Dynamic Graph Recommendation: Collaborative Filtering in Dynamic Recommendation Scenario

Zeyu Zhang
Department of Statistics and Data Science
Northwestern University
Evanston, IL, 60208
`zeyuzhang2028@u.northwestern.edu`

Jonathan Lai
Department of Statistics and Data Science
Northwestern University
Evanston, IL, 60208
`JonathanLai2026@u.northwestern.edu`

Abstract

In this study, we address the challenge of learning dynamic graphs for modern recommendation systems. Unlike static graphs, real-world recommendation systems involve entities (users and items) and their relationships that continuously evolve over time. To provide up-to-date recommendations, these systems must be iteratively updated with incoming data, resulting in dynamic graphs. Transitioning to dynamic graphs introduces challenges such as instability, vulnerability, and the balance between updating and preserving information. To overcome these issues, we propose a new framework, *Dynamic Graph Recommendation (DGR)*, that enables RS to adapt in real-time to changes in the graph structure, including the addition, deletion, or modification of nodes and edges. Our innovations in dynamic Graph Neural Networks aim to enhance the stability and reliability of dynamic recommendation systems, ultimately improving recommendation accuracy and user satisfaction.

1 Introduction

Dynamic recommender systems [20, 18, 1] have demonstrated their effectiveness across various online applications, including social media, e-commerce, and streaming services [2, 16, 10]. These systems utilize historical interaction data between users and items to predict future user interactions. In real-world scenarios, user preferences and item popularity continuously evolve. Therefore, it is essential for dynamic recommendation models to capture these temporal dynamics to provide precise predictions accurately. Additionally, leveraging collaborative information—where users with common interactions exhibit similar interests—enhances recommendation accuracy. This approach, known as Collaborative Filtering (CF) [19, 5], is a cornerstone in recommendation systems. Thus, integrating the dynamic evolution of user preferences and item popularity with collaborative filtering is a critical objective in the development of dynamic recommender systems.

To build a dynamic recommender system, an intuitive approach is to model sequences of interactions. Various sequence-based methods have been developed, focusing on user-item interactions. For instance, RNN-based sequential prediction models [8, 15, 17, 23] leverage recurrent architectures to capture long-term dependencies in item sequences. Additionally, to incorporate user sequences, models such as Jodie [14] and other dynamic evolution models [27, 22, 24] employ dual RNNs to simultaneously model the updates of users and items based on their evolutionary processes. However, these models often fail to directly utilize collaborative information, as they primarily focus on the transition relationships between items and overlook user similarities. To address the lack of collaborative information, employing graph structures is a promising alternative for dynamic recommender systems.

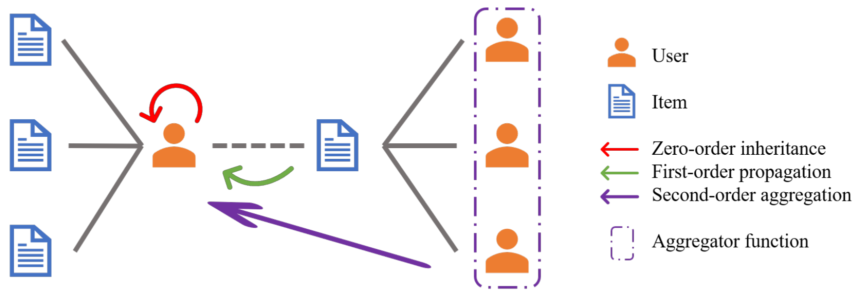

Figure 1: User-Item graph. A solid line means the user has interacted with the item, a dashed line means the user and the item are interacting. Arrows in different colors represent different relations when updating the embedding of a node. The aggregator function represented as a purple dashed box is used to capture second-order information.

Several studies have highlighted the benefits of using graph structures in recommender systems, despite their limitations. According to [21, 28], graph structures can explicitly incorporate collaborative information by representing user-item interactions as a bipartite graph. These models exploit the high-order connectivity of users and items, encoding collaborative information within the graph. However, they are typically suited for static scenarios and do not effectively utilize sequential dependencies or temporal information. Although SR-GNN [25] demonstrates the superiority of graph structures over sequence-based methods in dynamic recommendation contexts, it fails to account for the evolution of items. To address these shortcomings, we propose leveraging dynamic graphs to model the evolutionary processes in dynamic recommender systems, capturing both sequential dependencies and the evolving nature of user-item interactions.

In our proposed dynamic graph model, nodes represent users and items, while edges denote their interactions. Initially, the graph consists of isolated user and item nodes, but it evolves and expands as more users, items, and their interactions are added. To model this process and learn embeddings over time, we developed three update mechanisms, illustrated in Fig.1. The zero-order inheritance mechanism allows each node to retain its previous embedding and update it with new features. The first-order propagation mechanism connects interacting nodes, updating their embeddings simultaneously by propagating information from one to the other. The second-order aggregation mechanism uses aggregator functions to compute an overall embedding for all neighbors of a node on the user side, which is then passed to the item node, directly leveraging collaborative information. These mechanisms enable the dynamic graph to capture the evolving nature of user-item interactions and enhance recommendation accuracy.

Building on the three update mechanisms described above, we introduce *Dynamic Graph Recommendation (DGR)*, which integrates all three mechanisms within a unified framework. As illustrated in Fig.2, the DGR model comprises three modules corresponding to the three update mechanisms. Each module generates an embedding, which is subsequently fused to form the node's final embedding. To incorporate temporal information, we apply an evolutionary loss that considers timestamps, enhancing the model's recommendation accuracy. Detailed explanations of the model are provided in Section 3. Overall, DGR effectively learns user and item embeddings and performs recommendation tasks in an end-to-end manner.

Our main contributions in this paper are listed as follows:

- We design a novel framework for the dynamic recommendation task with graph structure, which can effectively model the dynamic relationship between users and items.
- We introduce the dynamic graph into dynamic recommendation scenarios to model the interactions and update users and items. Collaborative information is explicitly formulated and embedded.
- We conduct empirical experiments on the public dataset Gowalla. Experimental results demonstrate that our DGR model achieves state-of-the-art results.

The remainder of the paper is structured as follows: Section 2 presents the mathematical formulation of the problem. In Section 3, we detail our proposed method, DGR. Subsequently, we describe the

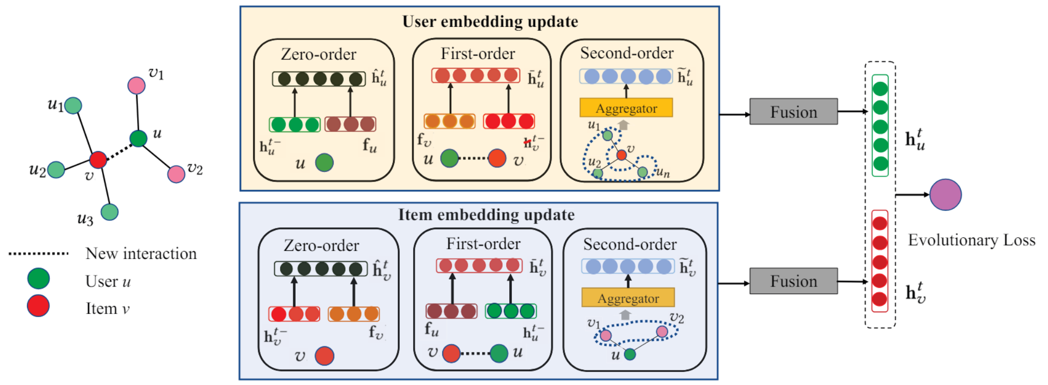

Figure 2: Framework of *Dynamic Graph Recommendation (DGR)*. The whole framework can be divided into two parts. *Left*: Original user-item graph. A new interaction denoted as a dashed line joins the graph. *Right*: Pipeline of DGR. First, the algorithm does *embedding updates*, where pink and light green nodes are the neighbors of the target nodes. $\mathbf{h}_u^{t^-}$ and $\mathbf{h}_u^{t^-}$ denote the embedding of user u and item v before timestep t, and $\hat{\mathbf{h}}_u^t$, $\bar{\mathbf{h}}_u^t$, and $\tilde{\mathbf{h}}_u^t$ refers to the zero, first, and second order information respectively. Then after the *fusion function*, we have embeddings $\mathbf{h}_u^t$ and $\mathbf{h}_u^t$ at current timestep t. With the *evolutionary loss*, we can get the recommendation in the end.

experimental setup, results, and analysis in section 4. Finally, we conclude the paper and suggest directions for future research in section 5.

2 Problem Definition

In this section, we formulate the *dynamic graph recommendation problem*. Compared to the previous static scenario, we need to add the time variable in the formulation. Specifically, we define *dynamic recommendation* and *dynamic graph* for our problem.

Dynamic recommendation Let U, V represent the user and item sets. In a dynamic scenario, the i^{th} user-item interaction is represented in a tuple

$$S_i = (u_i, v_i, t_i, \mathbf{f}_i), \quad i \in \{1, 2, \ldots, I\},$$

where I is the total number of interactions; $u_i \in U$ and $v_i \in V$ are the user and item in the interaction i; t_i is the timestep; $\mathbf{f}_i$ denotes the features of the interaction, and it includes user features $\mathbf{f}_u$ and item features $\mathbf{f}_v$. The target of dynamic recommendation is to learn the representations of both the user and the item from current interaction and historical records, and then use these representations to predict the most possible item that the user will interact with in the future.

Dynamic graph The interactions between users and items at timestep t construct a dynamic graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, where $\mathcal{V}_t, \mathcal{E}_t$ refer to the sets of nodes and edges in $\mathcal{G}_t$ respectively. Under recommendation settings, $\mathcal{V}_t$ contains all user and item nodes, and $\mathcal{E}_t$ has all interactions between users and items before timestep t. Essentially, the graph here is a *bipartite* graph, and all edges are between user and item nodes.

Specifically, we use $\mathbf{h}_u^t \in \mathbb{R}^d$ and $\mathbf{h}_v^t \in \mathbb{R}^d$ to represent the embedding of the user node u and the item node v at time t. The initial graph $\mathcal{G}_{t_0} = (\mathcal{V}_{t_0}, \mathcal{E}_{t_0})$ at time t_0 consists of isolated nodes, and the initial embeddings of users and items are the initial feature vectors or random vectors. Then while another interaction $S = (u, v, t, \mathbf{f})$ joins the graph, user and item embeddings $\mathbf{h}_u^t$ and $\mathbf{h}_v^t$ are updated by our DGR algorithm. Besides, $\mathbf{h}_u^{t^-}$ and $\mathbf{h}_v^{t^-}$ represent the most recent embeddings of user node u and item node v before timestep t.

3 Methodology

In this section, we present our approach, Dynamic Graph Recommendation (DGR). We begin by introducing the embedding update mechanisms and the recommendation modules. Following this, we briefly discuss the optimization process of DGR. Figure 2 provides an overview of our proposed method. As user-item interactions occur over time, they form a dynamic graph. Upon the arrival of a new user-item interaction, we update the embeddings of both the user and item nodes using the embedding update mechanism, detailed in Section 3.1. Subsequently, we project the user and item embeddings into the future using projection functions, as described in Section 3.2. Finally, we compute the L_2 distance between the predicted item embedding and all other item embeddings, recommending the items with the smallest distance to the predicted item embedding.

3.1 Embedding Updates

The embedding updates in DGR incorporate three key components: 1) *Zero-order inheritance*, where users and items update their embeddings by combining previous embeddings with new features; 2) *First-order propagation*, which involves updating embeddings based on the current user-item interaction; and 3) *Second-order aggregation*, which aggregates the embeddings of users who previously interacted with the item and integrates them into the current user's embedding, and vice versa.

Zero-order inheritance In a dynamic graph, the node to be updated first inherits the influence of its previous state and incorporates new features. Following existing sequential prediction methods [14, 8, 15], we use the previous embedding as part of the input to *inherit* the node's prior state. Additionally, we encode the time interval between the current and previous embeddings as part of the features to refine the user and item embeddings. For user embedding $\mathbf{h}_u$ and item embedding $\mathbf{h}_v$, the forward formulas can be expressed as follows:

$$\begin{aligned}\hat{\mathbf{h}}_u^t &= \theta_u\left(\mathbf{W}_0^u\mathbf{h}_u^{t^-} + \mathbf{w}_0\Delta t_u + \mathbf{W}_0^f\mathbf{f}_u\right)\\ \hat{\mathbf{h}}_v^t &= \theta_v\left(\mathbf{W}_0^v\mathbf{h}_v^{t^-} + \mathbf{w}_0\Delta t_v + \mathbf{W}_0^f\mathbf{f}_v\right),\end{aligned}$$

where Δt is the time interval between current time t and previous interaction time t^-; $\mathbf{h}_u^{t^-}$, $\mathbf{h}_v^{t^-}$ are the most recent embeddings of user u and item v before t; and $\mathbf{f}_u$ and $\mathbf{f}_v$ are the features of the user and the item. $\mathbf{W}_0 \in \mathbb{R}^{d\times d}$ are parameter matrices and $\mathbf{w}_0 \in \mathbb{R}^d$ is the parameter vector to encode time interval Δt. θ_u, θ_v are activation functions and we use `ReLU`.

First-order propagation In our model, we construct a dynamic bipartite graph to represent interactions between user and item nodes, where a user's immediate neighbors are the items they have interacted with, and vice versa. In dynamic recommendation scenarios, the items a user engages with reflect their current interests, while the users interested in a specific item help define its attributes. Therefore, leveraging first-order neighbor information is essential for accurately learning user and item embeddings. Unlike other models, our approach uniquely focuses on the currently interacted node, rather than all first-order neighbors, as input.

Specifically, when a user u interacts with an item v, the embedding and features of item v (such as reviews or descriptions) are incorporated into the embedding of user u. Conversely, the embedding and features of user u (such as the user's profile) are injected into the embedding of item v. The formal update process can be represented as follows:

$$\begin{aligned}\overline{\mathbf{h}}_u^t &= \phi_u\left(\mathbf{W}_1^u\mathbf{h}_v^{t^-} + \mathbf{W}_1^f\mathbf{f}_v\right)\\ \overline{\mathbf{h}}_v^t &= \phi_v\left(\mathbf{W}_1^v\mathbf{h}_u^{t^-} + \mathbf{W}_1^f\mathbf{f}_u\right),\end{aligned}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d\times d}$ are parameter matrices and ϕ_u, ϕ_v are the activation functions. With this first-order embedding, the features of the current interaction are propagated to update the connected user and item embeddings.

Second-order aggregation In the context of modeling collaborative filtering within a dynamic graph, node updates consider not only the historical sequences and direct interactions but also the structural information among nodes. The principle of second-order aggregation is designed to capture the collaborative relationships between users and items. A dynamic graph effectively models the influence of nodes at second-order proximity by accounting for the interactions passing through other nodes involved in these interactions.

Specifically, consider a user who has previously purchased several items and now buys a new item. We assume that the newly purchased item has a collaborative relationship with the user's previously purchased items. To model this relationship, we establish a direct connection, denoted as $v_u \to u \to v$. Here, $v_u \in \mathcal{H}_v^u = \{v_1, v_2, \ldots, v_m\}$, where $\mathcal{H}_v^u$ represents the set of items previously purchased by user u. In this scenario, u and v denote the user and the item in the current interaction, respectively. Intuitively, node u acts as a *bridge*, passing information from v_u to v, allowing v to receive aggregated second-order information through u. Similarly, for the user u, the relationship is represented as $u_v \to v \to u$, where $u_v \in \mathcal{H}_u^v = \{u_1, u_2, \ldots, u_n\}$, which is the set of users who previously purchased item v.

To make the second-order information flow from the neighbors of v to u, we take u as the *anchor node* and $u_i \in \mathcal{H}_u^v$ as second-order neighbor nodes in the graph. Then we use aggregator functions to transmit neighborhood information to the anchor node. This process is formulated as

$$\widetilde{\mathbf{h}}_u^t = \zeta_u \left(\mathbf{h}_u^{t^-}, \mathbf{h}_{u_1}^{t^-}, \mathbf{h}_{u_2}^{t^-}, \ldots, \mathbf{h}_{u_n}^{t^-} \right)$$

$$\widetilde{\mathbf{h}}_v^t = \zeta_v \left(\mathbf{h}_v^{t^-}, \mathbf{h}_{v_1}^{t^-}, \mathbf{h}_{t^-}, \ldots, \mathbf{h}_{v_m}^{t^-} \right),$$

where ζ_u and ζ_v are the aggregator functions. In the dynamic graph, users and items are distinct types of nodes with unique properties. For instance, a user's neighbors are the items they have interacted with, which are time-dependent, while an item's neighbors are the users who have interacted with it, often exhibiting similar characteristics. Additionally, popular items may have a significantly large number of interacting users. Consequently, the aggregator functions utilized should differ for user and item nodes. Below, we present several candidate aggregator functions suitable for second-order updates:

- *Mean aggregator* [6] is a straightforward operator to aggregate the neighbor information of user u and item v. It can be viewed as an inductive variant of the GCN [13] approach. The formulation can be written as follows:

$$\widetilde{\mathbf{h}}_u^t = \mathbf{h}_u^{t^-} + \frac{1}{|\mathcal{H}_v^u|} \sum_{u_i \in \mathcal{H}_u^v} \mathbf{W}_u^m \mathbf{h}_{u_i}^{t^-}$$

$$\widetilde{\mathbf{h}}_v^t = \mathbf{h}_v^{t^-} + \frac{1}{|\mathcal{H}_u^v|} \sum_{v_i \in \mathcal{H}_v^u} \mathbf{W}_v^m \mathbf{h}_{v_i}^{t^-},$$

  where $\mathbf{W}^m \in \mathbb{R}^d \times d$ are aggregation parameters.

- *LSTM aggregator* [9] is a sequential aggregator function based on the LSTM architecture, renowned for its robust non-linear memory capabilities, enabling it to capture long-term dependencies. From the user's perspective, previous interactions with items exhibit a clear sequential dependency. Thus, we input the user's neighboring items into the aggregator function in chronological order. Similarly, from the item perspective, connected users are ordered by the time of interaction and fed into the LSTM. The formulation of the LSTM aggregator is as follows

$$\widetilde{\mathbf{h}}_u^t = \mathbf{h}_u^{t^-} + \mathrm{LSTM}_u \left(\mathbf{h}_{u_1}^{t^-}, \mathbf{h}_{u_2}^{t^-}, \ldots, \mathbf{h}_{u_n}^{t^-} \right)$$

$$\widetilde{\mathbf{h}}_v^t = \mathbf{h}_v^{t^-} + \mathrm{LSTM}_v \left(\mathbf{h}_{v_1}^{t^-}, \mathbf{h}_{v_2}^{t^-}, \ldots, \mathbf{h}_{v_m}^{t^-} \right),$$

- *Graph attention aggregator* [3] calculates attention weights between the central node and its neighbors, indicating the importance of each neighbor to the central node. Drawing inspiration from the GAT model, we define the graph attention aggregator as follows

$$\widetilde{\mathbf{h}}_u^t = \sum_{u_i \in \mathcal{H}_u^v} \alpha_{ui} \mathbf{h}_{u_i}^{t^-}, \quad \widetilde{\mathbf{h}}_v^t = \sum_{u_i \in \mathcal{H}_v^u} \alpha_{vi} \mathbf{h}_{v_i}^{t^-}$$

where the attention weights α_{ui}, α_{vi} are defined as

$$\alpha_{ui} = \frac{\exp\left(\text{LeakyRelu}\left(\mathbf{W}_w\left[\mathbf{h}_u^{t-}\|\mathbf{h}_{u_i}^{t-}\right]\right)\right)}{\sum_{u_i\in\mathcal{H}_u^v}\exp\left(\text{LeakyRelu}\left(\mathbf{W}_w\left[h_u^{t-}\|\mathbf{h}_{u_i}^{t-}\right]\right)\right)}$$

$$\alpha_{vi} = \frac{\exp\left(\text{LeakyRelu}\left(\mathbf{W}_w\left[\mathbf{h}_v^{t-}\|\mathbf{h}_{v_i}^{t-}\right]\right)\right)}{\sum_{v_i\in\mathcal{H}_v^u}\exp\left(\text{LeakyRelu}\left(\mathbf{W}_w\left[\mathbf{h}_v^{t-}\|\mathbf{h}_{v_i}^{t-}\right]\right)\right)},$$

and $\mathbf{W}_w \in \mathbb{R}^{2d}$ refers to a weight matrix. $\|$ is the concatenation operation.

In implementation, we select a *fixed* number of neighbors for the second-order aggregation in case of enormous computational costs. We call the number of neighbor nodes selected as *aggregator size* and set it to 10 as default.

Fusion function To combine the three kinds of update mechanisms in learning node embeddings in the dynamic graph, we fuse the aforementioned three representations to obtain the final update formula:

$$\mathbf{h}_u^t = \mathrm{F}_u\left(\mathbf{W}_u^{\text{zero}}\hat{\mathbf{h}}_u^t + \mathbf{W}_u^{\text{first}}\overline{\mathbf{h}}_u^t + \mathbf{W}_u^{\text{second}}\widetilde{\mathbf{h}}_{\mathbf{u}}^{\mathbf{t}}\right)$$

$$\mathbf{h}_v^t = \mathrm{F}_v\left(\mathbf{W}_v^{\text{zero}}\hat{\mathbf{h}}_v^t + \mathbf{W}_v^{\text{first}}\overline{\mathbf{h}}_v^t + \mathbf{W}_v^{\text{second}}\widetilde{\mathbf{h}}_{\mathbf{v}}^{\mathbf{t}}\right),$$

where $\mathbf{h}_u^t, \mathbf{h}_v^t \in \mathbb{R}^d$ are the node embeddings updated after the user u interact with item v at timestep t. F_u, F_v are the fusion functions of the user and item respectively. Here we choose `ReLU` as the activate function. $\mathbf{W}_u^{\text{zero}}$, $\mathbf{W}_u^{\text{first}}$, and $\mathbf{W}_u^{\text{second}} \in \mathbb{R}^{d\times d}$ are parameters to control the influence of three update mechanisms.

3.2 Recommendation Modules

In dynamic recommendation scenarios, the objective is to forecast the item that a user is most inclined to interact with at time t, based on their historical interaction sequence preceding time t. Conceptually, this mirrors link prediction tasks in dynamic graphs. Specifically, we aim to anticipate the item node v in the dynamic graph that the user node u is most likely to connect with at time t. Motivated by prior work [14], we introduce an evolutionary loss tailored for dynamic graphs.

Evolution projection Diverging from conventional collaborative filtering approaches, our model is engineered to forecast future interactions. In particular, with a future time point as input, our model can generate predictions for future embeddings, facilitating recommendation generation. This setup offers enhanced flexibility as the predicted outcomes are not contingent on specific sequences but rather rely on embeddings learned from the dynamic graph structure.

Considering $\mathbf{h}_u^t$ as the predicted embedding of the user's future, it becomes necessary to establish an estimated future embedding to assess the accuracy of the prediction. Motivated by previous research [14], we posit that user growth is inherently *smooth*, implying that the embedding vector of the user node evolves continuously. Consequently, we introduce a projection function to estimate the future embedding, derived from the element-wise product of the previous embedding and the time interval. The embedding projection formula for user u from the current time t to the future time t^+ is defined as follows:

$$\widehat{\mathbf{h}}_u^{t^+} = \mathrm{MLP}_u\left(\mathbf{h}_u^t \odot \left(\mathbf{1} + \mathbf{w}_t\left(t^+ - t\right)\right)\right),$$

where $\mathbf{w}_t \in \mathbb{R}^d$ is the time parameter to convert the time interval to vector; $\mathbf{1} \in \mathbb{R}^d$ is a vector with all ones. With this projection function, the future item embedding grows in a smooth trajectory with respect to the time interval.

Once the projected embedding $\widehat{\mathbf{h}}_u^{t^+}$ for user u is obtained, we proceed to learn the future embedding of item v, denoted as $\widehat{\mathbf{h}}_v^{t^+}$, by employing another projection function. The projected item embedding is composed of three components: the current interacting user, the updated user features, and the item itself. Consequently, we define the projection formula for item v as follows:

$$\widehat{\mathbf{h}}_v^{t^+} = \mathrm{MLP}_v\left(\mathbf{W}_2\widehat{\mathbf{h}}_u^{t^+} + \mathbf{W}_3\mathbf{f}_u + \mathbf{W}_4\mathbf{f}_v\right)\Big)$$

where $\mathbf{W}_2, \mathbf{W}_3$ and $\mathbf{W}_4$ refer to the weights.

Loss function Upon obtaining the estimated future embeddings through projection functions, we utilize them as ground truth embeddings in our loss function. The training process of our model involves minimizing the Mean Square Error (MSE) between the model-generated embeddings $\mathbf{h}_u^t, \mathbf{h}_v^t$ and the estimated embeddings $\widehat{\mathbf{h}}_u^{t^+}, \widehat{\mathbf{h}}_v^{t^+}$ at each timestep t. Additionally, to prevent overfitting, we impose a constraint on the item embeddings. Specifically, we enforce the distance between the model-generated $\mathbf{h}_v^t$ and the most recent embedding $\mathbf{h}_v^{t^-}$ of item v, and between $\mathbf{h}_u^t$ and $\mathbf{h}_u^{t^-}$ of user u, to maintain consistency of embeddings. This constraint is grounded in the assumption that the properties of items and users remain relatively stable over short periods. Thus, the loss function is formulated as follows:

$$\mathcal{L} = \sum_{(u,v,t,f)\in\{S_i\}_{i=0}^{I}} \|\widehat{\mathbf{h}}_v^{t^+} - \mathbf{h}_v^t\|_2 + \lambda_u \|\mathbf{h}_u^t - \mathbf{h}_u^{t^-}\|_2 + \alpha_v \|\mathbf{h}_v^t - \mathbf{h}_v^{t^-}\|_2$$

where $\{S_i\}_{i=0}^{I}$ denotes the interaction events sorted by chronological order; λ_u and α_v are the penalty coefficients used to prevent the embedding of user and item from deviating too much during the update process.

To make recommendations for a user, we calculated the L_2 distances between the predicted item embedding that we obtained from the loss function and all other item embeddings. Then the nearest Top-k items are what we predict for the user. This recommendation method is more suitable for dynamic scenarios as it takes time into account. As a result, the changing trajectories for users and items are modeled by this time-variant loss, and it can make more precise recommendations for the next item.

3.3 Optimization and Training Process

Given the sequential nature of our model, the utilization of the back-propagation through time (BPTT) [4, 11] algorithm is essential, though it will slow down the training process. To speed up training, we adopt a batching method similar to that described in prior work [12]. The training algorithm must adhere to two key criteria: 1) it should process interactions within each batch simultaneously; 2) the batching process should preserve the original temporal order of the interactions, thus retaining the sequential dependency in the generated embeddings.

In practice, we arrange interaction events S_i in chronological order to form an event sequence $\{S_1, S_2, \ldots, S_I\}$ indexed by integers. Subsequently, we iterate through the temporally sorted sequence of interactions and allocate each interaction to a batch B_k, where $k \in [1, I]$. The batch construction process follows a specific order: initially, each batch is empty and indexed as -1. We denote $B_{\text{init}}(u) = B_{\text{init}}(v) = -1$. Upon adding each interaction $(u, v, t, \mathbf{f})$ to the batch, we update the batch index of both u and v. The index of the added batch for each interaction is determined as $\max\{B(u) + 1, B(v) + 1\}$. Consequently, upon adding the interaction to the batch, we update the indices of the involved u and v. This mechanism guarantees that the embeddings of users and items within the same batch are updated concurrently during both the training and testing processes.

4 Experiments

4.1 Experiments Setup

Dataset The Gowalla dataset, a widely-used benchmark in recommendation systems research, comprises user check-ins at various locations. This dataset is employed to assess the performance of our dynamic graph learning approach due to its rich temporal interactions between users and items (locations), making it ideal for examining the dynamic aspects of recommendation systems. In this dataset, each entry begins with a User ID followed by multiple Item IDs, each indicating a positive interaction between the user and the item. Given the absence of explicit temporal or timestamp information, we interpret the position of each consecutive interaction as its corresponding timestep. The statistics of the Gowalla dataset are detailed in Table 1.

Baselines To evaluate the performance of DGR, we compare it with the following 3 baselines, two static and one dynamic.

- *NGCF*[21] A state-of-the-art graph-based recommendation model, NGCF leverages the high-order connectivity in user-item interaction graphs to enhance recommendation accuracy.

Table 1: Statistics of the Gowalla dataset.

Dataset	#Users	#Items	#interactions
Gowalla	29,858	40,981	1,027,370

Table 2: Baselines Comparison. Our project compares our method (DGR) with three SOTA baselines: NGCF, LightGCN, and GraphSAIL. The **bold** numbers mean the best on each metric.

Model	NDCG@5	NDCG@10	NDCG@20	Recall@5	Recall@10	Recall@20
NGCF	0.208	0.218	0.233	0.036	0.073	0.147
LightGCN	0.160	0.191	0.232	0.007	0.015	0.030
GraphSAIL	0.218	0.232	0.246	0.039	0.078	0.156
DGR	**0.285**	**0.264**	**0.247**	**0.055**	**0.084**	**0.160**

Its advantage lies in its ability to capture complex user-item interactions through multiple layers of graph convolutions. However, it can be computationally intensive due to its intricate network structure and it assumes a non-dynamic graph structure. In our benchmarking, we configure NGCF with a 64-dimensional embedding, 64 hidden dimensions, a 3-layer architecture, a 0.5 dropout rate, a batch size of 1024, and a learning rate of 0.001.

- *LightGCN*[7] A simplified graph convolution network stemming from NGCF, LightGCN focuses solely on embedding propagation without unnecessary complexities. Its main advantage is its efficiency and effectiveness in capturing user-item interactions with reduced computational overhead. However, the simplification may lead to the loss of some interaction nuances. We configure LightGCN with a 64-dimensional embedding, a 3-layer architecture, a batch size of 1024, and a learning rate of 0.001 for our experiments.
- *GraphSAIL*[26] Incorporating graph structure awareness into the learning process for incremental learning, GraphSAIL effectively captures dynamic interactions in recommendation systems. It excels in adapting to changes in user-item interactions over time, but its increased complexity compared to other models results in a higher computational load. For our benchmarking, we set up GraphSAIL with a 64-dimensional embedding, 64 hidden dimensions, a 3-layer architecture, a 0.5 dropout rate, a batch size of 1024, and a learning rate of 0.001.

Evaluation Metrics We employ two evaluation metrics to assess model performance: Normalized Discounted Cumulative Gain (NDCG) and Recall. NDCG evaluates the ranking quality of recommendations by considering both the presence and the position of relevant items in the recommended list. Higher values indicate better performance in ranking relevant items higher. Recall measures the proportion of relevant items successfully retrieved out of the total relevant items available, with higher values indicating the system's effectiveness in recommending relevant items to users.

4.2 Results and Analysis

Baselines Comparison This experiment evaluates our Dynamic Graph Recommendation (DGR) method against three state-of-the-art baselines: NGCF, LightGCN, and GraphSAIL. We utilized NDCG and Recall at various cut-off points (5, 10, 20) as evaluation metrics. The results, presented in Table 2, indicate that GraphSAIL exhibited the best performance among the three state-of-the-art models. Notably, LightGCN significantly underperformed in Recall metrics while maintaining competitive NDCG values, suggesting that its top recommendations were highly relevant, but the quality of its broader recommendations was lower. Crucially, our DGR framework outperformed all baseline models in both NDCG and Recall metrics, underscoring its superior capability in generating relevant recommendations.

Ablation Study on Aggregator Functions This experiment examined the impact of different aggregator functions on the performance of the Dynamic Graph Recommendation (DGR) model. We evaluated three functions: MEAN, LSTM, and ATTENTION. The results, presented in Table 3, demonstrate that the ATTENTION aggregator consistently outperformed MEAN and LSTM across all

Table 3: Ablation study 1: study the effect of different aggregator functions. We compare three different aggregator functions: MEAN, LSTM, and ATTENTION. The **bold** numbers mean the best on each metric.

Aggregator	NDCG@5	NDCG@10	NDCG@20	Recall@5	Recall@10	Recall@20
MEAN	0.282	0.260	0.235	0.047	0.075	0.148
LSTM	0.280	0.262	0.238	0.052	0.078	0.154
ATTENTION	**0.285**	**0.264**	**0.247**	**0.055**	**0.084**	**0.160**

Table 4: Ablation study 2: study the effect of different levels of information aggregation. We compared three configurations with some level of information lost against the default DGR, which retains all information levels. Specifically, DGR-i means DGR without ith level information.

Aggregator	NDCG@5	NDCG@10	NDCG@20	Recall@5	Recall@10	Recall@20
DGR-0	0.281	0.263	0.245	0.054	0.080	0.158
DGR-1	0.280	0.263	0.244	0.053	0.078	0.154
DGR-2	0.279	0.261	0.240	0.053	0.078	0.152
DGR (default)	**0.285**	**0.264**	**0.247**	**0.055**	**0.084**	**0.160**

metrics. These findings suggest that the ATTENTION mechanism effectively captures collaborative information among second-order neighbor nodes, resulting in more accurate recommendations.

Ablation Study on Levels of Aggregation In this study, we evaluated the impact of different levels of information aggregation on the performance of the Dynamic Graph Recommendation (DGR) model. We compared three configurations against the default DGR, which retains all information levels: DGR-0 (excluding 1st-level information), DGR-1 (excluding 2nd-level information), and DGR-2 (excluding 3rd-level information). The results, presented in Table 4, reveal that the default DGR, which aggregates information at all levels, achieved the highest performance across all metrics. Notably, DGR-2 exhibited the lowest performance, underscoring the critical role of second-order information (collaborative filtering) in updating embeddings. These findings suggest that comprehensive aggregation of information at all levels is essential for optimal performance in dynamic graph recommendation systems, as it ensures a thorough representation and better adaptation to changes.

5 Conclusion and Future Work

In this paper, we introduce Dynamic Graph Recommendation (DGR), a novel framework that integrates dynamic graphs into recommendation systems. DGR features three distinct embedding update mechanisms designed for learning node embeddings and making accurate recommendations. Our experimental results demonstrate that DGR outperforms all current state-of-the-art baselines.

DGR represents an initial step in merging dynamic graphs with recommender systems, highlighting several avenues for future enhancement. Incorporating higher-order aggregation techniques could improve information integration. The growing trend of incremental learning in recommender systems suggests promising opportunities for combining our framework with incremental learning methods. Additionally, exploring various graph structures beyond the traditional user-item bipartite graphs, such as knowledge graphs, social networks, and attributed graphs, could further advance the capabilities and applications of dynamic graph-based recommender systems. These directions offer significant potential for enhancing the performance and versatility of DGR.

6 Acknowledgement

We are thankful to Professor Kaize Ding and Teaching Assistant Brody Kendall for their assistance with our project and the informative materials in the course. In addition, we are thankful to all the fellows discussing our project together.

References

[1] M Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys*, 55(7):1–38, 2022.

[2] Anitha Anandhan, Liyana Shuib, Maizatul Akmar Ismail, and Ghulam Mujtaba. Social media recommender systems: review and open research issues. *IEEE Access*, 6:15608–15628, 2018.

[3] PVGCA Casanova, Adriana Romero Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR. Petar Velickovic Guillem Cucurull Arantxa Casanova Adriana Romero Pietro Liò and Yoshua Bengio*, 2018.

[4] Orlando De Jesus and Martin T Hagan. Backpropagation algorithms for a broad class of dynamic networks. *IEEE Transactions on Neural Networks*, 18(1):14–27, 2007.

[5] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.

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

[7] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020.

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

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

[10] Hyunwoo Hwangbo, Yang Sok Kim, and Kyung Jin Cha. Recommendation system development for fashion retail e-commerce. *Electronic Commerce Research and Applications*, 28:94–101, 2018.

[11] Anil Kag and Venkatesh Saligrama. Training recurrent neural networks via forward propagation through time. In *International Conference on Machine Learning*, pages 5189–5200. PMLR, 2021.

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

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

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

[15] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428, 2017.

[16] Seth Siyuan Li and Elena Karahanna. Online recommendation systems in a b2c e-commerce context: a review and future directions. *Journal of the association for information systems*, 16(2):2, 2015.

[17] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. Context-aware sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1053–1058. IEEE, 2016.

[18] Chhavi Rana and Sanjay Kumar Jain. A study of the dynamic features of recommender systems. *Artificial Intelligence Review*, 43:141–153, 2015.

[19] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, pages 291–324. Springer, 2007.

[20] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)*, 54(7):1–38, 2021.

[21] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[22] Yichen Wang, Nan Du, Rakshit Trivedi, and Le Song. Coevolutionary latent feature processes for continuous-time user-item interactions. *Advances in neural information processing systems*, 29, 2016.

[23] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 495–503, 2017.

[24] Qitian Wu, Yirui Gao, Xiaofeng Gao, Paul Weng, and Guihai Chen. Dual sequential prediction models linking sequential recommendation and information dissemination. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 447–457, 2019.

[25] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 346–353, 2019.

[26] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. Graphsail: Graph structure aware incremental learning for recommender systems. In *Proceedings of the 29th ACM International Conference on Information and amp; Knowledge Management*, CIKM '20. ACM, October 2020.

[27] Jiaxuan You, Yichen Wang, Aditya Pal, Pong Eksombatchai, Chuck Rosenburg, and Jure Leskovec. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The world wide web conference*, pages 2236–2246, 2019.

[28] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S Yu. Spectral collaborative filtering. In *Proceedings of the 12th ACM conference on recommender systems*, pages 311–319, 2018.