in *1st Workshop on Deep Learning for Recommender Systems*, (2016), 17–22. https://doi.org/10.1145/2988450.2988452
23. Z. Wang, W. Wei, G. Cong, X. Li, X. Mao, M. Qiu, Global context enhanced graph neural networks for session-based recommendation, in *43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, (2020), 169–178. https://doi.org/10.1145/3397271.3401142
24. S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, T. Tan, Session-based recommendation with graph neural networks, in *AAAI Technical Track: AI and the Web*, **33** (2019), 346–353. https://doi.org/10.1609/aaai.v33i01.3301346
25. C. Xu, P. Zhao, Y. Liu, V. S. Sheng, J. Xu, F. Zhuang, Graph contextualized self-attention network for session-based recommendation, in *Twenty-Eighth International Joint Conference on Artificial Intelligence*, (2019), 3940–3946. https://doi.org/10.24963/ijcai.2019/547
26. B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, preprint, arXiv:1511.06939.
27. J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, J. Ma, Neural attentive session-based recommendation, in *2017 ACM on Conference on Information and Knowledge Management*, (2017), 1419–1428. https://doi.org/10.1145/3132847.3132926
28. Q. Liu, S. Wu, D. Wang, Z. Li, L. Wang, Context-aware sequential recommendation, in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, (2016), 1053–1058. https://doi.org/10.1109/ICDM.2016.0135
29. J. You, Y. Wang, A. Pal, P. Eksombatchai, C. Rosenburg, Hierarchical temporal convolutional networks for dynamic recommender systems, in *The World Wide Web Conference*, (2019), 2236–2246. https://doi.org/10.1145/3308558.3313747
30. Y. Wang, N. Du, R. Trivedi, L. Song, Coevolutionary latent feature processes for continuous-time user-item interactions, in *30th Conference on Neural Information Processing Systems*, (2016), 1–9.

31. Q. Wu, Y. Gao, X. Gao, P. Weng, G. Chen, Dual sequential prediction models linking sequential recommendation and information dissemination, in *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, (2019), 447–457. https://doi.org/10.1145/3292500.3330959
32. R. Trivedi, M. Farajtabar, P. Biswal, H. Zha, DyRep: Learning representations over dynamic graphs, in *International Conference on Learning Representations 2019 Conference*, (2019).
33. A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, et al., Latent cross: Making use of context in recurrent recommender systems, in *Eleventh ACM International Conference on Web Search and Data Mining*, (2018), 46–54. https://doi.org/10.1145/3159652.3159727
34. Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, et al., What to do next: Modeling user behaviors by time-LSTM, in *26th International Joint Conference on Artificial Intelligence*, (2017), 3602–3608.
35. Y. Zhang, X. Yang, J. Ivy, M. Chi, ATTAIN: Attention-based time-aware LSTM networks for disease progression modeling, in *Twenty-Eighth International Joint Conference on Artificial Intelligence*, (2019), 4369–4375. https://doi.org/10.24963/ijcai.2019/607
36. W. Kang, J. McAuley, Self-attentive sequential recommendation, in *2018 IEEE International Conference on Data Mining*, (2018), 197–206. https://doi.org/10.1109/ICDM.2018.00035
37. D. Hendrycks, K. Gimpel, Gaussian error linear units (GeLUs), preprint, arXiv:1606.08415.
38. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. Chen, MobileNetV2: Inverted residuals and linear bottlenecks, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (2018), 4510–4520. https://doi.org/10.1109/CVPR.2018.00474
39. Y. N. Dauphin, A. Fan, M. Auli, D. Grangier, Language modeling with gated convolutional networks, in *34th International Conference on Machine Learning*, **70** (2017), 933–941.
40. R. K. Srivastava, K. Greff, J. Schmidhuber, Highway networks, preprint, arXiv:1505.00387.
41. S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, *Neural Comput.*, **9** (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
42. J. Hu, L. Shen, G. Sun, Squeeze-and-excitation networks, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (2018), 7132–7141. https://doi.org/10.1109/CVPR.2018.00745
43. C. Santos, M. Tan, B. Xiang, B. Zhou, Attentive pooling networks, preprint, arXiv:1602.03609.
44. R. Hadsell, S. Chopra, Y. LeCun, Dimensionality reduction by learning an invariant mapping, in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, (2006), 1735–1742. https://doi.org/10.1109/CVPR.2006.100
45. S. Yang, W. Yu, Y. Zheng, H. Yao, T. Mei, Adaptive semantic-visual tree for hierarchical embeddings, in *27th ACM International Conference on Multimedia*, (2019), 2097–2105. https://doi.org/10.1145/3343031.3350995
46. X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, M. Wang, Lightgcn: Simplifying and powering graph convolution network for recommendation, in *43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, (2020), 639–648. https://doi.org/10.1145/3397271.3401063
47. C. Hsieh, L. Yang, Y. Cui, T. Lin, S. Belongie, D. Estrin, Collaborative metric learning, in *26th International Conference on World Wide Web*, (2017), 193–201. https://doi.org/10.1145/3038912.3052639

48. B. Fu, W. Zhang, G. Hu, X. Dai, S. Huang, J. Chen, Dual side deep context-aware modulation for social recommendation, in *Web Conference 2021*, (2021), 2524–2534. https://doi.org/10.1145/3442381.3449940
49. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in *26th International Conference on Neural Information Processing Systems*, **2** (2013), 3111–3119.
50. R. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, (2014), 580–587. https://doi.org/10.1109/CVPR.2014.81
51. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, *J. Mach. Learn. Res.*, **15** (2014), 1929–1958.
52. B. Hidasi, D. Tikk, Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback, in *2012th European Conference on Machine Learning and Knowledge Discovery in Databases*, (2012), 67–82.
53. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in *13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010*, (2010), 249–256.
54. J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, preprint, arXiv:1412.3555.
55. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, et al., Attention is all you need, in *31st International Conference on Neural Information Processing Systems*, (2017), 6000–6010.
56. T. Silveira, M. Zhang, X. Lin, Y. Liu, S. Ma, How good your recommender system is? A survey on evaluations in recommendation, *Int. J. Mach. Learn. Cybern.*, **10** (2019), 813–831. https://doi.org/10.1007/s13042-017-0762-9

AIMS Press